Universidad ORT Uruguay

Facultad de Ingeniería

TuBocado:

Plataforma de venta de comida con sistema de fidelización

Entregado como requisito para la obtención del título de Ingeniero en Sistemas

Jimena Sanguinetti - 228322

Valentina Rostán - 194793

Federico Torres - 231465

Federico Charbonnier - 231301

Tutor: Álvaro Ortas

2024

Declaración de autoría

Nosotros, Jimena Sanguinetti, Valetina Rostán, Federico Torres y Federico Charbonnier declaramos que el trabajo que se presenta en esta obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizamos el proyecto de grado de la carrera de Ingeniería en Sistemas.
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad.
- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra.
- En la obra, hemos acusado recibo de las ayudas recibidas.
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros.
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.






Jimena Sanguinetti	Valentina Rostán	Federico Torres	Federico Charbonnier
20 marzo 2024	20 marzo 2024	20 marzo 2024	20 marzo 2024

Agradecimientos

Expresamos nuestra más grande gratitud a todos los que han colaborado en el desarrollo de nuestro proyecto de final de carrera para culminar nuestra carrera universitaria.

En primer lugar, agradecemos sinceramente a nuestras familias, amigos y a quienes nos han respaldado en todo el trayecto de nuestra formación académica y especialmente durante todo el proceso de esta tesis. Su apoyo ha sido fundamental para nuestro logro.

También queremos agradecer a nuestro tutor de la tesis, Álvaro Ortas, el cual nos brindó invaluable ayuda y dedicación en este proyecto. Nos dejó enseñanzas muy valiosas que nos permiten desarrollarnos tanto como profesionales como personalmente. Su guía a través de este complejo proyecto, fue fundamental al momento de tomar decisiones en momentos de dificultad.

A su vez, extendemos nuestro agradecimiento a la Universidad ORT Uruguay y a todos los docentes que participaron en nuestra formación académica, proporcionándonos los conocimientos necesarios para poder enfrentar los desafíos que nos enfrentamos en el camino y que nos vamos a enfrentar.

Por último, extendemos nuestro agradecimiento a los profesores de la cátedra de Ingeniería de *Software*, que actuaron como revisores en distintas etapas de nuestro proyecto: Rafael Bentancour, Dario Macchi y Pablo Hernández. Su retroalimentación fue esencial para llegar a esta etapa final con buena preparación.

A todos aquellos mencionados, les reiteramos nuestro más sincero agradecimiento por su invalorable colaboración y apoyo a lo largo de este trayecto académico y profesional.

Abstract

Dentro del marco de la industria de alimentos, más específico en la venta y entrega de los mismos, se enfrenta un desafío crucial: la necesidad de mejorar la interacción entre consumidores y restaurantes para garantizar una experiencia satisfactoria. Uno de los problemas importantes surge de los altos porcentajes que cobran las actuales plataformas de pedidos de comida en Uruguay, reduciendo así las ganancias de los restaurantes. En respuesta a estos problemas, la empresa AccesIt ha estado buscando experimentar un modelo innovador utilizando *marketing* multinivel para el servicio de pedidos de comida. De este modo junto a ellos, hemos creado el proyecto "TuBocado", una solución integral que aborda la problemática mediante una plataforma buscando abordar dichas problemáticas.

Una de las características distintivas de TuBocado es el enfoque de marketing multinivel, este ofrece beneficios significativos tanto para los consumidores como para los restaurantes asociados. A diferencia de otras plataformas existentes en nuestro país, este sistema permite a los consumidores ordenar comida y obtener recompensas por ello. Estos beneficios los denominamos "Bocados" y se generan por referir a otros usuarios y por realizar compras. Esto crea un poderoso incentivo para que los consumidores promocionen activamente la plataforma entre sus amigos y familiares, generando un efecto de red y aumentando la base de usuarios de manera orgánica, permitiendo reducir costos en publicidad. A su vez, este sistema de recompensas fomenta la lealtad de los consumidores, ya que ellos reciben beneficios adicionales cada vez que realizan una compra, lo que mejora la retención y la satisfacción del cliente a largo plazo.

El desarrollo del proyecto se ha dividido en dos etapas claves: la etapa de *Discovery*, donde se aplicaron metodologías de *Design Thinking* para identificar los requisitos y tecnologías adecuadas, y la etapa de *Delivery*, que implicó el desarrollo iterativo de la aplicación a lo largo de cuatro *Releases* basados en metodologías ágiles y algunas prácticas de BDD. El sistema fue desarrollado en NodeJs para el *backend* y ReactJs para el *frontend.*

Palabras clave

TuBocado, Scrum, Kanban, Behavior Driven Development (BDD), Design Thinking, Amazon Web Services (AWS), consumidor, dueño de restaurante, manager de TuBocado, backend, frontend, NodeJs, ReactJs, Typescript, API REST, Gherkin, Postgres.

Glosario

En esta sección mostraremos el glosario, una sección enfocada en comprender el vocabulario específico y los términos técnicos utilizados a lo largo de nuestro documento. Su propósito es clarificar y unificar el entendimiento de las palabras que aparecen en el texto, asegurando que todos los lectores, independientemente de su conocimiento previo sobre el tema, puedan seguir el contenido sin dificultades.

Aquí se encuentran definiciones precisas y explicaciones detalladas de conceptos críticos, acrónimos y jerga pertinente al proyecto. Para una consulta rápida y fácil, cada término está ordenado alfabéticamente y acompañado de una descripción. Para profundizar en la explicación de cada término y su aplicación dentro del contexto de nuestro trabajo, hemos detallado esta información en la sección A.1. Glosario.

ÍNDICE

1. Introducción..... 14
1.1. Estructura del documento..... 14
1.2. Presentación del equipo.....16
1.3. Presentacion del cliente..... 17
1.4. Elección de Proyecto.....18
1.5. Objetivos..... 18
2. Problema.....22
3. Solución.....26
3.1. Requerimientos..... 35
4. Arquitectura y diseño..... 46
4.1. Descripción de la arquitectura.....46
4.2. Decisiones de arquitectura.....64
4.3. Stack Tecnológico..... 70
4.4. Tácticas.....75
5. Proceso Gestión.....89
5.1. Introducción.....89
5.2. Discovery.....89
5.3. Selección de metodología.....91
5.4. Definición de requerimientos.....96
5.5. Criterios de aceptación vs prototipo..... 116
5.6. Roles..... 118
5.7. Selección de tecnologías.....119
5.8. Capacitaciones.....121
5.9. Delivery..... 125
5.10. Ejecución.....126
5.11. Gitflow.....152
5.12. Release Plan.....153
6. Gestión de riesgos.....156
6.1. Identificación de riesgos.....156
6.2. Estrategia utilizada.....157
6.3. Control y seguimiento.....159
6.4. Evaluación de riesgos..... 160
6.5. Resultados..... 164
7. Gestión de la configuración.....165
7.1. Desarrollo y manejo de repositorios.....165
7.2. Documentación..... 169
8. Aseguramiento de calidad..... 171
8.1. Definición de calidad..... 171
8.2. Estrategia seguida.....171
8.3. Actividades de aseguramiento de la calidad.....172
8.4. Pruebas.....176

8.5. Estándares..... 182
9. Métricas.....190
9.1. Cantidad de defectos graves pendientes.....190
9.2. Cantidad de defectos medios y leves pendientes.....191
9.3. NPS “¿Cuán probable es que recomiendes nuestro producto a un amigo o colega?”, con Product Owner.....192
9.4. NPS “¿Cuán probable es que recomiendes nuestro producto a un amigo o colega?”, con 10 Consumidores..... 194
9.5. NPS “¿Cuán probable es que recomiendes nuestro producto a un amigo o colega?”, con 5 Dueños de restaurantes..... 195
9.6. Índice de satisfacción del cliente con el producto..... 197
9.7. Índice de satisfacción del cliente con el equipo..... 198
9.8. Cumplimiento de las 10 heurísticas de nielsen.....199
9.9. Desviación en la estimación del tamaño.....200
9.10. Desviación en la estimación de esfuerzo.....202
9.11. Costo acumulado.....203
9.12. Grado de satisfacción (del 1 al 3) del equipo con el proceso de metodología ágil seleccionado..... 205
9.13. Aprender y aplicar correctamente nuevas metodologías claves de desarrollo durante la ejecución del proyecto.....206
9.14. Adquirir conocimiento y habilidades en las tecnologías clave utilizadas en el desarrollo de la aplicación, como ReactJs, NodeJs, Azure y AWS..... 207
9.15. Distribución del esfuerzo.....207
9.16. Cobertura de pruebas unitarias.....210
9.17. Retrabajo.....211
9.18. Velocidad.....212
9.19. Productividad.....214
10. Conclusiones.....216
10.1. Evaluación de Objetivos de Producto..... 216
10.2. Evaluación de Objetivos de Proceso.....222
10.3. Evaluación de Objetivos Personales.....225
10.4. Conclusiones generales.....227
11. Lecciones aprendidas.....228
11.1. Gestión..... 228
11.2. Ingeniería.....231
11.3. Tecnología..... 233
12. Pasos a seguir.....234
12.1. Validar modelo de negocio con Poc.....234
12.2. Expandir base de clientes..... 234
12.3. Sistema de facturación.....234
12.4. Implementar sistema de delivery.....235
12.5. Agregar promociones con Bocados..... 236
12.6. Manejar más de un restaurante con una única cuenta..... 236
Referencias bibliográficas..... 237
Anexo..... 244

A.1. Glosario....244
A.2. Vista Descomposición TuBocado.TuBocadoBE....252
A.3. Vista Layers TuBocado.PointsBE....255
A.4. Selección de tecnologías....256
A.5. Descripción de estructura de las APIs....265
A.6. Guía de deployment....268
A.7. Especificación de API servicio de puntos....274
A.8. Especificación de cumplimiento de heurísticas....280
A.9. Matrices de Pugh - Metodología....285
A.10. Entrevistas e inmersiones y los resultados obtenidos....285
A.11. Benchmarking aplicación de ventas de comida 1....300
A.12. Pretotipo....304
A.13. Prototipo....311
A.14. Validación con el cliente y pruebas con usuarios - pretotipo y prototipo....329
A.15. Miró completo....334
A.16. Product Backlog completo....335
A.17. Matrices de Pugh - tecnologías....359
A.18. Definition of Done de cada Release....360
A.19. Historias de usuario estimadas - hoja de cálculo....363
A.20. Resultado de Sprint Plannings....367
A.21. Retros....369
A.22. Riesgos identificados por cada release y por cada Sprint....378
A.23. Riesgos identificados a lo largo del proyecto....380
A.24. Análisis de riesgos....381
A.25. Evidencia de aplicación de estándares de calidad....384
A.26. Evidencia pruebas de aceptación con usuarios....389
A.27. Resultados encuesta “¿Cuán probable es que…” a P.O....401
A.28. Cálculo key result NPS “¿Cuán probable es que…” a P.O....402
A.29. Resultados encuesta “¿Cuán probable es que…” a Consumidores....402
A.30. Cálculo key result NPS “¿Cuán probable es que…” a Consumidores....405
A.31. Resultados encuesta “¿Cuán probable es que…” a Dueños de restaurantes....406
A.32. Cálculo key results NPS “¿Cuán probable es que…” a Dueños de restaurantes 408
A.33. Encuesta satisfacción del cliente con producto - 4 temáticas....409
A.34. Encuesta satisfacción del cliente con el equipo - 5 temáticas....410
A.35. Cálculo key result cumplimiento heurísticas de nielsen....411
A.36. Trackeo de horas mensuales....412
A.37. Encuestas de grado de satisfacción del equipo con metodología....413
A.38. Cálculo de retrabajo....413
A.39. Cálculo de la velocidad....413
A.40. Cálculo de la productividad....414
A.41. Evidencia de cobertura de código....414
A.42. Evidencia de prueba unitaria creada....417
A.43. Diagrama con un resumen de tecnologías usadas....418

Índice tablas

Tabla 1.5.1: Key Results objetivo de producto N°1.....19
Tabla 1.5.2: Key Results objetivo de proceso N°2.....21
Tabla 1.5.3: Key Results objetivo personal N°3.....21
Tabla 3.1: Descripción de roles de usuario y actores..... 27
Tabla 3.2: Descripción de distribución de porcentajes de costos propuestos.....30
Tabla 4.1.1: Catálogo de elementos de la vista de componentes y conectores..... 48
Tabla 4.1.2: Catálogo de elementos de la vista de layers.....55
Tabla 4.1.3: Nodos de la vista de asignación.....61
Tabla 4.1.4: Catálogo de elementos de vista de asignación.....61
Tabla 4.2.1: Tabla comparativa entre monolito y microservicios.....66
Tabla 4.3.1: Tabla comparativa proveedor de infraestructura.....72
Tabla 4.3.2: Tabla comparativa proveedor de framework de desarrollo de frontend..... 73
Tabla 4.3.3: Tabla comparativa proveedor de infraestructura de backend..... 75
Tabla 4.4.1: Tabla de requerimientos no funcionales con atributos de calidad asociados...76
Tabla 5.3.1: Tabla comparativa entre Kanban y Scrum..... 94
Tabla 5.6.1: Tabla de roles..... 119
Tabla 5.7.1: Ejemplo de Matriz de Pugh para lenguajes de frontend..... 121
Tabla 6.1.1: Ejemplo de aplicación del formato para riesgos.....157
Tabla 8.5.1: tabla de lineamientos para backend.....183
Tabla 8.5.2: tabla de lineamientos para backend.....184
Tabla 8.5.3: tabla de estándares personales..... 186
Tabla 8.5.4: tabla de resumen de estandares de documentacion..... 187
Tabla 10.1.1: Key Results objetivo de producto N°1.....217
Tabla 10.2.1: Key Results objetivo de proceso N°2.....222
Tabla 10.3.1: Key Results objetivo personal N°3.....225

Índice ilustraciones

Ilustración 1.2.1: Valentina Rostán..... 16
Ilustración 1.2.2: Federico Torres.....16
Ilustración 1.2.3: Jimena Sanguinetti.....16
Ilustración 1.2.4: Federico Charbonnier.....16
Ilustración 1.3.1: Logo de la empresa cliente AccessIt.....17
Ilustración 3.1: CANVAS del modelo de negocio.....31
Ilustración 3.1.1: Listado de restaurantes.....36
Ilustración 3.1.2: Categorías y platos.....36
Ilustración 3.1.3: Carrito de compra.....37
Ilustración 3.1.4: Confirmación compra..... 37
Ilustración 3.1.5: Proceso de compra..... 37
Ilustración 3.1.6: Bocados y referidos.....38
Ilustración 3.1.7: Datos personales.....38
Ilustración 3.1.8: Gestión de platos.....39
Ilustración 3.1.9: Gestión categorías..... 39
Ilustración 3.1.10: Finanzas restaurante.....40
Ilustración 3.1.11: Gestión de órdenes..... 40
Ilustración 3.1.12: Administración restaurantes registrados..... 42
Ilustración 3.1.13: Gestión parámetros.....42
Ilustración 3.1.14: Finanzas de cada restaurante.....43
Ilustración 3.1.15: Gestión usuarios.....44
Ilustración 4.1.1 : Diagrama de componentes y conectores.....47
Ilustración 4.1.2: Diagrama de secuencia de datos del usuario 1..... 52
Ilustración 4.1.3: Diagrama de secuencia de datos del usuario 2..... 52
Ilustración 4.1.4: Diagrama de vista layers.....54
Ilustración 4.1.5: Representación primaria de vista de usos..... 58
Ilustración 4.1.6: Diagrama de despliegue.....60
Ilustración 4.2.1: Diagrama inicial de arquitectura de microservicios..... 66
Ilustración 4.2.2: Diagrama inicial de arquitectura monolítica modular con un servicio de puntos externo..... 66
Ilustración 5.1.1: Cronograma de trabajo.....89
Ilustración 5.2.1: Tablero Kanban para tareas de Discovery..... 91
Ilustración 5.4.1: ciclo de design thinking..... 97
Ilustración 5.4.2: analizando estadísticas restaurante de sushi.....99
Ilustración 5.4.3: con el gerente del restaurante.....99
Ilustración 5.4.4: Barra de navegación de aplicación de venta de comida 1..... 103
Ilustración 5.4.5: Dashboard de aplicación de ventas de comida 1.....104
Ilustración 5.4.6: Resultados pregunta 1 encuesta.....105
Ilustración 5.4.7: Resultados pregunta 2 encuesta.....105
Ilustración 5.4.8: Resultados pregunta 3 encuesta.....106
Ilustración 5.4.9: Resultados pregunta 4 encuesta.....106

Ilustración 5.4.10: Resultados pregunta 5 encuesta....107
Ilustración 5.4.11: Resultados pregunta 6 encuesta....107
Ilustración 5.4.12: Ejemplo de descripción de funcionalidades....109
Ilustración 5.4.13: Ejemplos de las funcionalidades de la aplicación restaurante....110
Ilustración 5.4.14: Pretotipo de la vista: Restaurant - Datos generales....111
Ilustración 5.4.15: Prototipo V1 de la vista: Restaurante - Datos generales....112
Ilustración 5.4.16: Prototipo V1 historial de pedidos....113
Ilustración 5.4.18: Prototipo V3 historial de pedidos....114
Ilustración 5.4.19: Prototipo V4 historial de pedidos....114
Ilustración 5.4.20: Sección del User story map....115
Ilustración 5.4.21: Ejemplo de User Stories en el backlog....115
Ilustración 5.8.1: Tarjetas de prueba brindadas por Mercado Pago....123
Ilustración 5.8.2: Credenciales de prueba brindadas por Mercado Pago....123
Ilustración 5.10.1: Ciclo de desarrollo....126
Ilustración 5.10.2: Los tres tipos de Tags disponibles en Click Up y Jira....128
Ilustración 5.10.3: Ejemplo de Planning Poker....131
Ilustración 5.10.4: Planilla Excel con estimaciones....133
Ilustración 5.10.5: Ejemplo de resultado de Sprint Planning....134
Ilustración 5.10.6: Parte del Sprint Product Backlog....135
Ilustración 5.10.7: Clasificación del Sprint Backlog....136
Ilustración 5.10.8: Estado inicial de un Sprint....137
Ilustración 5.10.9: Ejemplo de archivo .feature en lenguaje Gherkin....138
Ilustración 5.10.10: Etapa de Requirements Definition....139
Ilustración 5.10.11: Definición de requerimiento y escenarios en Jira....140
Ilustración 5.10.12: Etapa de In Progress....141
Ilustración 5.10.13: Etapa de In Review....141
Ilustración 5.10.14: Etapa de Integration Testing....142
Ilustración 5.10.15: Etapa de Deploy....143
Ilustración 5.10.16: Etapa final de una user story....144
Ilustración 5.10.17: Sprint Risks representados en el tablero....146
Ilustración 5.10.18: Ejemplo de tarea en el tablero....148
Ilustración 5.10.19: Ejemplo de tablero en Retro Metro....151
Ilustración 5.10.20: Ejemplo de documento en Drive....151
Ilustración 5.12.1: Cronograma inicial de Delivery....154
Ilustración 5.12.2: Cronograma final de Delivery....154
Ilustración 6.2.1: Backlogs de riesgos....158
Ilustración 6.3.1: planilla evaluación de riesgos....159
Ilustración 6.3.2: gráfica de exposición de riesgos....160
Ilustración 6.4.1: Gráfica riesgos más relevantes....161
Ilustración 7.1.1: Backlogs de riesgos....165
Ilustración 7.1.2: estrategia de branching....167
Ilustración 7.1.3: ejemplos de ramas....168
Ilustración 8.4.1: evidencia cobertura de código....177
Ilustración 8.4.2: evidencia de cobertura de código....178

Ilustración 8.4.3: ejemplo de prueba de integración manual..... 180
Ilustración 8.5.1: ejemplo de pull requests de TuBocado.TuBocadoBE..... 188
Ilustración 9.5.2: ejemplo de descripción de pull request..... 189
Ilustración 8.5.3: ejemplo de revisión de código..... 189
Ilustración 9.1.1: Grafica cantidad de defectos graves..... 191
Ilustración 9.2.1: Grafica cantidad de defectos medios y leves..... 192
Ilustración 9.3.1: Gráfica NPS con P.O..... 193
Ilustración 9.4.1: Gráfica NPS con Consumidores..... 195
Ilustración 9.5.1: Gráfica NPS con Restaurantes..... 196
Ilustración 9.6.1: Gráfica índice satisfacción del cliente con producto..... 197
Ilustración 9.7.1: Gráfica índice satisfacción del cliente con equipo..... 199
Ilustración 9.8.1: Gráfica cumplimiento heurísticas de nielsen..... 200
Ilustración 9.9.1: Gráfica desviación estimación tamaño..... 201
Ilustración 9.10.1: Gráfica desviación estimación esfuerzo..... 203
Ilustración 9.11.1: Gráfica costo acumulado..... 205
Ilustración 9.12.1: Gráfica satisfacción del equipo con metodología..... 206
Ilustración 9.15.1: Gráfica distribución del esfuerzo..... 209
Ilustración 9.17.1: Gráfica retrabajo..... 212
Ilustración 9.18.1: Gráfica velocidad..... 213
Ilustración 9.19.1: Gráfica productividad..... 215

1. Introducción

El presente documento describe el proyecto final de carrera “TuBocado”, el cual es un requisito para la obtención del título de Ingeniería de Sistemas de la Universidad ORT Uruguay. El mismo fue realizado durante los meses de marzo 2023 hasta marzo 2024 por los estudiantes Jimena Sanguinetti, Valentina Rostán, Federico Torres y Federico Charbonnier, bajo la tutoría del docente Álvaro Ortas.

1.1. Estructura del documento

A continuación brindamos un breve resumen de lo que el lector podrá encontrar en cada uno de los siguientes capítulos del documento.

Problema

Detallamos el problema que aborda el proyecto, resaltando su importancia y respaldando su existencia con pruebas concretas.

Solución

Detallamos la solución que el proyecto brinda para abordar el problema mencionado, junto con los requisitos, el plan de negocio y sus funciones principales.

Arquitectura y Diseño

Explicamos cómo estructuramos y construimos el producto de *software* que le corresponde a este proyecto. Presentamos un análisis de la arquitectura del proyecto, junto con diagramas elaborados por el equipo para otorgar al lector un mayor entendimiento de la solución, basándonos en el modelo *Views and Beyond*.

Estrategia de Gestión del Proyecto

Detallamos cómo realizamos el proceso de investigación previa al inicio del desarrollo. Cuáles fueron nuestras incertidumbres iniciales, cómo las resolvimos y las decisiones realizadas para cada una.

Delivery

Describimos detalladamente la metodología de trabajo utilizada durante todo el desarrollo. Además realizamos énfasis en resaltar las decisiones, herramientas y técnicas que utilizamos para la gestión del proyecto.

Gestión de Riesgos

Mostramos cómo llevamos a cabo el manejo de los riesgos durante el proyecto. El objetivo que nos presentamos, el proceso que se utilizó, el impacto y el resultado obtenido de los riesgos que consideramos más importantes.

Gestión de la configuración

Detallamos los procesos y estándares utilizados para el desarrollo y el manejo de repositorios. Describimos todo el proceso, con el cual manejamos las tareas y cambios en el ámbito de desarrollo.

Aseguramiento de la calidad

Resaltamos las diversas actividades que utilizamos para asegurar la calidad, controlar los defectos y los tipos de pruebas implementados en el desarrollo.

Métricas

Mostramos las métricas usadas en el proyecto, comparamos resultado obtenido vs esperado, y detallamos las decisiones tomadas a partir de resultados.

Conclusión

Realizamos un análisis sobre el cumplimiento de los objetivos.

Lecciones aprendidas

Reflexionamos sobre lo aprendido a lo largo de todo el proceso de desarrollo.

Pasos a seguir

Describimos actividades a realizar a futuro con respecto al producto.

1.2. Presentación del equipo

Integrantes

El equipo de trabajo de este proyecto está integrado por cuatro estudiantes de la carrera de Ingeniería en Sistemas de la Universidad ORT Uruguay. Nos conocemos desde el inicio de la carrera y algunos desde el liceo. Hemos formado una amistad y llevamos varios proyectos de facultad juntos. Esto fue lo que nos llevó a realizar el proyecto final de carrera juntos. El equipo está conformado por:



Ilustración 1.2.1: Valentina Rostán



Ilustración 1.2.2: Federico Torres



Ilustración 1.2.3: Jimena Sanguinetti



Ilustración 1.2.4: Federico Charbonnier

Contamos con tres o más años trabajando como desarrolladores full stack:

- **Valentina Rostán**: desarrolladora *full stack* en Effectus Software, cuenta con amplia experiencia en NodeJs y ReactJs.
- **Federico Torres**: desarrollador *full stack* en Light-It, cuenta con amplia experiencia en php, NodeJs y ReactJs.
- **Jimena Sanguinetti**: desarrollada *full stack* en Moon Ideas, cuenta con amplia experiencia en php y javascript.
- **Federico Charbonnier**: desarrollador *full stack* en GlobalUy, cuenta con amplia experiencia en Ruby on Rails.

Adicionalmente el equipo también cuenta con un integrante en representación del cliente, el cual es Nicolas Rostán que actúa como *product owner* del proyecto.

Finalmente, el tutor del proyecto es Álvaro Ortas, cuenta con un Master en Computación y Sistemas de Información de la Universidad ORT Uruguay, Diploma en Computación y Sistemas de Información de la University of Oxford (Reino Unido). Coordinador del área de Ingeniería de Requerimientos, Laboratorio de Ingeniería de *Software*, Facultad de Ingeniería, Universidad ORT Uruguay, entre otros títulos.

1.3. Presentacion del cliente

AccessIt es una empresa de tecnología de la información con sede en Montevideo; fue fundada en 2018 y ofrece soluciones de infraestructura, *software* y servicios de consultoría para diversas empresas. Sus principales clientes se encuentran en Uruguay y Brasil, trabajó en varios rubros como salud, retail y área financiera. Está dirigida por Nicolas Rostán, el cual también es su fundador.



Ilustración 1.3.1: Logo de la empresa cliente AccessIt

La importancia del proyecto para el cliente es agregar un nuevo rubro a su portfolio e introducirse a un nuevo mercado, el cual es altamente competitivo. Para esto se aplicará toda la experiencia que fue adquiriendo el cliente en diversos proyectos, generando un producto con una entrada recurrente e incremental.

Este proyecto surge en base a una inquietud de un restaurante que llega a nuestro cliente pidiéndole una solución innovadora, dicho restaurante será el piloto de nuestro cliente. El éxito del proyecto no es crítico para la empresa pero es fundamental para incursionar en este nuevo rubro y lograr el objetivo de tener un aplicativo que les genere una entrada recurrente e incremental.

1.4. Elección de Proyecto

Comenzamos por buscar activamente temas para trabajar en la feria de proyectos de la ORT, sin embargo, no encontramos ningún proyecto que nos motivara suficiente para tomarlo como trabajo final de la carrera.

Debido a estos motivos nos pusimos a buscar en conocidos, nuestro lugares de trabajo, nuestro colegios, etc. Llegamos a dar con el Colegio Santa Margarita, el cual se veía frente a la necesidad de un sistema de gestión para sus alumnos, ya que no tenían. Nos reunimos con la directora del colegio para conocer más sobre sus necesidades y expectativas para obtener una idea de que podría ser este proyecto, pero por falta de desafío e innovación optamos por no realizarlo.

Buscábamos un proyecto que se adapte más a nuestro alineamiento, por eso optamos por realizar la propuesta planteada por AccessIt. Encontramos fascinante la oportunidad de contar con una empresa de *software* como cliente, dado que su experiencia en el sector y métodos de trabajo implicaría que tendríamos que cumplir con ciertos estándares, tanto en la calidad del código como en el proceso de desarrollo. Esto representaba una oportunidad para aprender y mejorar nuestras habilidades como desarrolladores profesionales además de aprender en sectores nuevos como el marketing multinivel (explicaremos más adelante este concepto).

1.5. Objetivos

Al iniciar el proyecto, establecimos una serie de objetivos que sirvieron como guía para el desarrollo y permitieron medir el progreso de manera efectiva. Optamos por utilizar la metodología de OKRs (*Objective and Key Results*) para gestionar estos objetivos, lo que nos proporcionó las herramientas necesarias para comprender claramente cuándo se completaba un objetivo o qué nivel de avance teníamos en un momento determinado del proyecto.

Además, en relación con la naturaleza de cada objetivo, determinamos tres clasificaciones distintas para categorizarlos: producto, proceso y personal. Cada objetivo se asigna a una categoría según si está vinculado al producto en sí, relacionado con la mejora del proceso utilizado o si está ligado a una motivación personal. A continuación se detallan los OKRs.

Producto

1. Entregarle un producto de calidad al cliente y que le proporcione un valor añadido

Key Results	Meta
a. Cantidad de defectos graves pendientes	0
b. Cantidad de defectos medios y leves pendientes	< 10
c. NPS “¿Cuán probable es que recomiendes nuestro producto a un amigo o colega?”, con ***Product owner***	>= 50%
d. NPS “¿Cuán probable es que recomiendes nuestro producto a un amigo o colega?”, con una muestra de **10 Consumidores**	>= 50%
e. NPS “¿Cuán probable es que recomiendes nuestro producto a un amigo o colega?”, con una muestra de **5 Dueños de restaurantes**	>=50%
f. Índice de satisfacción del cliente con el producto [Encuestas de satisfacción en áreas de enfoque del 1 al 5] i. Rendimiento y velocidad ii. Facilidad de uso iii. Cumplimiento de requerimientos iv. Evaluación general	>= 4
g. Índice de satisfacción del cliente con el equipo [Encuestas de satisfacción en áreas de enfoque del 1 al 5] i. Trato profesional y respeto ii. Puntualidad iii. Claridad en la comunicación iv. Compromiso del equipo v. Adaptación a los cambios	>= 4
h. Cumplimiento de las 10 heurísticas de nielsen	>= 80%

Tabla 1.5.1: Key Results objetivo de producto N°1

a. Este *Key Result* se orienta hacia la meta de mantener la cantidad de defectos graves en cero. Se define como defecto grave cualquier error o impedimento que obstaculice el uso adecuado de alguna funcionalidad del sistema. En ese sentido, nuestro objetivo es poder cumplir con la correcta implementación de las funcionalidades establecidas mediante cero defectos graves pendientes.
b. Este *Key Result* evalúa la cantidad de defectos de naturaleza media y leve que aún están pendientes. Un defecto de categoría media representa a aquel que se aleja de los requisitos o que presenta un comportamiento inesperado y un defecto de categoría leve se refiere a alguna imperfección básica y sencilla que, a pesar de su existencia, permite el uso de la funcionalidad. En este caso, manteniendo ambos tipos de defectos por debajo de 10 brinda un margen para dedicar tiempo a otros detalles.

c. Este *Key Result* se centra en alcanzar un “*Net Promoter Score”* (NPS) igual o superior al 50%, basado en una encuesta realizada al *product owner* con la pregunta "¿Qué tan probable es que recomiendes nuestro producto a un amigo o colega?". Al finalizar cada *Release*, se le acercaba la encuesta al *product owner* y el contestaba en base a su experiencia hasta el momento.
d. En este caso también se utiliza NPS como indicador *Key Result* con la misma meta, pero con el foco en los clientes. También luego de cada *Release* llevamos a cabo encuestas dirigidas a diversos usuarios, quienes compartieron sus experiencias después de participar en las pruebas guiadas.
e. En este *Key Result* usamos el índice NPS con el foco en dueños de restaurantes. Estos son con quienes hicimos la investigación al comienzo del proyecto, fueron parte de la etapa de obtención de métricas y son a quienes enviamos dichas encuestas luego de cada *Release*.
f. Este *Key Result* es índice de satisfacción del cliente enfocado al producto. En este caso se le realizó una encuesta con valores del 1 al 5 en 4 temáticas diferentes relacionadas al producto a medida que se fue desarrollando. La meta era un valor mayor o igual a 4, y la idea era medir qué tan satisfecho estaba el cliente con el producto evaluandolo desde diferentes enfoques.
g. Este *Key Result* hace referencia a otra encuesta, con valores del 1 al 5 pero en este caso con el foco en el equipo. Se buscaba medir qué tan satisfecho el cliente estaba en cualquier momento con el equipo. La meta se mantuvo y esto nos sirvió para tener un buen relacionamiento con el PO.
h. Para favorecer la usabilidad planteamos que el proyecto cumpla con las 10 heurísticas de Nielsen. La meta era mantener un valor mayor o igual a 80% y de esta forma poder medir qué tanto cumplimos con las heurísticas. Es decir, tener un índice relacionado con el nivel de la calidad de nuestro producto.

Proceso

2. Establecer un proceso eficiente y adecuado para el proyecto

Key Results	**Meta**
a. Desviación en la estimación del tamaño	No ascendente
b. Desviación en la estimación de esfuerzo	No ascendente
c. Costo acumulado	<= 3172 USD

d. Grado de satisfacción (del 1 al 3) del equipo con el proceso de metodología ágil seleccionado	> 2

Tabla 1.5.2: Key Results objetivo de proceso N°2

a. Este *Key Result* evalúa el porcentaje de desviación entre el tamaño real y estimado de las historias de usuario en cada *Release*. La meta fue mantener una tendencia no ascendente, mediante estimaciones más precisas.
b. Este *Key Result* expresa en porcentaje la desviación entre el esfuerzo real y el esfuerzo estimado, en horas-persona. Buscamos mantener una tendencia no ascendente y mejorar continuamente nuestra capacidad de estimación.
c. Este *Key Result* refleja el costo en dinero del desarrollo de TuBocado, se basa en la relación de 1 hora = 1 USD. Inicialmente, las estimaciones proyectaban un costo final de 2880 USD, pero luego se realizaron reajustes en el costo y la meta final se estableció en 3172 USD. Nuestra meta fue que el costo acumulado no supere dicho estimado.
d. Este *Key Result* evalúa el nivel general de satisfacción del equipo con la metodología ágil seleccionada. Para medirla, cada miembro votó en una encuesta después de cada *Release*, valores del 1 al 3 (1 para insatisfecho y 3 para muy satisfecho). La meta era que el valor resultante sea mayor a 2.

Personal

3. Aprender nuevas tecnologías

Key Results	**Meta**
a. Aprender y aplicar correctamente nuevas metodologías claves de desarrollo durante la ejecución del proyecto.	>= 2
b. Adquirir conocimiento y habilidades en las tecnologías clave utilizadas en el desarrollo de la aplicación, como ReactJs, NodeJs, Azure y AWS	>= 4

Tabla 1.5.3: Key Results objetivo personal N°3

a. La métrica consiste en aprender y aplicar bien 2 o más nuevas metodologías de desarrollo durante el proyecto. La meta fue analizar si se cumplió.
b. Esta métrica expresa que teníamos que adquirir conocimiento y habilidades técnicas sobre tecnologías relevantes con impacto en el proyecto como ReactJs, NodeJs, Azure y AWS. La meta fue realizar dicho aprendizaje con 4 o más.

2. Problema

Uno de los objetivos principales de cualquier restaurante es aumentar sus ventas, y es en este contexto, que los mismos tienen como meta agrandar su clientela y mantener este aumento en el tiempo, generando así una mayor rentabilidad sostenida. La industria de la alimentación se enfrenta a una variedad de desafíos que obstaculizan la capacidad de cada restaurante para lograr los objetivos. A continuación detallamos los desafíos que, junto con el cliente, identificamos como parte del problema que existe dentro de esta industria.

Competencia desigual: Los restaurantes de tamaño pequeño y mediano se encuentran en una posición de desventaja en comparación con las grandes cadenas y servicios de comida. Estos últimos cuentan con más y mejores recursos tanto financieros como logísticos para atraer comensales, mientras que los primeros no tienen la misma disponibilidad. En términos de *marketing*, esta disparidad se refleja en la capacidad de las grandes cadenas para ofrecer promociones o financiar campañas publicitarias, algo que los restaurantes más pequeños no pueden hacer. Para estos últimos, esto se traduce en una menor visibilidad y en el reto de poder destacarse entre su competencia y llegar a más clientes.

Dificultad para fidelizar a los clientes: Como mencionamos anteriormente, parte de los objetivos de los restaurantes es mantener sus niveles de clientela para lograr el éxito a largo plazo. En este sentido, muchos establecimientos enfrentan dificultades para fidelizar a sus clientes existentes y atraerlos con regularidad. Esto se debe a diversas razones, incluida la intensa competencia actual, los cambios constantes en las preferencias de los consumidores, los precios y las promociones, entre otros factores. Además, cualquier programa básico de fidelización que existe hoy en día puede tener primero, un costo muy alto, y segundo, puede no ser tan efectivo ya que no involucran tanto al consumidor final.

Costos elevados de aplicaciones de entrega de comida: Como parte de este mundo moderno y conectado, una de los medios más importantes de ventas para los restaurantes es a través de aplicaciones de entrega a domicilio. Esta forma de poder vender los productos se ha convertido en una herramienta indispensable en la

actualidad, facilitando a los consumidores hacer pedidos en cualquier momento y desde cualquier lugar, ya sea desde la oficina, el hogar, entre otros.

Sin embargo, dentro del marco de esa necesidad de participar de estas plataformas para poder mantenerse competitivos y vender más, los restaurantes se enfrentan a precios extremadamente altos en las principales aplicaciones que hoy operan en Uruguay. Si bien es clave contar con estos canales de venta para poder acceder al mercado y captar una amplia base de clientes, las tarifas que se les cobra a los establecimientos son porcentajes altísimos sobre las ventas haciendo que se reduzca el margen de ganancia significativamente.

De acuerdo con nuestro cliente, esta situación fue el principal desencadenante de la necesidad de crear una solución que logre sobrepasar este problema y reducir estos costos manteniendo los beneficios de participar de un mercado online y de fácil acceso para los restaurantes. El problema surge a raíz de una serie de conocidos del cliente, los cuales son dueños de restaurantes de sushi. Ellos se le acercaron y le plantearon esto como el mayor obstáculo al que se enfrentan hoy en día como proveedores del servicio de comida.

Necesidad de diferenciación: El mercado en el que operan los establecimientos de comida es altamente competitivo y es de suma importancia para los mismos poder destacarse sobre los demás. La falta de diferenciación puede llegar a significar una pérdida de clientes y como se mencionó antes, una disminución en las ventas. Debido a esa alta competencia, es muy difícil para los restaurantes poder diferenciarse ya que las herramientas disponibles para lograrlo son muy limitadas. Por lo tanto, es crucial que los restaurantes tengan herramientas efectivas para poder distinguirse y realizar estrategias de *marketing* y publicidad que hagan que lleguen a un mayor público.

Dificultad de tener un canal digital propio: La creación de un canal digital propio para la venta de comida presenta un desafío considerable debido a la complejidad y el costo involucrado. Para muchos restaurantes, especialmente aquellos sin un conocimiento técnico profundo, implementar y gestionar soluciones digitales propias resulta ser una tarea abrumadora. Las opciones disponibles en el mercado a

menudo se perciben como complicadas y fuera del alcance económico, haciendo que el desarrollo de un sistema personalizado sea especialmente engorroso.

Oportunidad de negocio

Frente a los desafíos mencionados, nuestro cliente identificó una oportunidad de negocio dentro del sector de comida basándose en el *marketing* multinivel. Este enfoque, basado en la creación de una red de usuarios que promueven productos y potencian la lealtad mediante recomendaciones directas, presenta una opción innovadora para fidelizar clientes y diferenciar a los restaurantes del resto utilizando la red de contactos y el boca a boca para promover los productos. En el contexto de nuestro proyecto, cuando un usuario invita a una persona a unirse a la red, este usuario es el referente y el nuevo usuario que se unió es el referido. Además, este es un modelo de negocio donde quienes participan generan ingresos (en este caso en puntos, los cuales a lo largo del proyecto nos referimos como Bocados) tanto por su compra directa, como por la compra de quienes se incorporen como referidos de ellos. Se aplicaría una red de referidos con diferentes niveles de generación de Bocados, que luego se podrían usar para comprar comida dentro del sistema. Esto fomenta la lealtad de los usuarios y estimula el aumento de clientes mediante recomendaciones personales. Este concepto y su implementación en el proyecto se explica en detalle en el capítulo de solución, a su vez en el glosario definimos cada uno de los roles mencionados en el capítulo.

De esta forma, el cliente nos planteó que ellos identificaron esta oportunidad que podría de alguna forma solucionar los desafíos reales que se presentaron antes. Es importante aclarar que el cliente ya tenía experiencia aplicando *marketing* multinivel en uno de sus productos de una aplicación para dar y participar de cursos *online*. Además, la idea de ellos con nosotros fue realizar un proyecto que pueda validar este modelo de negocio y que permita tomar esta oportunidad de negocio y probarla, y por esto es que nos vamos a referir al proyecto como una prueba de concepto o POC por su sigla en inglés “*Proof of Concept*”.

Dado que este proyecto se concibe como una POC, es fundamental comenzar en un mercado más limitado antes de expandirnos a escala. Junto con el tutor y el cliente concluimos, que el nicho del sushi ofrece un entorno óptimo para

probar nuestras ideas y validar el modelo de negocio en un contexto más manejable. Al limitar nuestro enfoque inicialmente, podemos adquirir información valiosa y ajustar el enfoque según los resultados obtenidos. La decisión de elegir el nicho de sushi fue basada principalmente en los contactos y relaciones que tenía el cliente con algunos dueños de restaurantes de sushi y su participación activa desde el inicio del proyecto. Esto garantizó una alineación con las necesidades reales del sector. Otra razón detrás de nuestra elección es mantener el alcance del proyecto manejable y realizable en el tiempo que manejamos. Concentrarnos en el nicho del sushi nos permite evitar la dispersión de recursos y esfuerzos en múltiples áreas, lo que nos permite dedicar atención y recursos adecuados para garantizar la entrega del proyecto.

Desafío

Basándonos en la idea inicial que tuvo con el cliente, aplicamos la metodología de *Design Thinking* para poder así obtener una versión enmarcada de un desafío que iba guiar el proyecto a lo largo del desarrollo. La implementación de esta metodología, incluyendo los procesos y resultados, se explican más adelante en el capítulo de *Discovery*.

Como resultado, obtuvimos la siguiente descripción de desafío: “¿Cómo podemos ayudar a mejorar la experiencia de compra de comida *online* para los clientes de los restaurantes de sushi de Montevideo y aumentar la rentabilidad para los restaurantes de sushi de Montevideo, teniendo en cuenta que existe competencia desigual, que hay dificultad para fidelizar a los clientes, que la comisión que cobran las aplicaciones existentes son muy elevadas y que hay necesidad de diferenciación entre restaurantes?”

A nivel general, el problema central que este proyecto busca afrontar es cómo superar los desafíos mencionados (competencia desigual, dificultad para fidelizar a los clientes, costos elevados de aplicaciones de entrega de comida, etc.) que afectan a la industria de la alimentación con una idea y una solución tecnológica innovadora a nivel de negocio. Nos planteamos como desafío específico mejorar la experiencia de compra de comida *online* y aumentar la rentabilidad para los restaurantes, considerando el contexto de problemas reales identificados.

3. Solución

Como parte fundamental del desarrollo del proyecto, realizamos un proceso de colaboración muy cercana con el cliente que nos permitió entender de una forma integral y abarcativa cuales eran sus objetivos y los desafíos que quería enfrentar. En este contexto, la solución que ideamos y desarrollamos es **TuBocado,** una plataforma SaaS (*software as a service*) con un doble objetivo.

Por un lado, ofrecer a los restaurantes una herramienta alternativa para su venta en línea que se distinga de las que ya existen por sus funcionalidades exclusivas. Esto incluye un sistema integrado de fidelización y costos más bajos en comparación con otras plataformas, entre otras ventajas. La idea de esta alternativa es solucionar en gran medida los desafíos mencionados anteriormente en el capítulo de problema. Por otro lado, busca ofrecer a los clientes de los restaurantes (consumidores finales) una experiencia de compra enriquecedora, agregando un valor adicional original y diferenciador.

Nuestra colaboración con el cliente comenzó a partir de su idea, inspirada en base a sus experiencias personales con dueños de restaurantes que conoce, particularmente de sushi. Su amplio conocimiento, tanto en el ámbito de negocios como en el tecnológico, lo llevó a proponernos esta idea. En el comienzo del proyecto, nos adentramos en un proceso de ideación en el marco de la metodología *Design Thinking* donde exploramos diversas ideas y enfoques en base a lo planteado por el cliente. Esta etapa fue clave al momento de establecer una base sólida y alinear las visiones del equipo y el cliente sobre el proyecto. A partir de ahí, realizamos un trabajo orientado a definir cómo estaría compuesta la solución considerando tener una sola o varias aplicaciones web, y relevando los requerimientos tanto funcionales como no funcionales.

El cliente quiso tomar el riesgo de probar una solución innovadora que sea usada para hacer frente a los desafíos mencionados en el capítulo de Problema. La solución está compuesta por tres módulos, que se terminan implementando como aplicaciones web, cada una diseñada para cumplir funciones específicas y dirigidas a diferentes roles de usuarios los cuales se explican en Tabla 3.1. En primer lugar, se diseñó el módulo de venta de comida que incorpora el modelo de *marketing*

multinivel. En segundo lugar se creó una aplicación web para dueños de restaurantes quienes gestionan toda la información relacionada al mismo. Por último, una tercera para los administradores de TuBocado, a los que denominamos como el tipo de usuario “Managers”. En la siguiente sección 3.1 Requerimientos explicaremos las funcionalidades principales de cada módulo/ aplicaciones web.

Rol de usuario / Actor	**Descripción**
Consumidor (Rol de usuario)	El consumidor representa el usuario final de la plataforma, cuyo principal objetivo es realizar compras de comida y participar en el programa de fidelización para acumular y canjear Bocados.
Dueño de restaurante (Rol de usuario)	Este tipo de usuario accede a la aplicación web de administración de restaurantes para gestionar integralmente la información relacionada con el funcionamiento de su establecimiento en la plataforma. Su responsabilidad incluye definir datos clave como los platos y categorías, horarios de apertura y cierre, ubicación del restaurante, y todo lo relacionado al manejo de las órdenes de comida que le entran, entre otros aspectos fundamentales para el correcto desempeño de su negocio en la aplicación.
Manager de TuBocado (Rol de usuario)	Este rol de usuario es el que va a usar nuestro cliente, encargado de la configuración de parámetros del sistema para la generación de Bocados y la gestión de referidos. Además, este usuario tiene un fuerte componente de monitoreo general de todos los restaurantes registrados y de acceso a la información generada en la plataforma de cada uno.
Referido (Actor)	En el contexto de que un consumidor A envía un link de invitación a un consumidor B y este se registra usando dicho link como invitado de A, el referido en este caso es el consumidor B.

Tabla 3.1: Descripción de roles de usuario y actores

Debido a la naturaleza exploratoria del proyecto (recuérdese que es una POC), su limitado alcance temporal considerando la fecha de entrega inamovible que establece la universidad como parte de las pautas del proyecto, y los recursos humanos disponibles, compuestos por cuatro desarrolladores que trabajan y estudian, la aplicación de venta de comida está específicamente enfocada en el nicho de sushi tal como se mencionó en el capítulo 2. Problema. Esto se debe a que al reducir el alcance a un nicho específico, se minimizan los contextos y las potenciales variables que podrían impactar en este tipo de negocio, y de esta forma, se concentran los recursos y esfuerzos en la creación de una solución más eficaz. Además, esta focalización facilita la obtención de retroalimentación y validación más rápidas y pertinentes por parte de los actores involucrados, como se evidencia en las pruebas, en comparación con un enfoque más generalizado.

Marketing Multinivel

Para poder entender cómo funciona la aplicación y cómo soluciona los problemas identificados, primero explicaremos que es *marketing* multinivel. Por definición: *"El marketing multinivel es una estrategia de marketing o un modelo de negocio que se estructura por medio de una red de contactos que contribuyen a generar un mayor nivel de ventas."* [8]

Según Amway, una de las empresas que aplica el modelo de *marketing* multinivel más grande del mundo, este es un método muy exitoso y efectivo de compensar a vendedores y consumidores por la comercialización, compra o distribución de productos y servicios.[11]

Algunas de las características principales de este son las siguientes:

- Reducción de costos de publicidad
- Viralidad (Expansión rápida y escalable)
- Mejora de la retención de clientes
- Mayor alcance de mercado
- Retroalimentación directa del mercado
- Fomento de la lealtad de marca

En otras palabras, el *marketing* multinivel, se basa en crear una red de personas en donde estas venden, refieren y compran productos o servicios, ganando comisiones por sus propias ventas y compras, o de las personas que ellos refieren. Se basa en la construcción de relaciones personales y el aprovechamiento del boca a boca para promover los productos.

La diferencia clave del marketing multinivel con otros esquemas que también aplican un sistema de referidos, es que este se enfoca en la compra o venta de un producto en sí. Normalmente a la venta de un producto se le gana un X%, pero con el *marketing* multinivel, en lugar de quedarse con toda la ganancia, se queda con una menor ganancia y el resto se usa para comisionar a la red.

En este contexto, la aplicación desarrollada permite a las personas acumular puntos y utilizarlos luego para canjearlos por comida dentro del mismo sistema. En

el proceso de personalización de este modelo para el proyecto, el cliente denominó Bocados a los puntos. Existen dos casos en los que un usuario genera estos Bocados y se explican a continuación.

En primer lugar, un usuario con el rol de Consumidor (consultar Tabla 3.1) puede acumular Bocados al efectuar compras de comida. Una vez que el pedido se marca como entregado, los Bocados se asignan al consumidor, quien puede canjearlos de inmediato en la aplicación. En segundo lugar, los consumidores pueden ganar Bocados cuando alguno de sus referidos (ver Tabla 3.1), ubicado dentro de los niveles configurados, realiza una compra. Los consumidores pueden invitar a otras personas a unirse a la plataforma y cuando estas personas compran, le generan puntos a dicho consumidor. La particularidad es que este sistema de recompensas se aplica no solo a los referidos directos, sino también a los referidos de sus referidos. Esto sucede en un máximo de niveles configurado por el cliente.

Respuesta al problema

La solución propuesta ofrece una forma de abordar los desafíos identificados anteriormente en el capítulo 2.Problema. Inicialmente, el enfoque de nuestro cliente estuvo en incorporar restaurantes de sushi pequeños y medianos en Montevideo, para luego extenderlo a otros tipos de establecimientos y, finalmente, segmentarlos por zona o región. Esto va a permitir una estrategia de crecimiento enfocada y adaptable a diferentes mercados, lo cual se detalla en el capítulo 12. Pasos a seguir.

Esto baja sustancialmente el costo del canal de venta utilizado. Actualmente, según los resultados que obtuvimos de la investigación inicial del proyecto, donde analizamos las plataformas que operan hoy en día en el Uruguay a través de entrevistas a dueños de restaurantes que las utilizan, pudimos relevar que dichas plataformas existentes cobran alrededor de un 23% de lo generado, mientras que la solución propuesta, cobraría significativamente menos. A continuación se adjunta una tabla (Tabla 3.2) con una síntesis de los porcentajes previstos en la plataforma.

Concepto	**Porcentaje (%)**
Ganancias	3
Distribución red de premios	3

Costos pago con efectivo o tarjeta	0 - 5
Costo delivery	6
Total	12 - 17

Tabla 3.2: Descripción de distribución de porcentajes de costos propuestos

Esto resulta en un total de costos que oscila entre el 12% y el 17%, en comparación con el 23%, mejorando así los márgenes de ganancia de los restaurantes.

Por otro lado, la aplicación de *marketing* multinivel, como característica distintiva de esta solución, logra resolver los siguientes problemas:

- **Competencia desigual:** Aunque en la plataforma pueden coexistir diferentes restaurantes, el enfoque en la reducción de costos y la implementación del *marketing* multinivel ayuda a nivelar el campo de juego. Los restaurantes pequeños y medianos, que tradicionalmente no pueden competir en términos de presupuesto de *marketing* y promociones, obtienen una mayor visibilidad sin necesidad de una gran inversión financiera.
- **Fidelización de clientes:** La estructura de incentivos y recompensas mediante Bocados promueve la retención y lealtad de los clientes, animándonos a continuar usando la aplicación y comprando en los restaurantes asociados.
- **Necesidad de diferenciación:** La plataforma proporciona a los restaurantes las herramientas para destacarse a través de estrategias de *marketing* y publicidad innovadoras, mejorando su visibilidad y atractivo en un mercado saturado sin la necesidad de gastar en publicidad ni *marketing*.
- **Canal digital propio:** Facilita a los restaurantes la creación de un canal de venta directa sin las complicaciones tecnológicas o financieras que implica desarrollar una solución independiente. Esto les permite tener un control más directo sobre su oferta, comunicación y relación con los clientes.

Modelo de negocio

Finalmente construimos un CANVAS para representar el modelo de negocio acordado con nuestro cliente.



Ilustración 3.1: CANVAS del modelo de negocio

El Canvas proporciona una visión integral de la plataforma SaaS donde se remarca el énfasis del sistema en *marketing* multinivel para incentivar la participación de consumidores. Este modelo de negocio se centra en la optimización de la experiencia de los consumidores y la expansión del alcance de los restaurantes asociados, a través de una plataforma tecnológica y estrategias de *marketing*, respaldado por un equipo dedicado de desarrollo y soporte técnico.

Socios clave:

Uno de los socios clave en este modelo de negocio van a ser los restaurantes de sushi de Montevideo, quienes proporcionan la oferta principal de productos a través de la plataforma. Además, los proveedores de servicios de pago en línea se encargaran de asegurar la viabilidad de las transacciones seguras. También el personal de cadetería que es el encargado de la gestión de la entrega de los pedidos. Otro de los elementos cruciales van a ser consumidores de la aplicación, lo cual es crucial mantener satisfechos con el uso del sistema. Ellos no solo van a generar ingresos directos mediante sus compras, sino que también actuarían como los principales promotores, impulsando la expansión de la plataforma a través de referencias y recomendaciones a sus conocidos.

En el marco de esta identificación de socios clave, entendimos que los restaurantes y consumidores, deberían tener un manejo personalizado y enfocado en sus tareas dentro de la aplicación para que pudieran cumplir con su rol y propósito de manera satisfactoria. Esta es una de las principales razones por las que tomamos la decisión de dividir la solución en diferentes aplicaciones web diferentes basándonos en roles. Sumado a esto, se identifica nuestro cliente como gestor (*Manager*) de algunas variables del sistema y supervisor de estadísticas y finanzas de todo el sistema, y en este sentido, fue que decidimos dividir en tres la cantidad de aplicaciones según el rol de cada uno.

Actividades clave:

Las actividades clave abarcan desde el desarrollo inicial y continuo del sistema hasta la implementación de rigurosos procesos de testing y QA (*Quality Assurance*) para garantizar su funcionalidad y seguridad. Asimismo, la adquisición constante de restaurantes y usuarios va a ser esencial para mantener un flujo de oferta y demanda en la plataforma, promoviendo así su crecimiento y diversificación. Creemos que la atención a estas actividades va a lograr garantizar la calidad y competitividad de la plataforma en este mercado, el cual es dinámico y competitivo.

Recursos claves:

Los recursos clave van a incluir la infraestructura tecnológica y los servidores necesarios para mantener la plataforma en funcionamiento de manera eficiente y segura. A su vez las pasarelas de pago van a asegurar transacciones confiables y seguras entre los consumidores y los restaurantes. Mientras que el personal de desarrollo y soporte técnico van a tener un cargo fundamental para garantizar la continuidad operativa y resolver cualquier problema técnico o de consumidor que pueda surgir, manteniendo así la satisfacción y fidelidad de los consumidores.

Propuesta de valor:

La propuesta de valor se va a centrar en ofrecer una experiencia integral y conveniente tanto para los consumidores como para los restaurantes. Por un lado, la plataforma SaaS va a brindar una interfaz fácil de usar que mejora la experiencia de compra y venta de comida en línea. Por otro lado, el sistema de *marketing*

multinivel con recompensas no solo va a fomentar la participación y la lealtad de los consumidores, sino que también va a reducir los costos publicitarios para los restaurantes, aumentando así su rentabilidad y presencia en el mercado.

Relaciones con clientes:

La atención al cliente es fundamental para mantener relaciones sólidas y duraderas tanto con los restaurantes como con los consumidores. El soporte técnico dedicado va a garantizar una experiencia sin problemas en la plataforma, mientras que la autogestión proporciona a los usuarios el control total sobre sus pedidos y transacciones. Además, los beneficios adicionales, como la acumulación de puntos, añaden valor y promueven la fidelidad de los clientes, lo que contribuye a la sostenibilidad y el crecimiento continuo del negocio.

Canales:

La distribución eficaz de la plataforma se va a realizar a través de múltiples canales, incluyendo la aplicación web para pedidos de comida y la aplicación web para la gestión de restaurantes. A su vez, existe un mínimo costo en *marketing* digital dirigido a los posibles consumidores y propietarios de restaurantes que ayuda a atraer y retener a los usuarios en la plataforma. Creemos que gastos en otros tipos de *marketing* es mínimo ya que el uso de *marketing* multinivel va a jugar un papel crucial en la expansión y promoción de la plataforma, fomentando a los usuarios a participar de forma activa y referir el uso de nuestro sistema.

Segmentos de clientes:

El sistema se va a dirigir a dos segmentos principales: consumidores que buscan una forma conveniente de pedir comida en línea, y restaurantes de sushi que buscan ampliar su negocio, mejorar sus ganancias y ofrecer sus servicios. Creemos que cumpliendo con las necesidades y expectativas de manera efectiva de estos, la plataforma va a mantener su relevancia y competitividad en el mercado.

La identificación de estos dos grupos fue clave al momento de definir los roles de los usuarios que participan de la plataforma que vemos en la Tabla 3.1. Entender cómo estaban compuestos los segmentos, los intereses y qué papel

cumplen en este tipo de negocio fue muy importante para definir estos roles y que funcionalidad debería ser ofrecida para cada uno. Separando en estos roles y sumando a nuestro cliente como *manager* general del sistema, fue que planteamos la división en 3 módulos (aplicaciones web).

Estructura de costes:

La estructura de costes estaría compuesta principalmente por los costos de desarrollo y mantenimiento de la plataforma, que van a incluir la inversión en infraestructura tecnológica, el personal de desarrollo y soporte técnico, así como los gastos generales asociados con la operación diaria del negocio. El soporte técnico y atención al cliente son fundamentales para poder garantizar una buena experiencia a todos los que usan la plataforma.

Fuentes de ingresos:

Las principales fuentes de ingresos provendrían de las comisiones por venta generadas a través de la plataforma y por las tarifas de ingreso pagadas por los restaurantes que desean ofrecer sus servicios en línea. Ambos son muy importantes ya que son el único ingreso del negocio ideado hasta el momento.

Para la gestión de los cobros de las comisiones por venta, tanto desde la perspectiva del restaurante como desde nuestro cliente, los involucrados deberían poder acceder a información relacionada con las finanzas. Esta información es de alta sensibilidad y clave para la transparencia de las transacciones y negociaciones. En este sentido, entendimos que era importante la separación del monitoreo de las estadísticas financieras. La visualización de estos datos es lo que permite a nuestro cliente saber cuánto cobrarle a los restaurantes, y a los restaurantes cuánto deben pagar por su participación en la plataforma. Pensamos en una sección de finanzas para dueños de restaurantes, y una sección de finanzas globales que involucren a todos los restaurantes para la aplicación web de nuestro cliente, contribuyendo así a la idea de dividir en las diferentes aplicaciones web según funcionalidad y roles.

A continuación presentamos el link que dirige al video subido en la plataforma YouTube que sintetiza y ofrece una explicación gráfica y hablada sobre la solución

del proyecto incluyendo el detalle sobre funcionamiento y modelo de negocio del mismo: https://youtu.be/HsljXglLqa4.

3.1. Requerimientos

En relación a las funcionalidades principales, como guía inicial definimos que íbamos a realizar 3 aplicaciones web diferenciadas según el perfil de usuario. Como mostramos en la Tabla 3.1, los perfiles de usuario son los consumidores de sushi, los dueños de los restaurantes y administradores de TuBocado. A continuación se detallan en alto nivel las funcionalidades principales que la solución ofrece.

Requerimientos funcionales

En esta sección se mencionan los principales requerimientos funcionales para los tres perfiles de usuario, para poder ver la lista completa de requerimientos funcionales ir al anexo A.16. Product Backlog completo.

Como se explicará en *5.2. Discovery*, estos requerimientos funcionales son el resultado de analizar el prototipo obtenido luego de la aplicación de *Design Thinking*.

Aplicación web para consumidores

La aplicación web orientada a los consumidores de comida representa el punto de acceso principal para los usuarios finales del sistema, son el mayor porcentaje de usuarios de la plataforma. Los consumidores pueden explorar los menús de los restaurantes, realizar pedidos y pagar de manera segura. Además, ofrece el beneficio de acumular Bocados con cada compra. Asimismo, la plataforma únicamente les permite registrarse si fueron referidos con un *link* brindado por un consumidor ya registrado, y permite referir a otros mediante dicho *link*. Cuando estos consumidores referidos realizan compras, tanto el referente como otros niveles de consumidores hacia arriba acumulan porcentajes de Bocados. Estas dos funcionalidades constituyen el sistema de fidelización, diseñado para fomentar la lealtad y la participación continua de los clientes.

En la sección del anexo A.13. Prototipo, se muestra el prototipo completo, a continuación vemos algunos ejemplos.

En las imágenes 3.1.1 y 3.1.2 se puede observar la funcionalidad de exploración y filtrado de restaurantes y platos de comida en la aplicación. En estas secciones los consumidores pueden filtrar por hora de entrega, reseñas y ubicación. Además, ingresando a un restaurante, los consumidores pueden explorar todos los platos de comida, buscando por categoría y agregandolos al carrito.

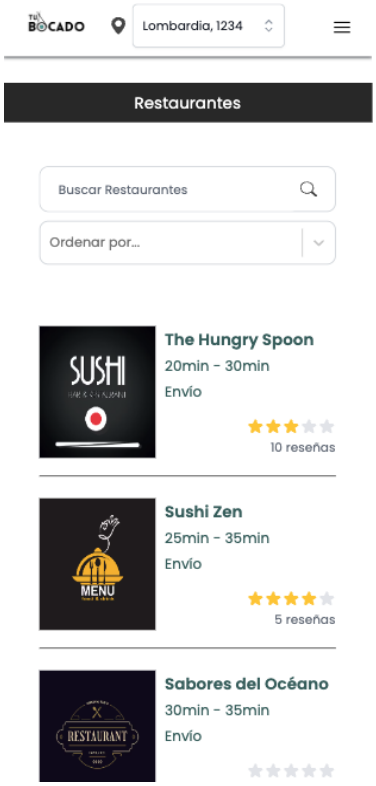


Ilustración 3.1.1: Listado de restaurantes

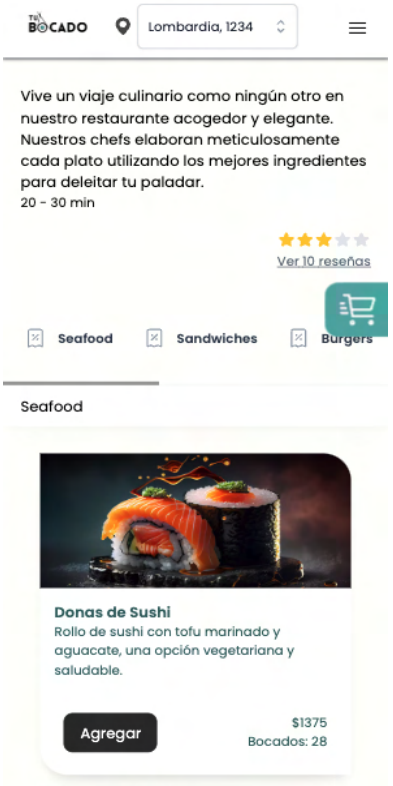


Ilustración 3.1.2: Categorías y platos

En las imágenes que 3.1.3, 3.1.4 y 3.1.5 podemos ver el proceso completo de compra en la aplicación, así como la integración con plataformas de pago y la funcionalidad de generación de Bocados. Los consumidores pueden seleccionar platos de comida, modificar su carrito de pedidos, seleccionar el método de entrega (domicilio o retiro en local), y seleccionar el método de pago (efectivo, Bocados o tarjeta) para realizar la compra. Una vez realizada la compra, los consumidores tienen la opción de observar en tiempo real el estado actual de su pedido (“Pendiente”, “En preparación”, “En tránsito” y “Entregada”), accediendo a los detalles del mismo.

Si el consumidor no utiliza Bocados como método de pago, toma lugar la funcionalidad de generación de Bocados. Esto sucede al comprar con efectivo o

tarjeta. Los consumidores reciben un bocado por cada cierta cantidad de dinero gastado, determinada por los managers de TuBocado.

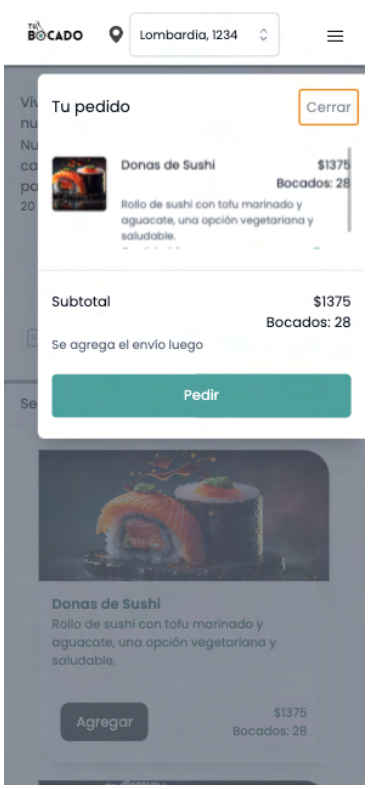


Ilustración 3.1.3: Carrito de compra

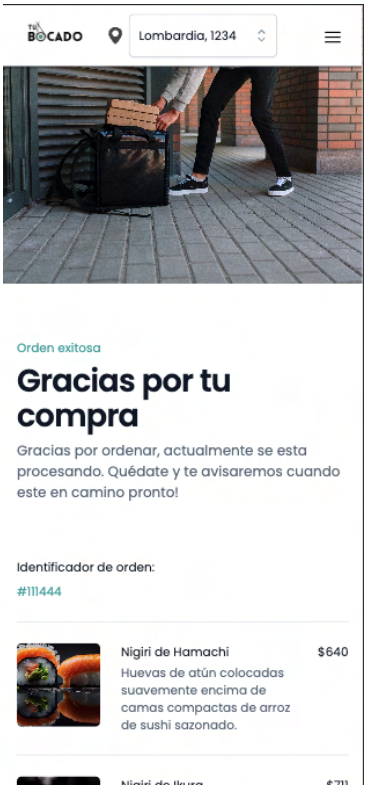


Ilustración 3.1.4: Confirmación compra

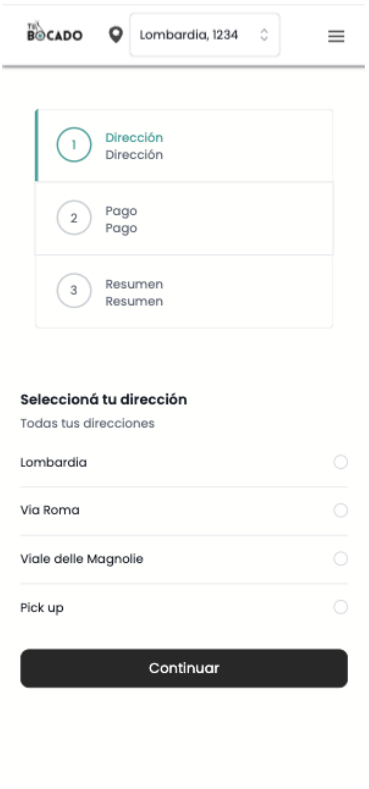


Ilustración 3.1.5: Proceso de compra

En las imágenes 3.1.6 y 3.1.7, mostramos una de las funcionalidades más importantes del proyecto, la visualización de los datos relacionados al *marketing* multinivel. Los consumidores tienen la opción de acceder a su propio enlace para invitar otros consumidores, pueden ver su red de referidos gráficamente, también tienen acceso a la cantidad de personas que refirieron así como los niveles de referidos que tienen. Por último pueden visualizar la cantidad de Bocados actuales y su historial de Bocados (cómo generaron los Bocados, si los generaron ellos o algún referido y como fueron gastando los mismos).

La inclusión de esta funcionalidad en la aplicación remarca la importancia del *marketing* multinivel como una estrategia efectiva para la adquisición de usuarios y la expansión del alcance de la plataforma de pedidos de comida.



Ilustración 3.1.6: Bocados y referidos

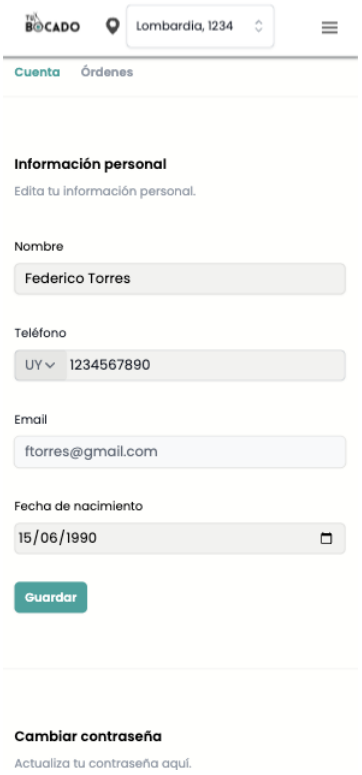


Ilustración 3.1.7: Datos personales

Aplicación web para dueños de restaurantes

La aplicación web para dueños de restaurantes ofrece herramientas para gestionar todo lo relacionado a sus restaurantes. Los datos generales del restaurante tales como horario apertura y cierre, logo, nombre y dirección, pueden ser editados. Además, la aplicación permite desde crear y personalizar platos con sus categorías, hasta gestionar los pedidos que entran. También se muestran reportes y datos sobre la contabilidad tanto en dinero como en Bocados.

Los dueños de restaurantes cuentan con un *dashboard* principal donde pueden actualizar fácilmente información como el horario de atención, dirección, rango de entrega y visibilidad para los consumidores. Una de las funcionalidades más importantes es la de visibilidad del restaurante, el dueño del restaurante además de configurar los horarios puede activar y desactivar la visibilidad del restaurante en cualquier momento. Cuando un restaurante se registra por primera vez y es aceptado por el *manager* de TuBocado, el mismo permanece invisible hasta que se termine de agregar los datos requeridos para ser visible.

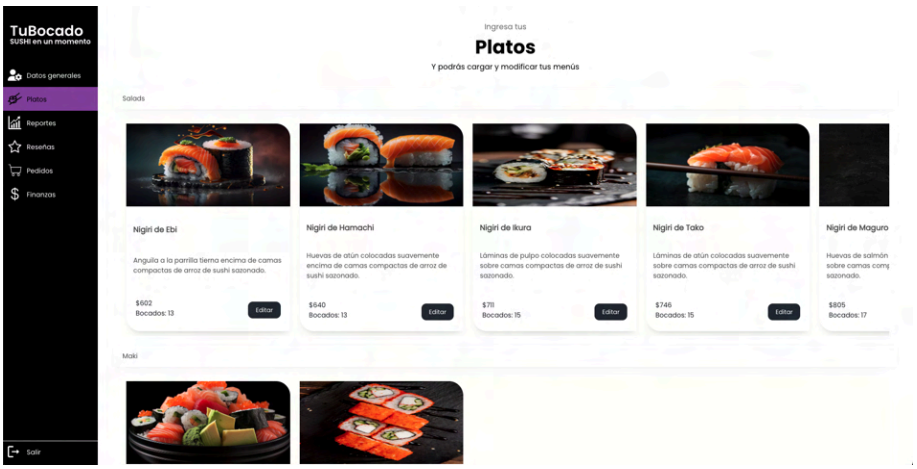


Ilustración 3.1.8: Gestión de platos

En la ilustración 3.1.8 se puede observar la funcionalidad de gestión de platos de comida diseñada para los dueños de restaurantes. En esta sección los dueños de los restaurantes pueden agregar platos, eliminarlos y editarlos, en el anexo se puede ver más en detalle esta funcionalidad.

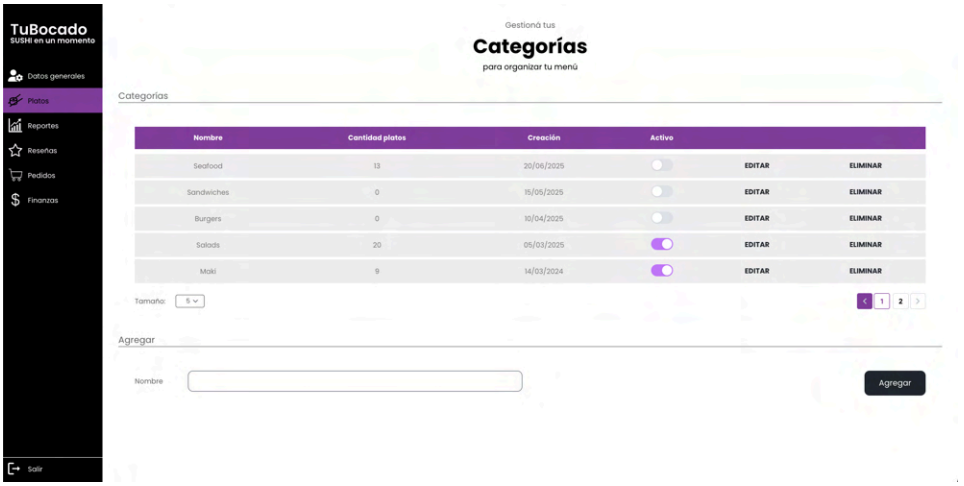


Ilustración 3.1.9: Gestión categorías

Además, se encuentra la funcionalidad de gestión de categorías de comida diseñada para administrar las categorías de cada plato de comidas que podemos ver en la ilustración 3.1.9.

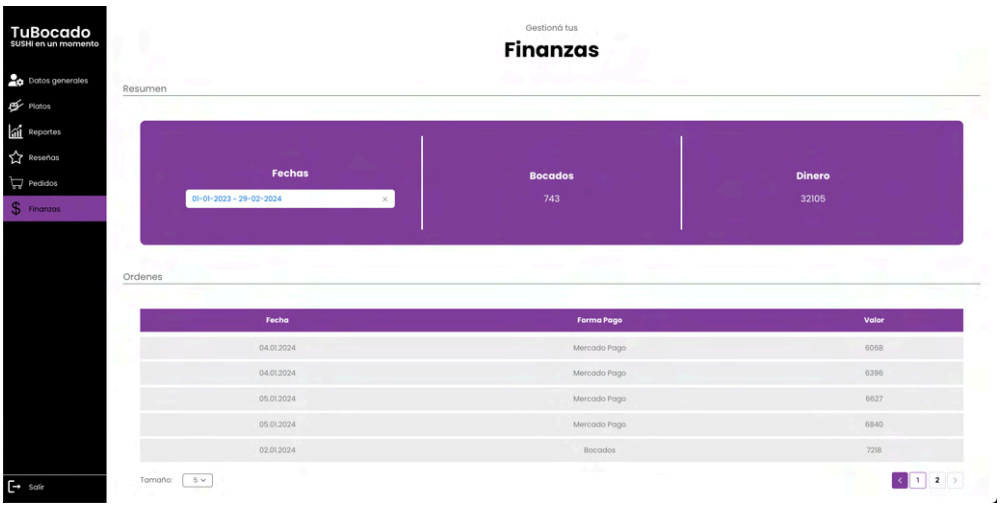


Ilustración 3.1.10: Finanzas restaurante

En la ilustración 3.1.10 vemos los informes financieros que muestran la suma de Bocados y monto total de las ventas en un rango de fechas .



Ilustración 3.1.11: Gestión de órdenes

En la ilustración 3.1.11 vemos como los propietarios pueden visualizar los pedidos entrantes en tiempo real y tienen la capacidad de cambiar su estado. Los seis estados a manejar , son: “Pendiente”, “En preparación”, “En tránsito”, “Entregado”, “Cancelado” y “Rechazado”. Los dueños son responsables de garantizar que el estado del pedido se actualice correctamente.

Existe también una vista para ver los reportes, donde los dueños de restaurantes pueden acceder a informes estadísticos según un rango de fechas.

Estos incluyen promedio de ventas, porcentaje de medios de pago, cantidad de platos vendidos y órdenes por día.

También existe la funcionalidad de visualización de reseñas. Los dueños de restaurantes pueden leer reseñas sobre su restaurante y el promedio de puntuación que tienen, lo que les permite a los consumidores tomar decisiones informadas sobre dónde comprar, y a los restaurantes les brinda *feedback*.

Aplicación web para administradores

Esta tercera aplicación web, para los *managers* de TuBocado, representa el núcleo de control de los restaurantes y consumidores asociados a TuBocado, ofreciendo las herramientas necesarias para supervisar y gestionar de manera integral todo lo que ocurre en el sistema.

La aplicación web para *managers* es crucial en el contexto del *marketing* multinivel implementado en nuestro proyecto, ya que constituye un punto central para el manejo de los parámetros clave del sistema. Entre estos parámetros se encuentran las cotizaciones monetarias asociadas a los Bocados, tanto para su canje como para su generación, así como los niveles de referidos que generan Bocados con cada compra, junto con sus respectivos porcentajes. El efecto de estos parámetros se puede observar en el siguiente ejemplo: si A refiere a B, B a C, C a D y D a F, en caso que se manejen tres niveles, A recibe Bocados de las compras de B, C y D, pero no de F porque este está a cuatro niveles de distancia. Asimismo, en este caso, si los porcentajes asociados a los tres niveles son 30%, 20% y 10% respectivamente a los niveles, A recibe por las compras de B, un 30%, por las compras de C un 20% y por las compras de D un 10% en Bocados.

Nos parece importante resaltar que la capacidad de modificar estos parámetros es fundamental para adaptar el sistema a las cambiantes necesidades de los restaurantes y el cliente. Además, estos ajustes son esenciales para optimizar la estrategia de *marketing* multinivel, ya que influyen directamente en la motivación y recompensa de los consumidores que forman parte del programa de referidos.

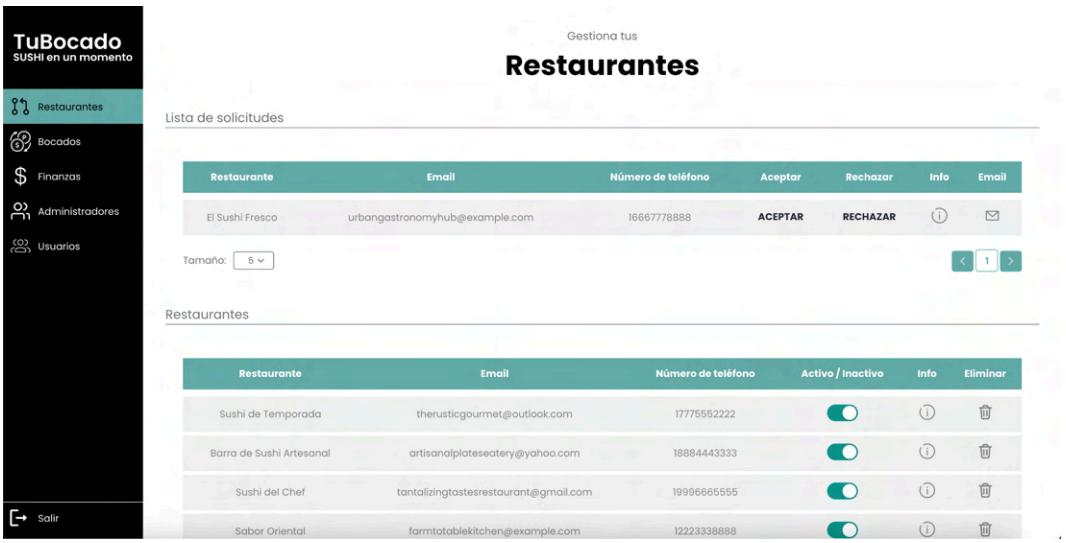


Ilustración 3.1.12: Administración restaurantes registrados

En la ilustración 3.1.12. se muestra la vista, donde los *managers* pueden ver los restaurantes registrados, acceder a sus detalles y desactivarlos o eliminarlos. Pueden revisar nuevos registros y activarlos. Además, tienen la opción de enviar consultas o sugerencias por correo electrónico a los dueños de los restaurantes.



Ilustración 3.1.13: Gestión parámetros

La ilustración 3.1.13 representa una de las vistas más importantes del sistema. En esta es donde se pueden visualizar y ajustar los parámetros críticos que manejan el funcionamiento de los Bocados. Por un lado, los *managers* pueden modificar la cotización para el canje y acumulación de Bocados, determinando así las recompensas para los consumidores y su red. Por otro lado, pueden establecer los parámetros relacionados con la cantidad de niveles de generación de Bocados y

los porcentajes de recompensa en cada nivel, lo que define la estructura y los incentivos del sistema de referidos.

Algo importante a destacar es que en este paso siempre se guarda la trazabilidad de los cambios del *manager* para luego poder gestionar correctamente las finanzas en cada línea del tiempo.

En la ilustración 3.1.14, los *managers* pueden acceder a reportes de cada restaurante, y visualizar estadísticas relacionadas con las ventas del establecimiento según un rango de fechas. Esta vista sirve para que posteriormente los dueños de TuBocado sepan cuánto se le debe facturar al restaurante.

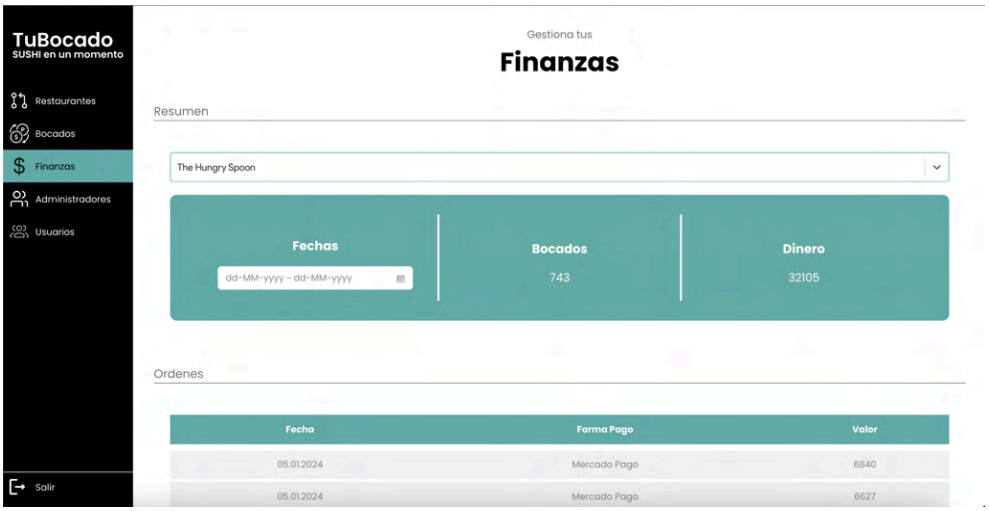


Ilustración 3.1.14: Finanzas de cada restaurante

En la ilustración 3.1.15 se muestra la vista de gestión de usuarios, además existe una vista de gestión de administradores del sistema. En estas vistas los *managers* pueden agregar más *managers* así como editar consumidores del sistema.



Ilustración 3.1.15: Gestión usuarios

Requerimientos no funcionales

Definimos los requerimientos no funcionales del sistema a partir de solicitudes y conversaciones con el cliente, investigación sobre el contexto y algunas soluciones existentes. En este proceso, pudimos validar dichos requisitos con expertos como Gastón Mosqués.

Para el diseño y estructura de la arquitectura del proyecto nos basamos en los requerimientos no funcionales agrupándolos por tipo de atributo de calidad que le corresponde, pudiendo así priorizar y diseñar en base a estos. En el capítulo de arquitectura se explica más en detalle cada uno de ellos agrupándolos por atributos de calidad y las tácticas que aplicamos para cada uno basándonos en el libro *Software Architecture in Practice*[57], además se indica qué impacto tuvieron en las decisiones de arquitectura.

Al igual que en sección de requerimientos funcionales, en esta sección se hace referencia a los mismos perfiles de usuarios ya mencionados. A continuación se mencionan los principales requerimientos no funcionales para los tres perfiles de usuario. Para poder ver la lista completa de requerimientos no funcionales ir a las sección A.16. Product Backlog - requerimientos no funcionales del anexo.

Las restricciones principales vinculadas a la naturaleza del proyecto (POC), se centran en los aspectos directamente visibles por el usuario, así como a aquellos que el cliente señaló como importantes. Entre estas restricciones se incluyen:

- **Servicio de Puntos Desacoplado:** El sistema de Bocados debe ser un servicio aparte del resto, de modo que pueda ser fácilmente integrado en otros sistemas o aplicaciones sin requerir modificaciones extensivas. Esta es una de las restricciones principales de nuestro proyecto ya que fue la que nos determinó gran parte de nuestra arquitectura. Además, el cliente especificó que era muy importante para él ya que le otorga flexibilidad para la integración con otros sistemas.
- **Interacción en Tiempo Real:** Se debe poder ver el estado de una orden en tiempo real. Tanto del lado del dueño de restaurante como del lado del consumidor. Para mejorar la experiencia de usuario y la eficiencia operativa, el sistema garantizará actualizaciones en tiempo real para el estado de las órdenes y la gestión de puntos, asegurando un procesamiento rápido de las mismas. Esto aplica tanto para la aplicación web de restaurantes como para la de los consumidores.
- **Experiencia de usuario atractiva y funcional:** El sistema debe brindar una interfaz atractiva y eficiente que cumpla con los prototipos iniciales. Además, el sistema debe permitir el fácil aprendizaje a los consumidores, dueños de restaurantes y *managers* de TuBocado brindando una interfaz gráfica que cumpla con las Heurísticas de Nielsen[58]. Creemos que este requerimiento es muy importante ya que al ser una POC, se busca priorizar los atributos de calidad que ve el usuario como la usabilidad del sistema.
- **Métodos de Pago Seguros:** El sistema debe contar con medidas de seguridad a modo de proteger las transacciones de pago de los consumidores utilizando una api de terceros confiable.

Estos requerimientos resaltan el compromiso del proyecto con la seguridad, la usabilidad, la extensibilidad y la eficiencia operativa, asegurando que el sistema no solo cumpla con las expectativas del cliente sino que también ofrezca una base sólida para el crecimiento y la adaptabilidad futura.

4. Arquitectura y diseño

El propósito de este capítulo es ofrecer una visión detallada de la arquitectura diseñada para la solución, así como justificaciones de las decisiones claves tomadas durante el proceso. Además, se proporciona información detallada sobre tecnologías usadas y decisiones tomadas en cuanto al desarrollo e infraestructura.

4.1. Descripción de la arquitectura

Para describir la arquitectura nos basamos en el modelo *Views and Beyond* [56], *del Software Engineering Institute de Carnegie Mellon University*, el cual segmenta las vistas en tres categorías: Vistas de módulos, Vistas de componentes y conectores, y Vistas de asignación. En primer lugar, la vista de módulos detalla la estructuración del sistema en un conjunto de unidades de código. En segundo lugar, la vista de componentes y conectores se centra en los componentes individuales del sistema y en cómo se conectan entre sí a través de conectores. Finalmente, la vista de asignación muestra cómo los módulos, componentes o servicios específicos se asignan y distribuyen en el entorno de ejecución, como *hardware*, sistemas de archivos y la disposición de los equipos de desarrollo de *software*.

Esta estructura nos permitió abordar de manera integral la arquitectura del sistema, proporcionando un análisis detallado y nos facilitó exponer la justificación de nuestras decisiones arquitectónicas en cada una de estas perspectivas.

Vista de componentes y conectores

En esta sección, se mostrará la vista de componentes y conectores específica del *backend*. Decidimos comenzar por esta vista ya que es fundamental para comprender cómo los elementos de software interactúan en el sistema.

Iniciaremos con la representación primaria, mostrando el diagrama de la vista de componentes y conectores para destacar cómo interactúan entre sí. Posteriormente, examinaremos el Catálogo de Elementos, donde detallamos las responsabilidades y contribuciones de cada componente al sistema en su conjunto. Luego, mencionaremos las decisiones de diseño, y la explicación y justificación de

las selecciones arquitectónicas realizadas. Este enfoque aclara el rol de cada servicio y conector y muestra la lógica detrás de la estructura del *backend*.

Por último, dentro de la subsección de comportamiento, presentamos un diagrama de secuencia que ilustra el flujo de una solicitud típica, permitiendo visualizar del proceso que involucra una aplicación web como el monolito principal *TuBocado.TuBocadoBE* y la interacción con el servicio TuBocado.PointsBE.

Representación primaria

A continuación se presenta el diagrama de la vista de componentes y conectores en el cual nos enfocaremos en cómo las aplicaciones web de TuBocado, específicamente, las de los restaurantes, *managers* y consumidores, interactúan con el sistema a través de *TuBocado.TuBocadoBE*, el monolito central. Este monolito sirve como el núcleo del servicio, ofreciendo interfaces a través de las cuales las aplicaciones pueden enviar solicitudes. Para procesar estas solicitudes, *TuBocado.TuBocadoBE* las recibe y de ser necesario las coordina con el otro servicio que se encarga de gestionar los parámetros del sistema, sistema de referidos y puntos, asegurando una respuesta efectiva.

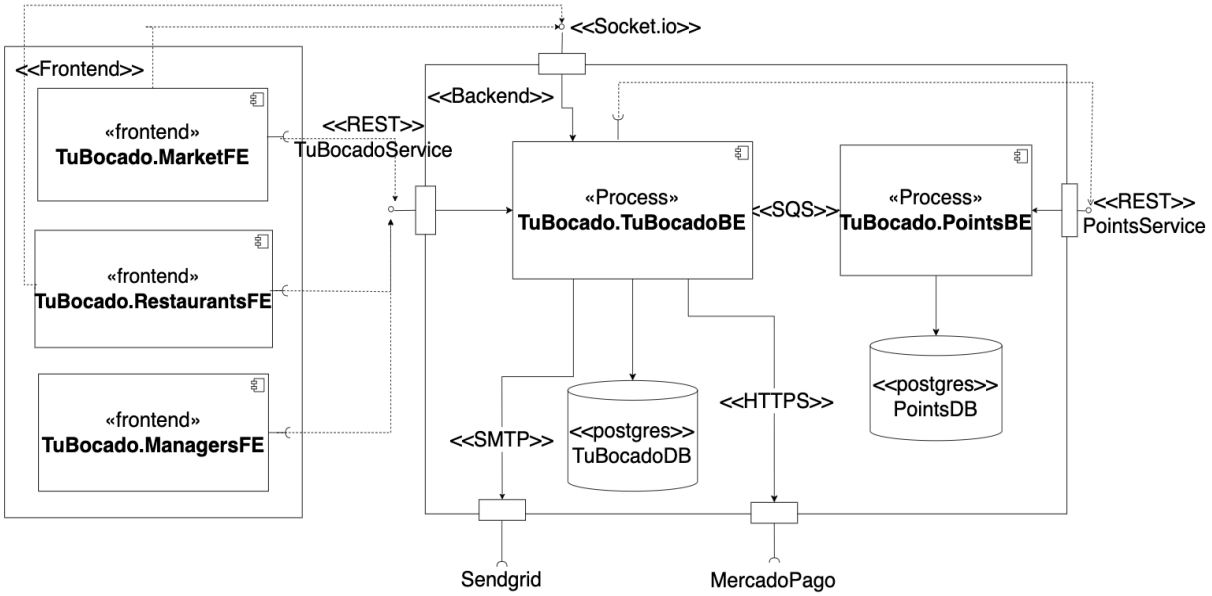


Ilustración 4.1.1 : Diagrama de componentes y conectores

Catálogo de elementos

Componente/Conector	Tipo	Descripción
TuBocado.TuBocadoBE	API REST	Es una API REST que se encarga de recibir todas las peticiones de los usuarios, consumidores, restaurantes y administradores de la aplicación. Se encarga de todas las funcionalidades del sistema y de consultar y comunicarse con el servicio de puntos en caso de ser necesario. Esto se encuentra desarrollado en NodeJS.
TuBocado.PointsBE	API REST	Es una API REST encargada de gestionar todo lo relacionado con la generación, canje y administración de puntos de todos los usuarios. Además se encarga de gestionar todos los parámetros de cotización y niveles de referidos del sistema. Esto se encuentra desarrollado en *NodeJS*.
TuBocado.MarketFE	WebAPP	Es la Aplicación Web principal destinada a los usuarios para que puedan encargar sushi por encargo, invitar amigos a la aplicación y hacer todas las funcionalidades de los usuarios. Esta se encuentra desarrollada en *ReactJS* y se comunica con *TuBocado.TuBocadoBE* mediante el protocolo HTTPS.
TuBocado.RestaurantsFE	WebAPP	Es la Aplicación Web destinada a los dueños de restaurantes para que puedan cargar sus platos, ingresar los datos de su restaurante y hacer todas las funcionalidades de los dueños de restaurantes. Esta se encuentra desarrollada en *ReactJS* y se comunica con *TuBocado.TuBocadoBE* mediante el protocolo HTTPS.
TuBocado.ManagerFE	WebAPP	Es la Aplicación Web destinada a los administradores de la aplicación para que puedan gestionar los parámetros claves de la misma como los valores de cotización de puntos, cantidad de niveles de referidos y hacer todas las funcionalidades de los administradores. Esta se encuentra desarrollada en *ReactJS* y se comunica con *TuBocado.TuBocadoBE* mediante el protocolo HTTPS.
PointsDB	Postgres	Motor de base de datos Postgres relacional, encargado del almacenamiento integral de los datos relacionados con los puntos, parámetros y referidos del usuario, tanto los generados como los gastados.
TuBocadoDB	Postgres	Motor de base de datos Postgres relacional, encargado del almacenamiento integral de los datos del sistema.
PointsQueue	Message Queue (SQS)	Cola de mensajes utilizada para actualizar la base de datos de *TuBocado.PointsBE* y proporcionar un medio de comunicación asincrónica sin pérdida de información.
MercadoPago API	Externo	Servicio externo con el cual se integra el sistema para gestionar los pagos.
Sendgrid	Externo	Servicio externo con el cual se integra el sistema para gestionar el envío de emails.

Tabla 4.1.1: Catálogo de elementos de la vista de componentes y conectores

Decisiones de diseño

Como se puede observar en el diagrama (Ilustración 4.1.1), para la arquitectura del backend optamos por una combinación de arquitectura monolítica modular y un servicio interconectado.

En primer lugar definimos la implementación de una arquitectura monolítica modular llamada *TuBocado.TuBocadoBE*, la cual está compuesta por módulos basados en el *Bounded Context* y es donde pasan las principales consultas. La justificación de porqué optamos por un monolito modular y no microservicios se detalla en este capítulo en la sección de porqué monolito modular.

Luego está el servicio de puntos, el cual se comporta como un servicio dedicado a la gestión de los puntos, parámetros y referidos de cada usuario. Este servicio almacena únicamente los datos relacionados a la gestión del *marketing* multinivel del usuario, y no la información de los usuarios en sí. Para poder identificar a cada usuario, únicamente guardamos el *id* de cada uno y en base a eso realizamos la lógica correspondiente. Esto incluye desde almacenar los puntos de cada usuario, generar los puntos necesarios en base a un valor de dinero y los referidos del mismo, hacer devoluciones de puntos y gestionar el consumo de los mismos. También se encarga de gestionar y almacenar toda la información necesaria para implementar el marketing multinivel, como los parámetros del sistema que determinan cuántos niveles hacia arriba de referidos generan puntos, los referidos de cada usuario, y el valor del canje de puntos, tanto de moneda a puntos como de puntos a monedas. Además es el encargado de ejecutar el algoritmo para calcular el porcentaje de ganancia de cada usuario en base al monto generado y el usuario que lo generó. Es importante destacar que estaremos utilizando el término “servicio de puntos” en lugar de “servicio de Bocados”. La razón es que al ser un servicio desacoplado, por ahora solo es utilizado por el monolito modular. Sin embargo, en el futuro, otros servicios podrían conectarse al servicio de puntos y referirse a estos con diferentes nombres.

Por otro lado se encuentran las aplicaciones web, *TuBoacado.MarketFE*, *TuBocado.RestaurantsFE* y *TuBocado.ManagersFE* se comunican únicamente con el monolito modular mediante protocolos HTTPS ya que expone APIs con rutas

diferenciadas para cada tipo de usuario: dueños de restaurantes, administradores y consumidores. Para los dueños de restaurantes y *managers* de TuBocado, disponemos de rutas específicas que incluyen *endpoints* destinados a gestionar operaciones de administración y manejo del restaurante. Estos *endpoints* emplean métodos de autenticación avanzados que aseguran que solo los usuarios autorizados tengan acceso a sus funcionalidades. Por otro lado, las rutas dirigidas a los consumidores de la aplicación se enfocan en brindar una experiencia de usuario fluida y segura para las operaciones de los mismos, como realizar pedidos, ver la cantidad de referidos y Bocados y revisar menús. La clave de este diseño radica en que todos los tipos de usuario operan con métodos de autenticación distintos, lo cual garantiza que no haya posibilidad de acceso cruzado entre los diferentes tipos de usuarios, esto se habla más en detalle en la sección de atributos de calidad.

En cuanto a la comunicación entre *TuBocado.TuBocadoBE* y *TuBocado.PointsBE*, el monolito se comunica con el servicio de puntos mediante el patrón *Remote Procedure Invocation* (API REST) y a su vez por el patrón *Messaging* (colas de mensajes). Con lo que respecta a la autenticación, para poder autenticarse entre la APIs definimos un *secret token* que valida *TuBocado.PointsBE* cada vez que le llega una request de *TuBocado.TuBocadoBE.* Mientras que para la cola de mensajes utilizamos los mecanismos de seguridad que brinda SQS. En el anexo A.5. Descripción de las estructuras de las APIs se detalla en profundidad.

Por otro lado, para mejorar la eficiencia y aumentar la claridad y mantenibilidad del sistema, implementamos una estrategia basada en el patrón *Command Query Separation (CQS)*. Este enfoque consiste en separar claramente las operaciones que modifican el estado (*command*) de las que retornan información (*queries*) con lo que respecta a la gestión de puntos, aunque no implica necesariamente utilizar bases de datos separadas para escritura y lectura, en nuestro caso no hicimos dicha división. *TuBocado.PointsBE* se encarga de manejar todos los aspectos relacionados con los puntos y su generación, incluyendo la actualización de los mismos a partir de la información enviada en la cola de mensajes por *TuBocado.TuBocadoBE*. Es decir, cada vez que el monolito modular termina de procesar un pedido que genera puntos, envía un comando a

TuBocado.PointsBE para actualizar la información correspondiente y que este se encargue de toda la lógica de la generación.

La implementación de esta estrategia permite minimizar el impacto en la performance de las aplicaciones web ya que al utilizar colas de mensajes, podemos manejar las actualizaciones de forma asincrónica, reduciendo la carga de procesamiento y mejorando el manejo de la concurrencia. Esto asegura que los procesos no interrumpan las operaciones principales y que los pedidos se manejan sin demoras. Además, permite que se dé una respuesta rápida al sistema y que no tenga que quedar esperando, pudiendo cumplir con los RFNs necesarios.

Cuando se termina de procesar un pedido en *TuBocado.TuBocadoBE*, si se efectúa mediante un método de pago distinto a Bocados, se registra la transacción en la base de datos y se envía un comando a *TuBocado.PointsBE* a través de la cola. Este proceso permite actualizar la información de puntos y calcular los referidos de un usuario de manera eficiente, optimizando las consultas para generación de puntos y reportes administrativos. Originalmente, el cliente había solicitado que la generación de puntos se realizará de manera asincrónica, mediante un proceso *batch* durante horas que la aplicación no tenga restaurantes activos y pedidos activos (ej: madrugada), permitiendo que los usuarios vieran los puntos generados al día siguiente. Sin embargo, tras implementar el *batch* y verlo funcionar, evaluamos diversas opciones junto con el PO (*product owner*) en los *Release Review* y *Release Planning* y decidimos que sería más beneficioso generar los puntos en el momento en que los pedidos alcanzan el estado de "entregado". Esto se debe a que, una vez entregado el pedido, el usuario no puede cancelarlo, lo que nos permite evitar situaciones de reversión en la generación de puntos. De esta manera, los usuarios pueden ver cuántos puntos han acumulado ese mismo día, ofreciendo una experiencia de usuario mejorada, ya que el proceso *batch* previo no ofrecía ninguna ventaja significativa.

Por lo tanto, la implementación de CQS en nuestro sistema ha sido fundamental para mejorar la claridad y la estructura del código. Logramos una estructura donde las operaciones de modificación de estado y las consultas de datos son manejadas de manera independiente, optimizando el flujo de trabajo sin necesidad de separar físicamente las bases de datos.

Comportamiento

A continuación, se muestra un diagrama de secuencia el cual muestra el flujo para obtener los datos de un consumidor actual. Para una mejor lectura del diagrama, lo partimos en dos, siendo el segundo la continuación del primero.

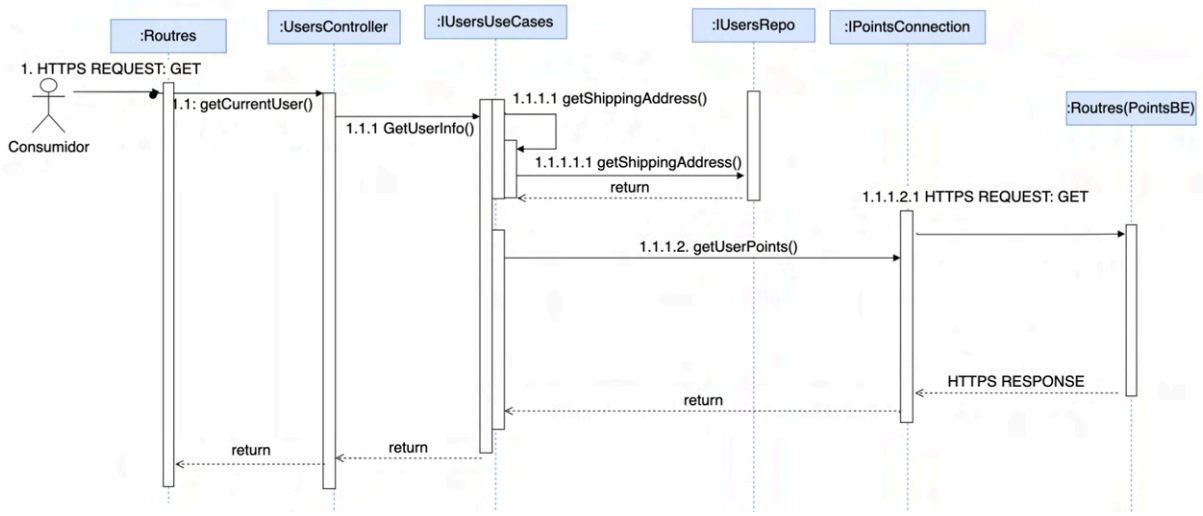


Ilustración 4.1.2: Diagrama de secuencia de datos del usuario 1

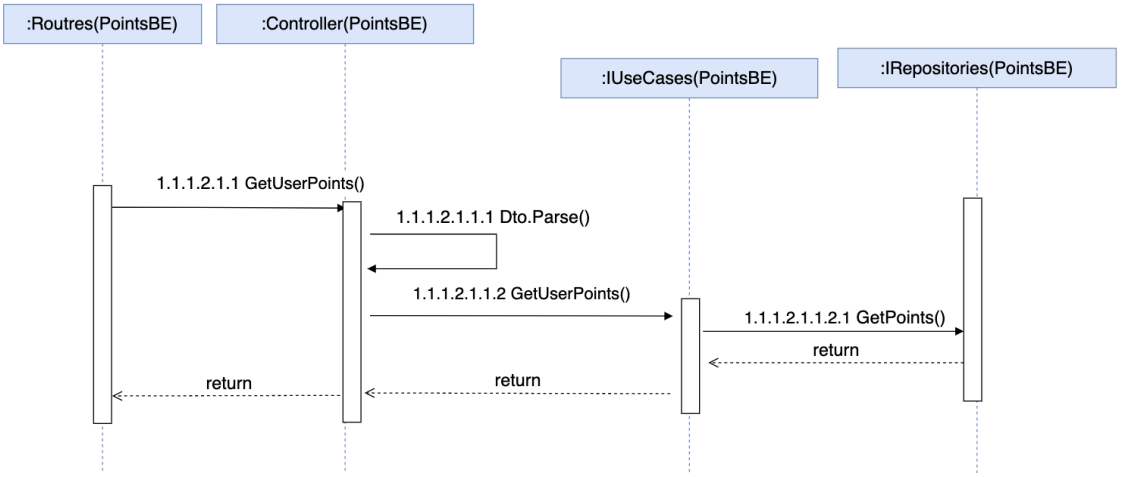


Ilustración 4.1.3: Diagrama de secuencia de datos del usuario 2

En este diagrama decidimos representar una funcionalidad integral que muestre cómo interactúan ambos servicios, el de puntos y el monolito principal. El diagrama de secuencia refleja la funcionalidad de la obtención de información del usuario actual (*getCurrentUser*), en este no solo se obtienen sus datos sino también se obtienen los puntos del mismo. Para simplificar y mostrar el flujo del sistema, evitamos mostrar la lógica del *middleware* y llamamos directamente al *controller* en

el módulo de *Users* ya que previamente al llamado del *controller*, en el *route* pasa por el *middleware* que autoriza y autentica el usuario y allí ya se guardan los datos del usuario en *req.user* y es por esto que más adelante no se muestra una función de *getGeneralUserInfo* para agarrar los datos básicos del usuario. La secuencia inicia con una solicitud del consumidor al endpoint */users/:id* donde busca la información del usuario y llama al *controller* con getCurrentUser, el cual actúa como el punto de entrada para recuperar datos detallados del usuario.

El proceso central es gestionado por la función *getUserInfo* en el *useCase* del usuario, particularmente el *useCase* de *getUserInfo*, que orquesta múltiples llamadas asíncronas para generar la información del usuario. Primero, se solicita la dirección de envío del usuario a través de *getShippingAddress*, la cual accede a la base de datos del monolito para recuperar los datos pertinentes. De forma paralela, se realizan llamadas al servicio externo de puntos para obtener los puntos actuales del usuario mediante *getUserPoints*.

Allí se puede ver cómo se comunica mediante HTTPs dado que es una consulta de puntos. Al igual que en el monolito, la consulta va desde el *controller*, hacia el *useCase* y luego al repositorio mediante interfaces y haciendo referencia a las dependencias de afuera hacia adentro. Este paso subraya la integración entre el sistema de puntos y el monolito principal, demostrando la colaboración entre diferentes partes del sistema para construir una vista completa de los datos del usuario. Una vez que todas las llamadas asíncronas han concluido se construye un objeto que luego es retornado. Este objeto se devuelve finalmente a *getCurrentUser*, que lo envía como respuesta al cliente.

El diagrama enfatiza la orquestación de llamadas asíncronas y la interacción entre componentes internos y externos, mostrando la complejidad de las operaciones necesarias para compilar una vista completa del usuario en el sistema monolito modular con funcionalidades extendidas a través de servicios externos.

Vista de módulos

Vista Layers *TuBocado.TuBocadoBE*

En esta representación, exhibimos la vista de *layers* del módulo de *orders* dentro del *backend*. Tomamos la decisión de mostrar este único módulo como representación, ya que todos los módulos adhieren a una misma estructura. De esta forma, al explicar esta vista, brindamos una comprensión general de la disposición de todos los módulos.

Representación primaria

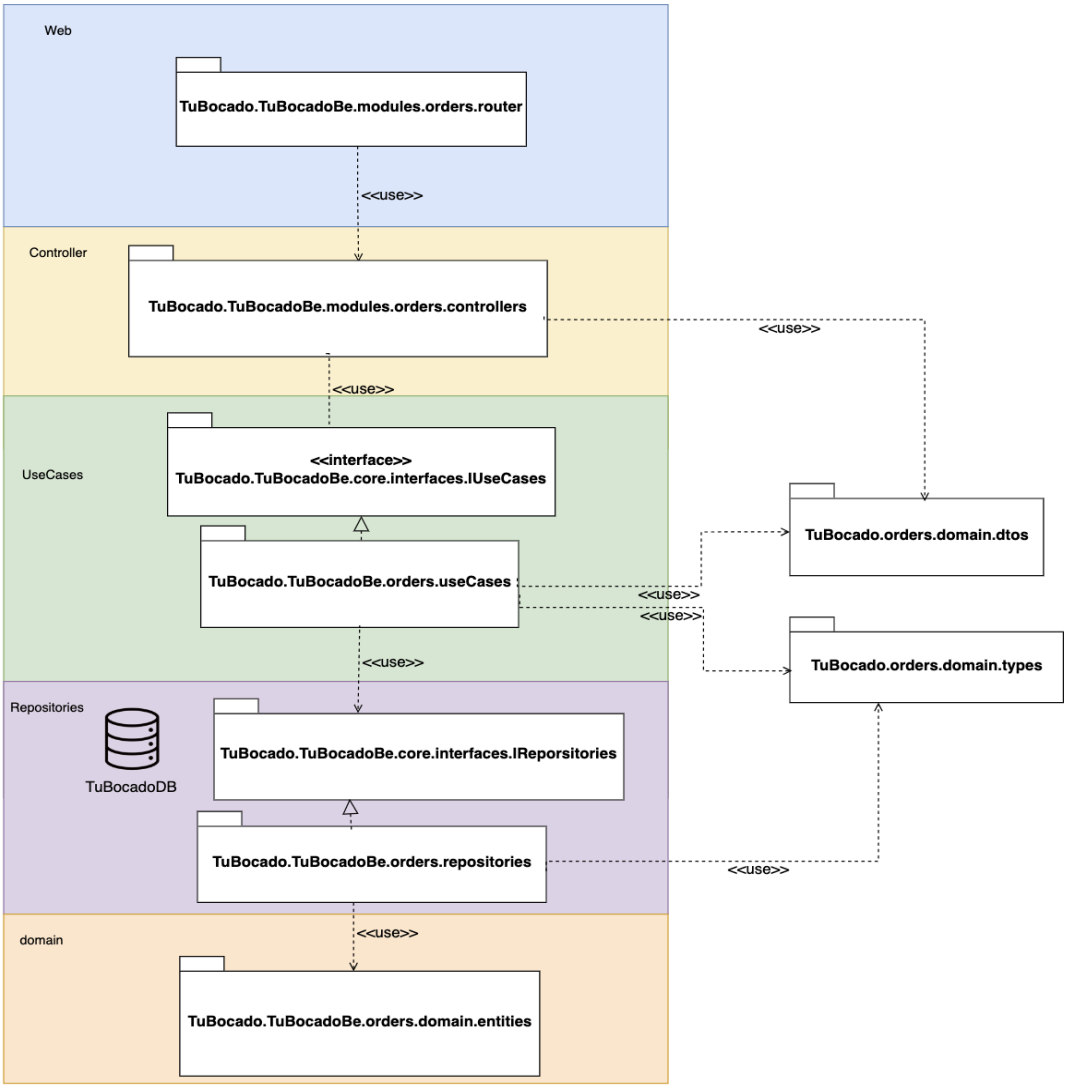


Ilustración 4.1.4: Diagrama de vista layers

Catálogo de elementos

Elemento	Descripción
TuBocado.TuBocadoBE.modules.orders.routes.	Define todas las rutas del módulo, las mismas direccionan las peticiones a los controllers correspondientes.

TuBocado.TuBocadoBE.modules.orders.controllers	Se encargan de recibir las peticiones, utilizados desde el router, para que las procese y responda adecuadamente, además de realizar el log de errores.
TuBocado.TuBocadoBE.core.interfaces.IOrdersUseCases	Contiene las interfaces definidas para todos los *useCases* del módulo de *orders*. Estos son utilizados desde los controladores.
TuBocado.TuBocadoBE.modules.orders.useCases	Implementa las interfaces definidas para los *useCases*. Aquí podemos encontrar la lógica de negocio del sistema.
TuBocado.TuBocadoBE.core.interfaces.IOrdersRepositories	Contiene las interfaces definidas para todos los *repositories* del módulo de *orders*. Estos son utilizados desde los controladores.
TuBocado.TuBocadoBE.modules.orders.repositories	Implementa las interfaces definidas para cada uno de los repositorios del módulo. Son los encargados de la comunicación con los ORM.
TuBocado.TuBocadoBE.modules.orders.domain.DTOs	En los DTO (*Data Transfer Object*) se encuentran objetos que se utilizan para transferir datos entre el *frontend* y el *backend*
TuBocado.TuBocadoBE.modules.orders.domain.entities	Estos son los tipos o interfaces que definen la estructura de los modelos de la base de datos.
TuBocado.TuBocadoBE.modules.orders.doamain.types	Aquí definimos tipos comunes que se utilizan, como tipos de respuesta de API, etc

Tabla 4.1.2: Catálogo de elementos de la vista de layers

Decisiones de diseño

En la vista de *layers* mostramos el manejo de responsabilidades de los diferentes componentes de los módulos de nuestro sistema y como están estructurados. Al observar el diagrama se puede ver que esta estructuración se basa en una adaptación del patrón *layers* con una estructura modular enfocada en distintas capas de funcionalidad independientes y cohesivas. Cada capa asume responsabilidades claras y se comunica con las otras a través de interfaces bien definidas y no de clases concretas; lo que aumenta la cohesión dentro de los módulos, reduce el acoplamiento entre ellos e incrementa la coherencia semántica. Además, cada capa tiene una responsabilidad específica, lo que es crucial para cumplir con el principio "S" de SOLID, el de Responsabilidad Única, indicando que una clase o módulo debe tener una sola razón para cambiar; SOLID es un conjunto de reglas y mejores prácticas a seguir al diseñar una estructura de clase [75]. Las capas más internas contienen la lógica de negocio y las reglas del dominio, mientras que las capas externas manejan los detalles de la interfaz de usuario y la infraestructura.

El diagrama 4.1.4 distingue cuatro capas principales. En primer lugar, se encuentra la capa *Web* compuesta por el *router* y los *controllers*. En el *router* se exponen los contratos para poder comunicarse con el servicio, y se realiza la gestión de las solicitudes derivando al *controller* correspondiente. El *controller* valida la entrada y coordina la ejecución de los casos de uso además de dar las respuestas a la interfaz de usuario.

En segundo lugar, se encuentra la capa *useCases* que contiene la implementación a las interfaces con la lógica de la aplicación y define cómo se realiza el trabajo específico de la misma. Aquí implementamos los casos de uso, los cuales representan las operaciones y flujos de trabajo que se pueden realizar. Los casos de uso se comunican con otros casos de uso y también con las interfaces de los repositorios dentro del mismo módulo. Además, de manera excepcional, dichos casos de uso pueden comunicarse con los que existen en otros módulos a través de las interfaces de estos últimos.

En tercer lugar, existe la capa de *repositories*, donde se encuentran las implementaciones de los repositorios del módulo, siendo esta la capa encargada de gestionar el acceso a la base de datos. Aísla y centraliza la conexión con la base de datos mediante el ORM, como la persistencia u obtención de los mismos. Por último, encontramos la capa de entidades, esta capa representa el núcleo de la aplicación y contiene las entidades que son persistidos en la capa de repositorios. Aquí también se encuentran los DTOs y los tipos utilizados para las respuestas.

En relación a la arquitectura, al adoptar una modificación del patrón de Layers a través de *Clean Architecture*, beneficiamos la modularidad, modificabilidad y seguridad del sistema. La modularidad y modificabilidad se ven beneficiadas ya que *Clean Architecture* promueve una clara separación entre la lógica de negocio, la interfaz de usuario, el almacenamiento de datos y otras preocupaciones externas. Este enfoque conduce a un diseño en el que los componentes del sistema son independientes, facilitando su desarrollo, prueba y mantenimiento de forma aislada. Además, las características de alta cohesión y bajo acoplamiento, inherentes a *Clean Architecture*, hacen que las modificaciones sean más sencillas y controladas. Gracias a esto, los cambios realizados en una parte del sistema tienen una menor probabilidad de impactar en otras áreas, disminuyendo así tanto el costo como el

riesgo asociados al mantenimiento e implementación de nuevas funcionalidades. [76]

Gracias a este diseño, muchos atributos se ven favorecidos, entre ellos la testabilidad. La declaración de interfaces claras y la aplicación de la inversión de dependencias simplifican notablemente la realización de pruebas unitarias e integrales. La inversión de dependencias la empleamos para garantizar que las capas internas del sistema no dependan de las externas. Esto se logra definiendo interfaces precisas y permitiendo la inyección de implementaciones concretas en lugar de su instancia directa. Desde el diseño, priorizamos la alta cohesión y un bajo acoplamiento, aplicando guías de GRASP para asignar responsabilidades de manera efectiva, lo cual mejora la modificabilidad y, posteriormente la mantenibilidad. Esta decisión también se alinea con el principio de dependencias estables, asegurando que cada componente dependa de otro más estable que él mismo, reforzando así la robustez y fiabilidad del diseño global.

En conclusión, para desarrollar la arquitectura Clean, nos apoyamos en los principios del *Domain Driven Design* (DDD), particularmente *Bounded contexts*, enfocándonos en la creación de un modelo de dominio bien definido y claro, que refleja adecuadamente el negocio de la solución. En este enfoque, el núcleo de toda la estructura de diseño es el dominio en sí: una vez que definimos y comprendimos el dominio, procedimos a reflejar esta comprensión en la arquitectura y el código del sistema. [77]

Para estructurar la arquitectura de los módulos en NodeJs nos basamos en varias prácticas que fueron fundamentales para organizar el proyecto. Particularmente en un video, este recurso resultó ser fundamental para nuestro enfoque. [78]

Vista Layers *TuBocado.PointsBE*

La representación de la vista *Layers* de *TuBocado.PointsBE* es muy similar a la que se encuentra dentro de cada módulo de *TuBocado.TuBocadoBE* ya que se basa en los mismos principios y diseños arquitectónicos, en la sección A.3. Vista Layers TuBocado.PointBE se encuentra detallada esta vista.

Vista de Usos

A continuación se muestra la vista de usos a nivel general, como el sistema de *TuBocado.TuBocadoBE* es el que usa al sistema de puntos *TuBocado.PointsBE.*

Representación primaria

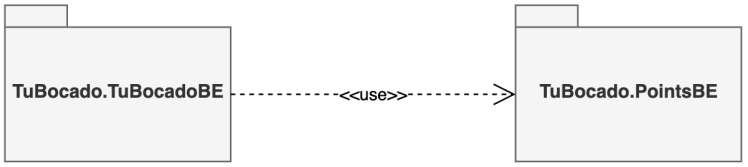


Ilustración 4.1.5: Representación primaria de vista de usos

Catálogo de elementos

- **TuBocado.TuBocadoBE**: Monolito modular y servicio principal de gestión de TuBocado, además se encarga de recibir todas las *requests* de los usuarios y derivarlas al servicio de puntos si corresponde.
- **TuBocado.PointsBE:** servicio encargado de gestionar toda la lógica correspondiente al *marketing* multinivel, es decir gestión de puntos, niveles, referidos y parámetros del sistema. Además se encarga de manejar la trazabilidad de los cambios de cotización de los Bocados y de los diferentes porcentajes correspondientes para cada uno de los niveles.

Justificación de diseño

Como se puede ver en la representación gráfica intentamos limitar el acoplamiento entre los servicios al máximo, de forma que cada servicio es capaz de gestionar completamente las entidades de sus dominios correspondientes, desde procesar *requests* hasta la persistencia en base de datos. En el caso de *TuBocado.TuBocadoBE* también se encarga de recibir todas las *requests* y manejarla o derivarla al servicio de puntos de ser necesario.

Como mencionamos en la vista de componentes y conectores esta interacción es dada por request HTTPs o mediante cola de mensajes dependiendo del caso, mediante SQS. Una dificultad que obtuvimos es que SQS no escucha eventos sino que es por *polling*, por lo que manualmente tiene que estar preguntando si hay algo en la cola. Para esto hicimos un *timeout* que cada 5

segundos que pregunte si hay algo en la cola, este tiempo está en las variables de ambiente, por lo que es variable en tiempo de ejecución, entonces en el caso que se quiera hacer cada más segundos, se puede hacer en tiempo de ejecución.

Por último, *TuBocado.TuBucadoBE* también es el encargado de la seguridad vertical, el cual para cada *request* se fija que el usuario tenga el rol adecuado y que además tenga un *token* válido, por lo que se encarga de la autorización y autenticación.

Vista Descomposición TuBocado.TuBocadoBE

La vista de descomposición de *TuBocado.TuBocadoBE*, junto con su catálogo de elementos se encuentra en la sección A.2. en el anexo.

Vista de asignación

En esta sección se presenta la representación primaria de cómo los componentes del sistema se distribuyen en el entorno de ejecución. Esta sección enfatiza la disposición física de los componentes en la infraestructura, sigue con el Catálogo de Elementos, que detalla los nodos y conectores utilizados, incluyendo servicios de AWS y protocolos de comunicación y finaliza con Decisiones de Diseño, donde se explica la elección de una arquitectura modular y el despliegue independiente de ambos servicios, fundamentando estas decisiones en objetivos específicos del proyecto.

Vista de despliegue

Representación primaria

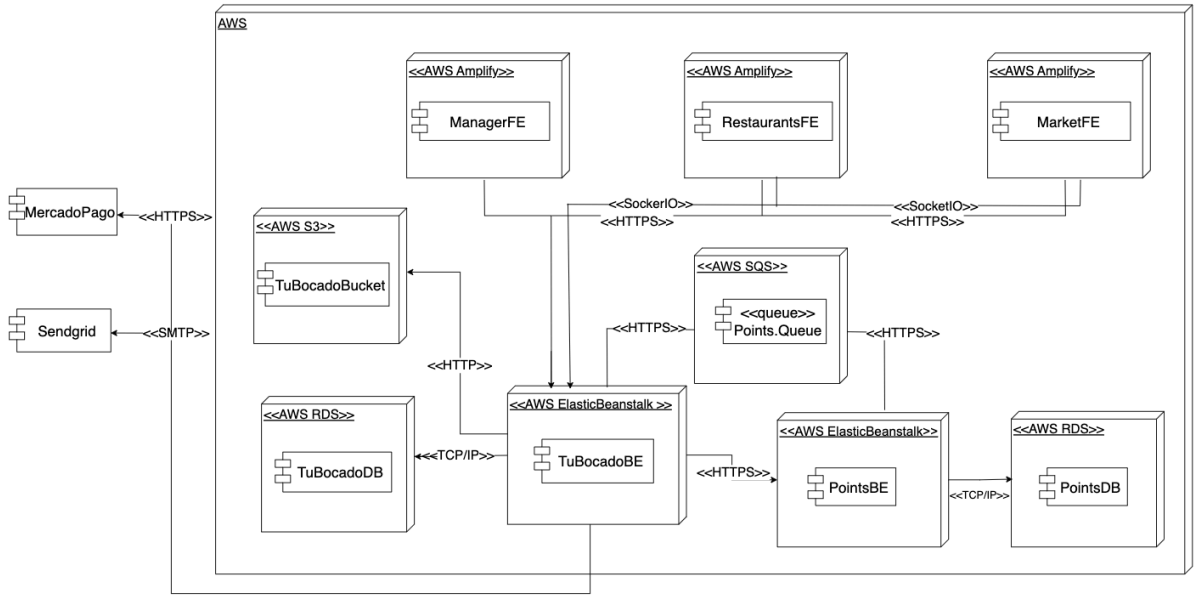


Ilustración 4.1.6: Diagrama de despliegue

Catálogo de elementos

Nodo	Descripción
AWS	*Amazon Web Services* es una plataforma que brinda un conjunto de servicios en la nube que utilizamos en nuestra infraestructura. Contiene toda la infraestructura del *backend* y *frontend* de TuBocado.
S3	Amazon S3 o *Amazon Simple Storage Service* es un servicio ofrecido por Amazon *Web Services* que proporciona almacenamiento de objetos a través de una interfaz de servicio web en la cual el *backend* se conecta para guardar imágenes.
Amplify	Amplify Studio es una interfaz visual para configurar y mantener *backends* de aplicaciones y crear interfaces de usuario frontend por fuera de la consola de AWS por lo que mantiene las aplicaciones web que se conectan con las instancias del *backend* y realizar consultas HTTPs
Elastic Beanstalk	*Elastic Beanstalk* es un servicio ofrecido por AWS diseñado para facilitar a los desarrolladores la implementación y gestión de aplicaciones en la nube de Amazon sin preocuparse por la infraestructura que corre detrás. Automatiza el proceso de despliegue, desde la provisión de instancias EC2, hasta el autoescalado y monitoreo del estado de la aplicación.
SQS	*Amazon Simple Queue Service* es un servicio de colas de mensajes distribuidos de AWS que admite el envío programático de mensajes a través de aplicaciones de servicios web como una forma de comunicarse a través de Internet.
RDS	*Amazon Relational Database Service* (Amazon RDS) es una colección de servicios administrados que facilita las tareas de configuración, operación y escalado de una

	base de datos en la nube. Sencillo de utilizar, configurar y escalar. Además es altamente seguro al encontrarse en una instancia separada a las demás. Contiene ambas bases de datos de PostgreSQL.

Tabla 4.1.3: Nodos de la vista de asignación

Conector	Características	Descripción
HTTPS connection	Altamente utilizado y conocido (estándar), sincrónico o asincrónico dependiendo de lo que se necesite, buen rendimiento en velocidad y permite encriptación.	Se utiliza https para la comunicación entre *TuBocado.TuBocadoBE* y *TuBocado.PointsBE*, además entre los tres *frontends* y *TuBocado.TuBocadoBE*. HTTPS permite una comunicación simple de implementar y monitorear, y también cumplir con buen nivel de seguridad y performance.
TCP/IP	Altamente utilizado y conocido (estándar), fiabilidad en que se recuperen datos perdidos y buena velocidad.	Se utiliza TCP/IP para la conexión a las bases de datos teniendo en cuenta que es el estándar presente en el motor utilizado.
SMTP	Muy utilizado, estándar para la comunicación de correo electrónico en Internet, fiable para la entrega de mensajes.	Implementación de Sendgrid para el envío de emails, tanto para actualizar la contraseña como para mandar emails a los dueños de los restaurantes.
Points SQS Queue	Mejora disponibilidad al ser asincrónico entre los componentes y hacer que la comunicación entre los mismos no dependa del estado de uno o del otro.	Implementación de colas de mensajes con el servicio de AWS “SQS” la cual tiene el objetivo de procesar de forma asíncrona algunas partes de procesos que no necesitan ser consideradas con rapidez. Esto permite que no se afecte a ninguna respuesta del sistema y que mientras tanto se desarrolle una actividad. Se utiliza respectivamente para: generar puntos cuando un usuario compra comida con efectivo o tarjeta, para restar los puntos de un usuario cuando los canjea por comida.
SocketIO	Basado en eventos, permite la comunicación bidireccional en tiempo real entre clientes y servidores. Mejora la performance.	Implementación de SocketIo para poder reflejar el cambio de estado de las órdenes en tiempo real.

Tabla 4.1.4: Catálogo de elementos de vista de asignación

Decisiones de diseño

Como se mencionó en las otras vistas, optamos por una arquitectura modular donde los distintos módulos se despliegan todos juntos, en conjunto con un servicio separado que se despliega independientemente al resto. Esta decisión se basa en la naturaleza y objetivos específicos de la POC, donde el enfoque principal no es la escalabilidad, sino la validación de conceptos clave y la eficiencia en el desarrollo. Además decidimos desplegar por separado el servicio de puntos ya que el cliente

especificó que quería que sea un servicio que luego pueda extraer a otro sistema de ser necesario, por lo que tenía que estar desacoplado al resto, tener su propia base de datos y entorno de despliegue.

Para desplegar los sistemas utilizamos tecnologías Cloud como explicamos en la parte de Stack tecnológico, particularmente AWS.

Para desplegar los servicios, empleamos un proceso de contenerización con Docker. Este método consiste en la creación automática de imágenes Docker locales para los dos servicios. Esto permite que las imágenes se utilicen para desplegar los servicios en AWS más fácilmente. Además, para simplificar la ejecución local del *backend* y evitar la necesidad de instalar dependencias adicionales, utilizamos Docker localmente, para esto además de crear el archivo DockerFile, creamos el archivo docker-compose que solo corriendo un comando “docker-compose up” permite crear el entorno local sin la necesidad de instalaciones previas.

Este procedimiento lo adoptamos debido a la interacción continua entre Git y AWS, que provocaba el despliegue automático de todo lo que se encontraba en el repositorio, incluyendo el archivo docker-compose.yml. Para prevenir que AWS ejecute automáticamente este archivo, decidimos moverlo fuera de la raíz de la aplicación.

Particularmente, para el despliegue de los dos servicios del *backend*, *TuBocado.MarketBE* y *TuBocado.PointsBE*, utilizamos AWS *Elastic Beanstalk*. Elegimos esto debido a su eficiencia en automatizar y simplificar la administración de aplicaciones en la nube. Elastic Beanstalk se ejecuta sobre Amazon EC2, y se encarga automáticamente de la configuración de servidores, el balanceo de carga y el escalado automático, lo que nos permitió enfocarnos más en las funcionalidades y menos en la gestión de la infraestructura. Este enfoque une la potencia y la flexibilidad de EC2 con la simplicidad de una plataforma gestionada, mejorando así la eficiencia en el despliegue y la operatividad del *backend*. Además al tener dockerizados(anglicismo derivado de docker) los servicios Elastic Beanstalk ofrece un entorno optimizado para ejecutar y administrar aplicaciones Docker, lo que

simplifica significativamente el proceso de despliegue y garantiza un rendimiento estable y confiable del sistema.

Los *backends* están integrados con AWS *Pipeline* lo que asegura un flujo de trabajo de integración continua. Cada vez que realizamos un *merge* en las rama main, se activa automáticamente un proceso en el *pipeline* de AWS lo que hace que se genere el proceso de construcción y una vez que esté terminado pasa a desplegar las actualizaciones en ElasticBeanstalk.

Por otro lado, los tres *frontends* se despliegan utilizando Amazon Amplify, lo que representa una elección estratégica para maximizar la eficiencia en el desarrollo y el despliegue. Esta elección hace hincapié al compromiso con la agilidad de desarrollo ya que Amplify permite conectar las instancias a los repositorios, particularmente a una *branch*, lo que hace que se pueda tener un despliegue continuo sin necesidad de manualmente activar el deploy.

La utilización de Amazon Relational Database Service (RDS) para alojar las bases de datos refleja la búsqueda de un rendimiento óptimo y alta disponibilidad. RDS ofrece una gestión simplificada de las bases de datos, lo que nos permite enfocarnos en la lógica de la aplicación sin la carga adicional de administrar la infraestructura de base de datos. Esta se comunica con los servicios de *backend* mediante el protocolo TCP/IP, cada uno mediante una base de datos impulsada por el motor Postgresql.

Otro de los servicios que utilizamos son los de Amazon Simple Queue Service (SQS) para conectar el *backend* modular y el servicio de puntos. Esto hace que se genere un acoplamiento débil, promoviendo la fiabilidad y seguridad. Las colas de mensajes facilitan la gestión de cargas de trabajo y la comunicación asincrónica entre servicios, lo que mejora la eficiencia y robustez del sistema lo cual es ideal para un servicio que posteriormente podrá ser utilizado por otros servicios. Además, la integración mediante HTTPS entre los diferentes componentes asegura que todas las comunicaciones sean seguras y cifradas, preservando la integridad y confidencialidad de los datos transmitidos.

Por lo tanto, la estrategia de despliegue para todo el ecosistema se basa en la automatización, utilizando AWS ElasticBeanstalk y AWS Amplify,

respectivamente, en conjunto con AWS *Pipeline*. En el *backend*, cada vez que se realiza un *merge*, se activa automáticamente el pipeline de AWS, que gestiona la construcción, las pruebas y el despliegue en Elastic Beanstalk lo que permite un fácil *rollback* en caso de problemas. De manera similar, para los frontends, AWS Amplify se conecta directamente a los repositorios de código fuente. Cada vez que realizamos un *merge* se desencadena un proceso de despliegue automático, donde Amplify construye y despliega la versión actualizada. Esta integración con AWS Pipeline y Amplify garantiza un flujo de trabajo eficiente que facilita la entrega continua.

4.2. Decisiones de arquitectura

¿Por qué monolito modular?

La primera decisión que tuvimos que tomar con respecto a la arquitectura fue si elegir un sistema basado en microservicios o una arquitectura monolítica, y de ser así, qué tipo de arquitectura monolítica debía ser. Luego de pasar por algunas iteraciones de *Design Thinking* y tener una primera versión del *Product Backlog* con los requerimientos principales que entrarían dentro del alcance del sistema, nos reunimos con el equipo para poder pensar el diseño de la arquitectura que luego validamos con expertos.

La elección de adoptar un enfoque monolítico o un modelo basado en microservicios fue una decisión muy importante para el proyecto, ya que fue necesario tomar en cuenta muchas características como el alcance, la simplicidad, los costos, entre otros. En primer lugar, una de las principales decisiones fue si la gestión de puntos se implementaría como un servicio independiente o no. Esta elección se basó en uno de los principales requerimientos no funcionales del cliente, destacando la importancia de tener una solución desacoplada para luego poder utilizar en otros tipos de sistemas. Tras evaluar las necesidades y especificaciones del proyecto, se optó por desarrollar este componente de manera que pudiera operar independientemente.

Con este enfoque definido, procedimos a listar todos los requerimientos no funcionales identificados hasta el momento, organizándolos en una tabla que se

presenta en su versión final en secciones posteriores del documento. Esta compilación de requerimientos no funcionales nos permitió tomar decisiones informadas sobre la arquitectura general del sistema. Específicamente, decidimos estructurar el servicio de gestión de puntos como un módulo independiente para facilitar su potencial comercialización por separado. Esta decisión se basó en el "US-15: Módulo de gestión de puntos desacoplado". Esta modularización asegura que el servicio de puntos pueda ser desarrollado, mantenido y vendido de manera independiente, sin afectar ni depender del resto del ecosistema de servicios, sean estos microservicios o un sistema monolítico.

Para tomar esta decisión, además de basarnos en los RNF que ya habíamos escrito con los atributos de calidad correspondientes, realizamos dos diagramas con las dos opciones y una tabla comparativa entre monolito y microservicios para ayudarnos a visualizar de mejor manera las ventajas y desventajas de cada estilo arquitectónico.

En la tabla a continuación, presentaremos una tabla comparativa, seguida de dos diagramas iniciales: uno representando la arquitectura basada en microservicios (ilustración 4.2.1) y otro ilustrando un enfoque monolítico (ilustración 4.2.3), ambos incluyendo el sistema de gestión de puntos.

Atributo	Monolito	Microservicios
Definición	Una única base de código que incluye todas las funciones del sistema.	Conjunto de servicios pequeños y autónomos que trabajan juntos, cada uno con su propia base de código.
Conocimiento del negocio	Generalmente se utiliza más cuando se está iniciando y no se sabe tanto del negocio.	Se utiliza cuando el conocimiento del negocio es más profundo.
Reutilizaciones de componentes	Se puede reutilizar porciones de código únicamente.	Se puede reutilizar cada uno de los servicios.
Escalabilidad	Peor escalabilidad ya que toda la aplicación debe escalar junta.	Mejor escalabilidad ya que cada servicio puede escalar independientemente del otro.
Simpleza de implementación	Muy fácil de implementar	Más complejo de implementar que un monolito ya que hay que tener en cuenta la comunicación entre servicios y la seguridad.
Modificabilidad	Cada modificación implica un nuevo deploy de toda la aplicación.	Las modificaciones pueden ser únicamente en los servicios que aplican.
Performance	Muy buena performance ya que	Peor performance ya que para conseguir

	tiene todos los datos en un mismo lugar.	datos generalmente se tienen que conectar diferentes servicios lo que hace que demore más.
Despliegue	Un único artefacto o proceso, simplificando el despliegue inicial.	Despliegue independiente por servicio, lo que aumenta la complejidad operativa pero mejora la escalabilidad.
Tiempo al mercado	Rápido inicialmente debido a la simplicidad del desarrollo y despliegue.	Puede ser más lento inicialmente debido a la necesidad de establecer patrones de comunicación y operaciones distribuidas.

Tabla 4.2.1: Tabla comparativa entre monolito y microservicios

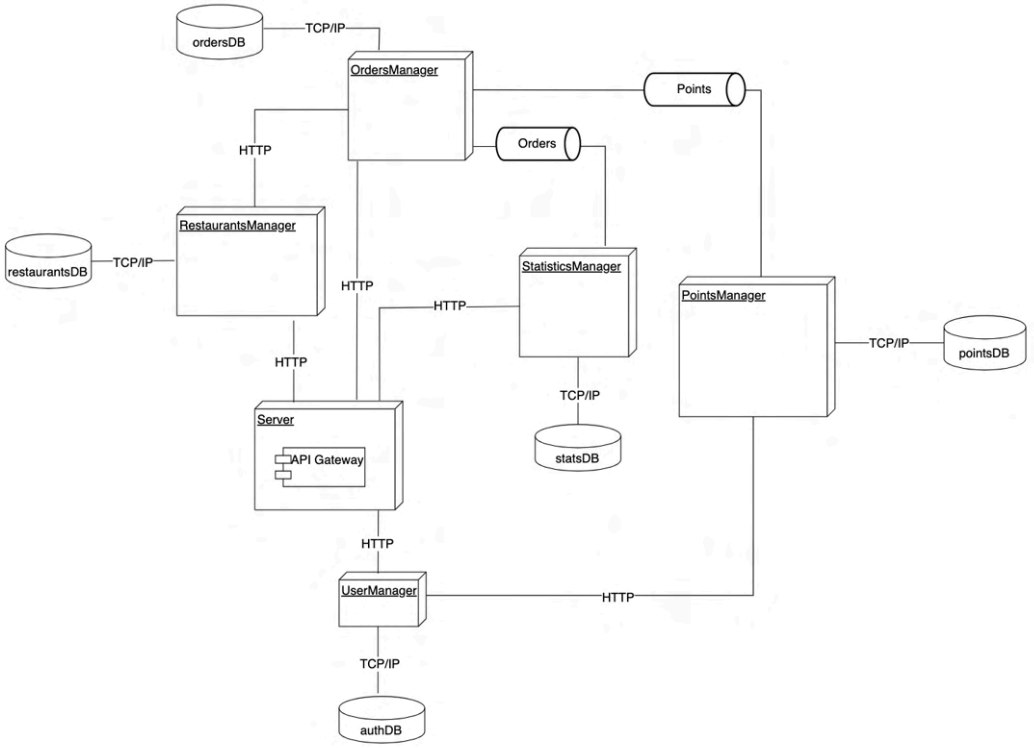


Ilustración 4.2.1: Diagrama inicial de arquitectura de microservicios

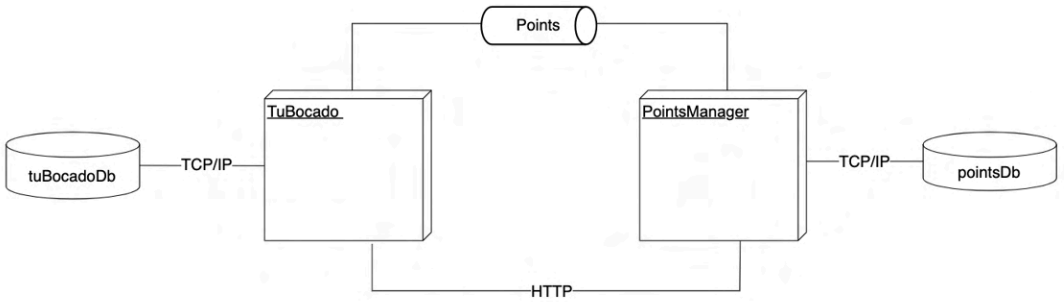


Ilustración 4.2.2: Diagrama inicial de arquitectura monolítica modular con un servicio de puntos externo

En base a esta información, y siguiendo los principios fundamentales de arquitectura de *software*, específicamente aquellos articulados por *Mark Richards y Neal Ford en su obra seminal "Fundamentals of Software Architecture"* [79] a través de su concepto de "estrellas de la arquitectura", fuimos rellenando la tabla y viendo los pros y contras de cada estilo. En este libro, se le asignan estrellas a los atributos de calidad indicando que se favorece, por ejemplo, si hay 5 estrellas en modificabilidad es porque favorece mucho a la modificabilidad.

Decidimos que para la POC la arquitectura elegida consistiría en un *backend* monolítico modular complementado por un servicio separado dedicado al manejo de puntos, conocidos como "Bocados".

Esta elección se debe a que, al ser una POC, la simplicidad y rapidez de desarrollo es muy importante ya que lo que se busca es probar rápidamente las funcionalidades centrales. La arquitectura modular permite una integración y prueba más sencillas, evitando la complejidad de los microservicios y como se indica en el libro, el monolito es una de las mayores ventajas que ofrece, ya que tiene muchas estrellas en simplicidad, mientras microservicios tiene solo una estrella en esta categoría.

Por otro lado en relación a la modificabilidad y modularidad, si bien los microservicios ofrecen una mejor modularidad, decidimos realizar no cualquier monolito sino un monolito modular el cual si bien es un monolito, consiste en una estructura interna claramente definida en módulos que pueden desarrollarse de manera casi independiente. Esto nos da las ventajas de los monolitos como puede ser la simpleza y despliegue más simple pero también con la ventaja de estar separado en módulos definidos por *Bounded Contexts* que beneficia la coherencia semántica y reduce el acoplamiento haciendo que los módulos se conecten mediante interfaces bien definidas lo que permite un grado de reutilización y separación bueno.

Otro de los puntos a favor del monolito es el despliegue en conjunto. Esto hace que los módulos se desplieguen todos juntos, asegurando así una coherencia en el entorno de prueba y reduciendo la complejidad operativa. De esta forma se

evita perder tiempo en el proceso de despliegue y se acelera el inicio del proyecto, lo cual es clave para la etapa en la que nos encontramos.

Existe la probabilidad de que en un futuro una arquitectura del tipo microservicios sea más adecuada para el proyecto. Esto puede ser por una variedad de razones tales como: se busca una mejor escalabilidad, se tiene un mejor dominio del negocio o tal vez el equipo crece mucho y se busca tener equipos diferenciados en lenguajes para cada servicio. Sin embargo, se puede decir que la elección actual de un monolito modular no es contraproducente sino que al elegir los módulos basándonos en *Bounded Contexts* y haciendo que no se acoplan entre ellos, estamos favoreciendo la futura separación de los módulos en microservicios para que el dia de mañana se pueda transformar a dicha arquitectura. Es decir, hoy presenta una mayor cantidad de ventajas el monolito para las necesidades de este sistema, pero la división en módulos en cierta forma prevé la futura división en microservicios, que como ya dijimos, creemos que es probable que suceda en un futuro.

Adicionalmente, una de las características que tuvimos en cuenta para tomar esta decisión es la ausencia de necesidad de escalabilidad. Dado que la POC no está diseñado para operar a gran escala, la escalabilidad no es una preocupación importante en esta etapa. Esto nos permite concentrarnos en otros aspectos, como la funcionalidad, lo que hace que un monolito modular sea más adecuado. Por lo que creemos que esta estrategia resulta ideal para nuestro sistema que aún no justifica la complejidad de una arquitectura basada en microservicios, pero que demandan una organización modular para mantener la claridad del código y facilitar su escalabilidad futura.

Sumado a esto, tener una estructura de microservicios es mucho más costosa que la de un monolito y, al ser una POC, cuidar los costos es importante ya que aún no se tiene un gran capital. Hacer un *rollback* de errores es muchísimo más complejo y costoso en microservicios, y aún más teniendo en cuenta que se está desplegando en la nube, donde cada uno de los servicios y cada funcionalidad que se tiene la cobran por uso, por almacenamiento, por cantidad de datos, por cantidad de *rollbacks* y reintentos, etc.

Por último, con lo que respecta a la estructura del monolito modular y la del servicio de puntos, decidimos utilizar una adaptación del patrón de Layers con Clean Architecture. Citamos lo que mencionamos en la sección de decisiones de diseño de la vista layers: “a través de Clean Architecture, beneficiamos la modularidad, modificabilidad y seguridad del sistema. La modularidad y modificabilidad se ven beneficiadas ya que Clean Architecture promueve una clara separación entre la lógica de negocio, la interfaz de usuario, el almacenamiento de datos y otras preocupaciones externas. Además, la separación de preocupaciones y la definición de capas claras contribuyen a una gestión de la seguridad más centralizada y coherente.”.

Luego de tomar estas decisiones tuvimos dos instancias con el Arquitecto de software Gastón Mosqués. En la primera instancia le planteamos las decisiones que habíamos tomado en base a los atributos de calidad, naturaleza del proyecto, tablas comparativas y diagramas y allí pudimos validar que efectivamente realizar un monolito modular en este caso resultaba muy beneficioso. La terminología de “monolito modular” fue sugerida por el experto para describir nuestro diseño. Aunque en las bibliografías utilizadas este concepto suele presentarse simplemente como "monolito", con un énfasis adicional en su naturaleza "modular", fue el experto quien nos dijo cómo en la industria se utiliza el término “monolito modular” para referirse a esta estructura que beneficia específicamente los atributos de calidad que presentamos. Luego tuvimos una segunda instancia con el experto para poder validar la arquitectura una vez que estaba finalmente decidida con los diagramas finales y los módulos definidos.

Por lo tanto, la estructura de despliegue de esta POC y la elección de una arquitectura modular reflejan un enfoque práctico y más eficiente, adecuado para las expectativas, necesidades y restricciones del proyecto en esta etapa. Adicionalmente, la inclusión de un servicio desplegado de manera independiente hace referencia a un requerimiento no funcional específico del cliente. Este servicio separado permite cumplir con las necesidades particulares sin comprometer la cohesión del sistema principal. Esto subraya nuestra capacidad para adaptar y personalizar soluciones, manteniendo al mismo tiempo una arquitectura coherente y eficiente para el sistema en su conjunto.

4.3. Stack Tecnológico

En esta sección presentaremos las tecnologías seleccionadas para nuestro proyecto. En primer lugar, mostraremos una imagen ilustrativa de las tecnologías para dar un mejor contexto y luego pasaremos a explicar las principales tecnologías y porqué se eligieron ya que a cada una la seleccionamos tanto por el deseo de explorar nuevas tecnologías, como por el alineamiento con las necesidades específicas del proyecto.

Para cada tecnología, hicimos una investigación, evaluando diferentes opciones e intercambiando opiniones con el cliente. En el anexo, adjuntamos el diagrama con un resumen de tecnologías usadas, y a continuación se detallan las razones detrás de la elección de cada una, enfatizando cómo estas se alinean con los objetivos y requisitos del proyecto.

La elección de las tecnologías adecuadas y el proveedor de infraestructura para nuestro proyecto implicó un proceso metódico y detallado, enfocado en cumplir con las expectativas del cliente y las necesidades del proyecto. Este proceso se desglosa en cuatro pasos principales.

Paso 1: Evaluación mediante Matriz de Pugh: iniciábamos nuestro análisis creando una matriz de Pugh para comparar las posibles opciones. Este enfoque nos permitió definir y ponderar los atributos críticos de la tecnología o herramienta que se buscaba analizar.

Paso 2: Selección de Opciones a Comparar: definíamos las tecnologías y proveedores a evaluar, realizando una tabla comparativa, basándonos en los atributos que habíamos ponderado en la matriz de Pugh, se hablará más en detalle sobre estas en la sección de Selección de Tecnologías en Estrategia de Gestión del Proyecto.

Paso 3: Comparación Detallada y Elección: tras la comparación, realizábamos una comparación detallada, observando las tablas realizadas de cada una de las opciones y realizábamos la elección por una herramienta.

Paso 4: Validación mediante Spikes Tecnológicos: para algunas de las herramientas y tecnologías implementábamos *spikes* tecnológicos para asegurarnos de que cumplieran con nuestros requisitos y expectativas. Esta etapa confirmó

nuestras elecciones en algunos casos, permitiéndonos avanzar con confianza en la implementación.

Una vez aclarados los pasos, a continuación nos centraremos en las tres principales elecciones tecnológicas realizadas para el proyecto, detallando específicamente el proveedor de infraestructura, el *framework* de desarrollo del *frontend* y el *framework* de desarrollo del *backend*.

Se podrán encontrar las tablas comparativas y los detalles de las otras selecciones tecnológicas y elecciones que acompañan al desarrollo del proyecto, en la sección de selección de tecnología en el anexo, ofreciendo una visión completa del proceso de decisión y justificación detrás de cada herramienta y tecnología seleccionada.

Proveedor de infraestructura

Para poder decidir el proveedor de infraestructura, tuvimos en cuenta todos los pasos mencionados anteriormente, pero principalmente tuvimos en cuenta la solicitud del cliente ya que este tuvo una participación muy activa en la decisión de la misma.

Inicialmente, optamos por Azure como nuestro proveedor de infraestructura en la nube, esto se debe a que el cliente nos había planteado que ya tenía otros proyectos alojados en Azure y contaba con técnicos que nos podrían ayudar a responder dudas en caso de ser necesario, por lo que, parecía una buena opción, especialmente teniendo en cuenta las preferencias del cliente. Incluso nos capacitamos específicamente en Azure para adaptarnos mejor a esta plataforma ya que ninguno tenía experiencia previa con la misma. No solo esto, sino que tuvimos charlas técnicas con uno de los técnicos del cliente para poder comprender mejor el ecosistema.

Sin embargo, a medida que fue avanzando el proyecto, el cliente expresó interés en explorar nuevas posibilidades y optó por cambiar a AWS. En ese momento volvimos a evaluar y observar la tabla comparativa para poder ayudar mejor en el cambio de decisión. Este cambio requería una adaptación rápida de nuestro equipo, que ya se estaba familiarizado con Azure. La transición a AWS,

aunque fue inicialmente desafiante ya que algunos tenían experiencia utilizándolo por la Universidad pero ninguno lo había aplicado en su trabajo, resultó ser beneficiosa. Esto es porque AWS ofrece una gran variedad de servicios y herramientas, lo que nos proporciona una flexibilidad y una capacidad de explorar muchos servicios para un futuro.

Atributos	Azure	AWS
Alcance global	Amplia presencia global, pero menos regiones que AWS.	La mayor red global con más regiones y zonas de disponibilidad.
Servicios ofrecidos	Amplia gama de servicios, fuerte en integraciones con herramientas Microsoft.	La mayor variedad de servicios y herramientas en la nube.
Precios	Estructura de precios competitiva.	Generalmente considerado más económico a gran escala, con más opciones de precios flexibles.
Comunidad	Comunidad sólida, especialmente entre usuarios de productos Microsoft.	Comunidad más grande y diversa, con una amplia gama de recursos de aprendizaje.
Facilidad de uso	Interfaz amigable, integración sencilla con productos Microsoft.	Curva de aprendizaje más empinada.
Soporte y documentación	Muy buena documentación y soporte técnico, además el cliente ofrece un integrante técnico experto en Azure.	Documentación extensa y soporte técnico robusto. Además, contamos con varios docentes de la Universidad expertos en AWS.
Seguridad	Buena seguridad.	Muy buena seguridad.
Fiabilidad	Muy confiable.	Altamente confiable.
Deploy de aplicaciones	Servicios como Azure App Service y Azure Kubernetes Service.	Elastic Beanstalk y Amplify para despliegues eficientes y escalables.
Bases de datos	Azure SQL Database, Cosmos DB entre otros.	Amazon RDS.

Tabla 4.3.1: Tabla comparativa proveedor de infraestructura

Framework de desarrollo del *frontend*

Para la selección de frameworks en el *frontend*, el cliente no exigió ninguna restricción. Dado que sabíamos que íbamos a utilizar algún framework de JavaScript evaluamos los siguientes Frameworks y Librerías, ReactJs, AngularJs y VueJs.

Para el desarrollo de nuestro sistema de entregas de comida, con múltiples interfaces para usuarios, *managers* de TuBocado, consumidores y restaurantes, consideramos los tres frameworks pero elegimos ReactJs como nuestra tecnología principal. Para esta decisión, nos basamos en una exhaustiva evaluación que incluyó todos los pasos mencionados anteriormente. En la tabla comparativa, se

examinan aspectos clave como la reutilización de código, el rendimiento, la seguridad y el soporte comunitario, entre otros. Esto nos proporcionó una visión clara de las ventajas de ReactJs frente a las otras opciones.

Elegimos *ReactJs* sobre las otras opciones ya que nos ofrece un ecosistema robusto y una comunidad mas activa que *AngularJS* y *VueJS*. Esto es importante ya que asegura un mayor acceso a una variedad de recursos y soporte técnico. Además, también nos garantiza interfaces de usuario intuitivas, fácilmente mantenibles y reutilizables, lo cual es muy importante para mejorar la experiencia del usuario en nuestro sistema.

Por otro lado, esta decisión se basó también en la amplia experiencia que nuestro equipo tiene con *ReactJs*, superando a la que tenemos con *AngularJS* y *VueJS*. Sumando a esto, existió una participación activa del cliente en el proceso de decisión y su postura fue muy positiva con respecto a la utilización de *ReactJs*.

Atributos	ReactJS	AngularJS	VueJS
Definición de la tecnología	Biblioteca JavaScript para construir interfaces de usuario.	Framework estructurado para aplicaciones web dinámicas.	Framework progresivo para construir interfaces de usuario.
Reutilizacion de codigo	Muy alta, con componentes reutilizables.	Buena, con directivas y servicios reutilizables.	Alta, con un sistema de componentes sencillo.
Performance	Alto rendimiento debido a la Virtual DOM.	Buen rendimiento, pero menos eficiente debido al real DOM.	Buen rendimiento, similar al del React.
Lenguaje de programación	JSX (JavaScript XML), con opción de usar TypeScript	TypeScript	JavaScript, con una opción para usar TypeScript.
Seguridad	Seguro.	Medidas de seguridad integradas.	Seguro.
Soporte	Muy amplia y extensa y en crecimiento.	Grande, pero en declive en comparación con React.	Creciente.
Costo	Gratuito y de código abierto.	Gratuito y de código abierto.	Gratuito y de código abierto.
Curva de aprendizaje	Curva de aprendizaje más suave que AngularJS.	Complejidad inicial más alta.	Curva de aprendizaje baja y fácil de entender.
Fiabilidad	Muy confiable, respaldado por Facebook.	Confiable, respaldado por Google.	Adoptado y considerado confiable.

Tabla 4.3.2: Tabla comparativa proveedor de framework de desarrollo de *frontend*.

Framework de desarrollo del *backend*

Para esta decisión nos basamos en varios factores críticos. En primer lugar, uno de los factores fue que NodeJs se destaca por su capacidad de ejecutar operaciones asincrónicas, lo que lo convierte en una opción mejor para sistemas que necesitan manejar operaciones en tiempo real; como nuestro servicio de *delivery*. Esta habilidad para manejar muchas solicitudes simultáneamente se debe a que su arquitectura está basada en eventos, y es lo que lo hace más adecuado para nuestro proyecto que, por ejemplo, Laravel o Ruby on Rails. Estos últimos, aunque son populares y buenos, están más orientados a patrones de solicitud-respuesta tradicionales y no están tan optimizados como NodeJs para la alta concurrencia y procesamiento en tiempo real.

Otro de los puntos importantes que tuvimos en cuenta fue la compatibilidad y cohesión con los *frameworks* de *frontend* seleccionados previamente. Al utilizar JavaScript tanto en el *frontend* como en el backend logramos una uniformidad en el desarrollo, lo que simplifica el proceso y reduce la curva de aprendizaje de los integrantes del equipo que no tenían conocimiento con ReactJs o NodeJs. Esto hace que el desarrollo de aplicaciones sea mucho más rápido, y por lo tanto el usuario puede acceder a una mejor experiencia de uso.

Otro de los fuertes puntos es la popularidad y la amplia comunidad de NodeJs que nos da acceso a una extensa biblioteca de módulos y herramientas a través de *npm (Node Package Manager)*.

Por último, al elegir el framework para el *frontend*, tuvimos en cuenta que tres de los integrantes tenían una amplia experiencia con NodeJs, y dado que el cliente estaba de acuerdo, ya que su equipo trabaja con este framework, optamos por este.

Atributos	NodeJS	Ruby on Rails (Ruby)	Laravel(PHP)
Lenguaje de programación	JavaScript, se puede integrar con typescript	Ruby	PHP
Rendimiento	Alto debido a operaciones no bloqueantes.	Bueno, optimizado para un desarrollo rápido.	Bueno para web apps.
Ecosistema y Bibliotecas	Extensa colección de módulos a través de npm.	Amplio.	Bueno en características, con un ecosistema de paquetes sólido.

Comunidad	Muy activa y en crecimiento.	Fuerte y enfocada en la calidad del código.	Muy activa, especialmente en el desarrollo web.
Curva de aprendizaje	Relativamente fácil, sobre todo para desarrolladores JavaScript.	Curva de aprendizaje inicialmente empinada.	Relativamente fácil, con buena documentación.
Mantenibilidad	Alta con un código bien estructurado.	Alta, si se siguen las convenciones de Rails.	Alta, por su arquitectura MVC.
Seguridad	Seguro.	Buena.	Seguridad robusta.
Integración con *Frontend*	Natural y muy buena para aplicaciones JavaScript/React.	Compatible, pero sin integraciones específicas con frameworks de *frontend*.	Compatible con cualquier *frontend*, integración manual requerida.
Uso en la industria	Muy popular en aplicaciones en tiempo real y APIs.	Popular en startups.	Muy utilizado en aplicaciones web.

Tabla 4.3.3: Tabla comparativa proveedor de infraestructura de *backend*

4.4. Tácticas

En esta sección mencionaremos los requerimientos no funcionales agrupados por atributos de calidad y las tácticas aplicadas en el desarrollo de nuestro proyecto. Iniciaremos con una tabla que sintetiza los requerimientos no funcionales identificados, junto con los atributos de calidad relacionados a los mismos, ofreciendo una visión global de los atributos de calidad centrales los cuales fueron fundamentales para la estructuración del proyecto. A continuación, profundizaremos en cada atributo de calidad, detallando cómo fueron abordados y qué tácticas específicas empleamos para cumplir con estos, asegurando así un correcto diseño de nuestra solución.

Requerimientos no funcionales agrupados por tipo de atributo de calidad

ID Requerimiento No Funcional	Agrupación de atributo de calidad	Descripción
US-1 Cambios del cliente	Modificabilidad	El sistema debe desarrollarse de modo que si el cliente plantea durante el desarrollo algún cambio, el costo de llevarlos a cabo sea lo más bajo posible.
US-2 Extensibilidad	Modificabilidad	El sistema debe ser desarrollado de forma que si se desea incorporar nuevas funcionalidades, conlleven el menor coste posible.
US-3 Reutilización	Modificabilidad	El sistema deberá estar modularizado para poder reutilizar partes de él en futuros proyectos del cliente.

US-4 Métodos de autenticación y autorización	Seguridad	El sistema debe contar con medidas de seguridad para el acceso a datos y acciones realizadas por el usuario según su rol, haciendo autenticaciones para cada petición de ser necesario.
US-5 Métodos de pago	Seguridad	El sistema debe contar con medidas de seguridad a modo de proteger las transacciones de pago de los clientes utilizando una api de terceros confiable.
US-6 Fácil de aprender	Usabilidad	El sistema debe permitir el fácil aprendizaje a los clientes, restaurantes y managers brindando una interfaz gráfica que cumpla con las Heurísticas de Nielsen.
US-7 Interfaz atractiva	Usabilidad	El sistema debe brindar una interfaz atractiva y eficiente que cumpla con los prototipos iniciales.
US-8 Responsive	Usabilidad	La web de los clientes debe ser responsive para poder adaptarse a todas las pantallas de los móviles y computadoras
US-9 Compatibilidad de navegadores	Disponibilidad, Usabilidad	La aplicación web debe ser compatible y funcionar de manera óptima en los navegadores Safari y Chrome, garantizando una experiencia de usuario consistente y satisfactoria en ambas plataformas
US-10 Reintento de generación de puntos	Disponibilidad	Se debe poder reintentar la generación de puntos en caso de que no logren cumplirse sea cual sea el motivo.
US-11 Disponible todo el tiempo	Disponibilidad	La aplicación debe tener una disponibilidad del 98% para que los usuarios puedan realizar órdenes en cualquier momento.
US-12 Calidad de las imágenes de los platos	Performance	La calidad de imágen de los platos y productos debe ser mayor o igual a 720p
US-13 Tiempos de respuesta notificaciones	Performance	Se debe poder ver el estado de una orden en tiempo real. Tanto del lado del dueño de restaurante como del lado del cliente.
US-14 Tiempo de respuesta creación ordenes	Performance	Se debe poder procesar una orden nueva en menos de 5s.
US-15 Módulo de gestión de puntos desacoplado	Modularidad	El sistema de puntos debe ser un servicio aparte del resto, de modo que pueda ser fácilmente integrado en otros sistemas o aplicaciones sin requerir modificaciones extensivas.
US-16 Comunicación entre servicios	Interoperabilidad	Debe asegurar la integración fluida con el servicio de gestión de puntos y otros posibles servicios.
US-17 Gestión de errores y fallas	Testeabilidad	El sistema debe brindar la información necesaria que permita hacer un diagnóstico preciso y eficiente sobre las causas de cualquier error que se presente.
US-18 Utilizar contenedores	Deployabilidad	El sistema debe ser contenerizado utilizando Docker, permitiendo un despliegue continuo.
US-19 Integración continua	Deployabilidad	Los sistemas deben implementar prácticas de Integración Continua (CI) y Despliegue Continuo (CD) para mejorar la deployabilidad del sistema. Esto asegura que las actualizaciones del *software* puedan ser desarrolladas, probadas y desplegadas de manera rápida, manteniendo la calidad del sistema.

Tabla 4.4.1: Tabla de requerimientos no funcionales con atributos de calidad asociados.

Modificabilidad y Modularidad

La Modificabilidad y Modularidad se encuentran dentro del atributo de calidad Mantenibilidad según *Software* Architecture in Practice[57], modificabilidad se refiere a la facilidad con la que un sistema puede ser modificado para cambiar, corregir, o adaptar su funcionalidad y rendimiento sin introducir defectos, mientras que modularidad es el grado en el cual un sistema o proyecto de *software* puede ser dividido en componentes más pequeños, que pueden ser desarrollados, probados, y comprendidos de manera independiente pero trabajan juntos como un conjunto cohesivo.

La modificabilidad y la modularidad son unos de los atributos más importantes en nuestro proyecto, a pesar de ser una POC, uno de los requisitos que nos solicitó el cliente es que el proyecto pueda adaptarse rápidamente a nuevos requerimientos y cambios, tanto del cliente como del entorno tecnológico. Además, solicitó expresamente que el sistema de puntos esté desacoplado del resto para poder utilizarlo en otros proyectos.

Para poder definir cómo beneficiar la mantenibilidad (que engloba la modificabilidad y la modularidad), evaluamos no sólo determinados tipos de arquitectura para estructurar nuestro sistema sino que también tuvimos en cuenta varias tácticas.

En primer lugar, como mencionamos en la vista de componentes y conectores y en la de módulos, utilizamos una arquitectura basada en DDD (Domain Driven Design) aplicando *Bounded Contexts*, lo cual nos permite una separación del sistema en módulos basados en el dominio para mejorar la reutilización y reducir el acoplamiento; al reducir en módulos basados en dominio hace que mejore la cohesión semántica ya que cada funcionalidad se encuentra agrupada por sector del dominio. Esta arquitectura es una modificación del patrón Layers y principalmente nos basamos en la arquitectura Clean. Esta arquitectura, su atributo principal es la modularidad justamente lo que busca es beneficiarla.

Al aplicar el concepto de *Bounded Context* de DDD en nuestra arquitectura, nos permite definir límites claros alrededor de cada dominio o subdominio dentro del sistema. *Bounded Context* lo que busca es encapsular toda la lógica asociada al

dominio, haciendo que los contextos tengan que interactuar entre sí mediante interfaces bien definidas. Gracias a esto, se reduce aún más el acoplamiento y facilita la integración y reutilización de componentes. Al combinar esto con principios de *Clean architecture*, no solo seguimos una separación de preocupaciones basada en capas sino que también garantizamos que cada capa dentro de un contexto delimitado esté alineada con las necesidades específicas del dominio, mejorando la cohesión semántica y la flexibilidad del diseño.

Por otra parte, con lo que respecta a la modificabilidad, anticipamos que en el futuro, los dueños de restaurantes podrán tener restaurantes asociados a una misma cuenta. Este tema se desarrollará con mayor detalle en el capítulo de Pasos a seguir, pero es importante mencionar que este aspecto se consideró al diseñar la arquitectura actual. Dentro de los módulos establecidos está el del restaurante, aquí se encuentra toda esta lógica ya que la relación entre restaurantes y sus dueños es clave, esto se debe a que ambos elementos están estrechamente vinculados y evolucionan conjuntamente. Esta consideración también se reflejó en la estructuración de las tablas de la base de datos, dado que, la arquitectura del *backend* está preparada desde ahora para admitir múltiples restaurantes por dueño. Aunque esta funcionalidad aún no está implementada en el *frontend*, la infraestructura *backend* ya está diseñada para facilitar su integración en el futuro.

Con lo que respecta a las tácticas, aplicando lo descrito estamos aplicando varias de ellas, en primer lugar utilizamos la técnica Encapsulación, al utilizar *Bounded Context* para encapsular la lógica de dominio y ocultar los contextos de cada módulo, en conjunto de esto estamos utilizando el uso de intermediarios para comunicar los diferentes módulos mediante interfaces definidas permitiendo así la modificación de los módulos sin afectar los demás. También se aplica la cohesión semántica ya que cada módulo se agrupa mediante funcionalidades relacionadas que se cambian por las mismas razones lo que hace que exista una alta cohesión semántica y facilite su mantenimiento.

Otra de las tácticas aplicadas es la separación de preocupaciones ya que al seguir los principios de *Clean Architecture* nos aseguramos una clara definición de los módulos con responsabilidades únicas, lo que hace que exista una clara separación de preocupaciones. También aplicamos la táctica de anticipación de

cambios esperados ya que, al estar basada en dominios, se puede anticipar futuros cambios en los requerimientos permitiendo así una adaptación más rápida.

Por otro lado también se pueden ver aplicadas las tácticas en otras cosas no solo en la decisión de la arquitectura, por ejemplo en la implementación de HTTP y colas de mensajes como mecanismo de intermediación, esto hace que se pueda cambiar o reemplazar el servicio de puntos sin alterar todo el servicio principal.

Por último, una de las tácticas utilizadas es la de diferir enlaces. Para abordar esta táctica definimos y utilizamos variables de entorno, tanto para la información sensible como para datos que cambian según el entorno en el que se encuentra (como pueden ser claves de test y de producción). Gracias a esto generamos una mayor flexibilidad para cambiar variables de la configuración sin necesidad de frenar o recompilar el sistema lo que hace que no se generen referencias y dependencias directas a las variables.

Seguridad

La seguridad se refiere a la capacidad de un sistema para proteger la información y datos de accesos no autorizados, garantizando al mismo tiempo el acceso a personas y sistemas autorizados.[57]

Con lo que respecta al atributo de seguridad, para decidir cómo favorecerlo en base a nuestros requerimientos no funcionales involucramos varias tácticas de seguridad recomendadas por el libro "Software Architecture in Practice, Third Edition" por Len Bass, Paul Clements, y Rick Kazman, adaptándolas a las necesidades específicas de nuestro sistema para garantizar un alto nivel de seguridad.

Uno de los puntos que tuvimos en cuenta es la protección a los datos, pensamos en un sistema con roles definidos que se centra en la autenticación y autorización de usuarios mediante el uso de JWT (Jason Web Tokens), gestionado por una biblioteca externa llamada *jsonwebtoken*. Para esto utilizamos el patrón llamado Autenticación Basada en Token con JWT, este se basa en un sistema que genera y valida JWTs para usuarios autenticados, incluyendo información sobre sus roles dentro del token que permite un control de acceso granular a diferentes partes

de la aplicación basado en la identidad y roles del usuario. Cada token está conformado por tres partes. En primer lugar se encuentra el encabezado donde se coloca el algoritmo de encriptación que utilizamos, luego está el contenido que es el que tiene la información del usuario con su rol y que utiliza el algoritmo de encriptación del encabezado y por último está la clave privada para la encriptación.

En nuestro caso, el flujo se basa en que los usuarios se identifiquen y reciban un JWT, un token que luego será utilizado para acceder a funcionalidades que requieren permisos especiales. Este JWT además incluye los roles del usuario y de qué usuario se trata, lo que nos permite no tener que guardar para que token pertenece cada usuario, sino que se externaliza. Al haber muchos tipos de usuarios que hacen uso de nuestro servicio, decidimos centralizar este proceso de validar y decodificar los tokens en un único lugar, además existe un middleware asignado para cada tipo de usuario donde llamamos a las funciones que se encarga de autenticar y luego autorizar, verificando que el usuario pertenezca al rol correspondiente, con esto buscamos cumplir con el US-4. Algunas de las tácticas aplicadas son la autenticación y autorización de actores ya que, como mencionamos anteriormente, utilizamos *middlewares* dependiendo de cada rol en donde verificamos que el actor es quien dice ser y además asegura que esos actores autenticados tienen los privilegios para acceder a lo que intentan acceder.

Por otro lado, una vez que el usuario se identifica, encriptamos su contraseña. No solo esto, sino que también encriptamos algunos datos sensibles como datos de las transacciones de los pagos de las órdenes para proteger datos sensibles de usuarios. Aquí aplicamos algunas tácticas como Encriptación de datos e Integridad de los datos. Además para poder cumplir con el US-5, no solo encriptamos los datos sensibles sino que decidimos externalizar el proveedor de pagos por temas de seguridad justamente para no manejar estos datos. Al realizar esto, aplicamos algunas tácticas como Limitar el Acceso y Exposición ya que se limita el acceso a los datos de pago dentro de nuestro sistema, asegurando que solo el proveedor de servicios de pago externo, que está equipado para manejar esos datos de manera segura, tenga la capacidad de acceder y procesar dicha información lo que limita la exposición a ataques dirigidos a robar datos de pago.

Vinculado a esto, otra de las razones por las que decidimos utilizar RDS es por la seguridad ya que al derivarlo a un proveedor externo nos aseguramos de que este se va a encargar de arreglar cualquier tipo de falla en caso de que ocurra alguna respecto a la seguridad del ambiente.

Por último, con lo que respecta a la comunicación entre el servicio de *TuBocado.MarketBE* y el de *TuBocado.PointsBE* mediante API REST, también se encuentra autenticado. Para autenticar ambos servicios utilizamos un Secret Token el cual es un token secreto que únicamente ambos conocen , de esta manera nos aseguraremos que se están enviando datos confidenciales de quienes dicen ser, esto también se basa en la táctica de autenticar autores.

Por lo tanto creemos que basándonos en las tácticas y al utilizar JWT como mecanismo de seguridad en las partes del sistema generamos un sistema muy beneficioso, que nos evitó tener que gestionar sesiones, métodos de pago y sobrecargar la base de datos con solicitudes constantes. Esta decisión acompaña mucho a la de performance, siempre teniendo en cuenta la Seguridad.

Usabilidad

La usabilidad se refiere a qué tan fácil es para el usuario realizar una tarea deseada y el tipo de soporte al usuario que brinda el sistema [57]. Creemos que la usabilidad es uno de los atributos más importantes, sobre todo para la POC ya que es lo que el usuario verá y lo que se va a validar conceptualmente.

Para poder abordar este atributo de calidad tuvimos en cuenta las Heurísticas de Nielsen [58] como se menciona en los requerimientos no funcionales. Para poder llegar el diseño del sistema no solo nos basamos en las heurísticas sino que recorrimos un camino extenso.

Primero construimos pretotipos (esto está descrito más en detalle en el capítulo de *Discovery*), para posteriormente construir los prototipos, lo que nos permitió comenzar a desarrollar. En cada *Release* hicimos sesiones tanto con los usuarios como dueños de restaurantes y PO para no solo validar la funcionalidad sino la UI para que esta sea lo más fácil de utilizar. Adicionalmente a esto, validamos la UX/UI con diseñadores expertos.

A continuación, detallaremos cada una de las heurísticas de Nielsen cómo fueron abordadas. Además, en el anexo se detallan algunos ejemplos de evidencias de la aplicación de heurísticas que se mencionan a continuación. Ver en anexo evidencia de aplicación de heurísticas.

1. **Visibilidad de sistema:** *el sistema siempre debe mantener informados a los usuarios sobre lo que está sucediendo, a través de comentarios apropiados dentro de un tiempo razonable.* [58]

   Esta primera regla viene a decir que el usuario debe recibir *feedback* cuando realiza una acción. Para cumplir con esto incorporamos varios ítems para lograr la visualización del sistema, uno de ellos es la incorporación de *loaders*. Por otro lado, incorporamos mensajes de error y mensajes de éxito cuando el usuario logra realizar una acción indicando cualquier tipo de comportamiento en el sistema.

2. **Coincidencia entre el sistema y el mundo real:** *el sistema debe hablar el idioma de los usuarios, con palabras, frases y conceptos familiares para el usuario, en lugar de términos orientados al sistema. [58]*

   Es importante usar palabras y conceptos que le sean familiares al usuario y le permitan comprender rápidamente qué está pasando. Por esto, para la nomenclatura de los sistemas utilizamos los términos comunes en varios de los sistemas de *delivery* de comida para que estos sean los más amigables posibles. Además de esto incorporamos íconos para acompañar el reconocimiento en los usuarios. Por lo que, nos enfocamos en simplificar la comunicación dentro de la aplicación, optando por un lenguaje cotidiano para facilitar que los usuarios, especialmente aquellos con poca experiencia técnica, puedan navegar y comprender las funciones del sistema sin esfuerzo.

3. **Dale al usuario el control y la libertad:** hay ocasiones en que los usuarios elegirán las funciones del sistema por error y necesitarán una “salida de emergencia” claramente marcada para dejar el estado no deseado al que accedieron, sin tener que pasar por una serie de pasos. Se deben apoyar las funciones de deshacer y rehacer.

Para poder cumplir con esta heurística buscamos darle al usuario el control y la libertad de realizar las acciones que desee, por eso en cada sistema colocamos tanto sidebars como *headers* para que en todo momento a través de estos el usuario pueda pasar a cualquier parte del sistema de forma instantánea. Además incorporamos *shortcuts* como la previsualización de los pedidos en el inicio para que le sea fácil al usuario navegar a este.

4. **Consistencia y estándares:** *los usuarios no deberían cuestionarse si acciones, situaciones o palabras diferentes significan en realidad la misma cosa; siga las convenciones establecidas [58].*

Realizamos un diseño unificado, manteniendo no sólo el diseño entre los tres sistemas, utilizando una paleta de colores coherente, tipografías uniformes y estilos de botones similares para aquellos usuarios que utilicen tanto la parte del restaurante como la del cliente, lo que asegura que los usuarios, ya sean clientes, dueños de restaurante o administradores, se familiaricen rápidamente con la interfaz, sino que también mantuvimos dentro de cada sistema todas estas características como la forma de los botones, la tipografía entre otras cosas.

Por otro lado, utilizamos la misma terminología en todos los sistemas. Por ejemplo, los Bocados se refieren claramente a los puntos acumulados en el programa de fidelidad en todos los contextos, evitando cualquier confusión sobre qué son y cómo se utilizan a través de los diferentes usuarios del sistema.

5. **Prevención de errores:** *mucho mejor que un buen diseño de mensajes de error es realizar un diseño cuidadoso que prevenga la ocurrencia de problemas [58].*

Para poder cumplir con esta heurística seguimos dos estándares principalmente. En primer lugar realizamos validaciones de entrada en tiempo, esto es, cada vez que se coloca una dirección para buscar restaurantes validamos y se da *feedback* al usuario con una validación en tiempo real para alertar a los usuarios si están cometiendo un error. Por otro lado, implementamos confirmaciones antes de acciones críticas, para cada acción irreversible como puede ser cancelar un pedido o del lado del restaurante cambiar el estado de un pedido implementamos mensajes de advertencia.

6. **Reconocer en lugar de recordar:** *se deben hacer visibles los objetos, acciones y opciones. El usuario no tendría que recordar la información que se le da en una parte del proceso, para seguir adelante. Las instrucciones para el uso del sistema deben estar a la vista o ser fácilmente recuperables cuando sea necesario [58].*

   Para poder cumplir con esta heurística, incorporamos resúmenes de acciones antes de realizar una acción importante como puede ser la confirmación de un pedido. Previo a confirmar un pedido, se muestra un resumen de la acción. Esto permite a los usuarios reconocer y revisar sus elecciones y no tener que recordar todos los detalles de su decisión.

   Por otro lado, como se mencionó en la aplicación otras heurísticas, diseñamos una estructura de menú intuitiva en todos los sistemas, con categorías definidas para que los usuarios, dueños de restaurante y administradores puedan encontrar fácilmente lo que buscan sin tener que recordar dónde se encuentran cada una de las opciones.

7. **Flexibilidad y eficiencia de uso:** *la presencia de aceleradores, que no son vistos por los usuarios novatos, puede ofrecer una interacción más rápida a los usuarios expertos que la que el sistema puede proveer a los usuarios de todo tipo. Se debe permitir que los usuarios adapten el sistema para usos frecuentes [58].*

   Esta heurística es una de las que decidimos aplicar en menos medida ya que al ser una POC no priorizamos la incorporación de atajos, el único atajo solicitado por el cliente es que a solo un click el usuario pueda ver en todo momentos sus Bocados. Por eso, en el header hay un botón especialmente para los Bocados así los usuarios rápidamente se dan cuenta que allí pueden consultarlo a solo un click. Esto fue una petición específica del cliente ya que la utilización de Bocados y aplicación de *marketing* multinivel es lo que los distingue de otros *delivery* de comida por lo que quería que sea notorio.

8. **Estética y diseño minimalista:** *los diálogos no deben contener información que es irrelevante o poco usada. Cada unidad extra de información en un diálogo,*

compite con las unidades de información relevante y disminuye su visibilidad relativa [58].

Con lo que respecta a la interfaz de los tres sistemas, sus diseños y pantallas son minimalistas y toda información mostrada por el sistema tiene un objetivo y valor. Adicionalmente, en los casos de uso consultamos a los usuarios y dueños de restaurantes la opinión respecto al diseño y facilidad de los sistemas y en base a eso fuimos modificando hasta que los resultados finales fueron muy satisfactorios.

9. **Ayuda al usuario a reconocer, diagnosticar y recuperarse:** *diagnosticar y recuperarse de errores: los mensajes de error se deben entregar en un lenguaje claro y simple, indicando en forma precisa el problema y sugerir una solución constructiva al problema [58].*

Para poder cumplir con esta heurística buscamos utilizar un lenguaje no técnico haciendo que todos los mensajes de error están escritos en un lenguaje sencillo, para no confundir a los usuarios. Esto asegura que todos los usuarios, independientemente de su familiaridad con la tecnología, puedan entender el problema y cómo resolverlo.

Además de eso agregamos instrucciones para la resolución de problemas en algunos casos como puede ser la del fallo de un pago. Si un pago no se puede procesar, agregamos mensajes de errores como que verifique si tiene los suficientes Bocados para realizar la compra o en caso de pagar con tarjeta, integramos con mercado pago que se encarga de dar el feedback necesario en caso de que alguna tarjeta falle.

Performance

Performance es la capacidad del sistema de *software* para cumplir requisitos de tiempo. Cuando ocurren eventos, interrupciones, mensajes, solicitudes de usuarios u otros sistemas, o eventos [57]. Para poder cumplir con los requerimientos no funcionales establecidos utilizamos comunicación asincrónica para mejorar los tiempos de respuesta a los servicios. Utilizamos diferentes tipos de comunicación asincrónica como REST, colas de mensajes y la librería Socket.io.

En primer lugar, uno de los requerimientos no funcionales más importantes para la experiencia del usuario es la vista en tiempo real del estatus de las órdenes, tanto para el cliente como para los dueños de los restaurantes. Para poder favorecer este atributo de calidad utilizamos la libreria Sockets.io lo que facilito la comunicación bidireccional entre el cliente y el servidor. Algunas de las tácticas que se aplicaron son la de introducir concurrencia ya que Socket.IO maneja múltiples conexiones a través de eventos de manera asincrónica. Esto permite que el servidor maneje eficientemente múltiples solicitudes de estado en tiempo real sin bloquear el procesamiento de otras tareas importantes.

Por otro lado, también aplicamos REST mediante HTTPS y colas de mensajes. HTTPS garantiza un manejo eficaz de las operaciones asincrónicas. Particularmente las colas de mensajes desacoplan los procesos, equilibrando la carga y mejorando la calidad de respuesta. Adoptamos el patrón CQS entre *TuBocado.MarketBE* y el sistema de puntos *TuBocado.PointsBE* para manejar toda la gestión de los puntos justamente para mejorar la performance; “para mejorar la eficiencia y aumentar la claridad y mantenibilidad del sistema, implementamos una estrategia basada en el patrón *Command Query Separation* (CQS). La implementación de esta estrategia permite minimizar el impacto en la *performance* de *TuBocado.MarketFE* ya que se optó por el uso de colas de mensajes, que permiten manejar las actualizaciones de forma eficiente y asincrónica, reduciendo la carga de procesamiento y mejorando el manejo de la concurrencia a la horoa de generar los puntos para un usuario. Esto asegura que los procesos no interrumpan las operaciones principales y que los pedidos se manejan sin demoras.” Una de las tácticas aplicadas en este caso es el manejo de ratio de eventos ya que la cola actúa como amortiguadora para las ráfagas de tráfico, gestionando y equilibrando la carga lo que asegura un procesamiento uniforme sin sobrecargar el sistema. Haciendo que al crear órdenes se deba procesar menos datos de forma sincrónica lo que hace posible el requerimiento US-14.

Por último, en la sección de tecnologías también hablamos sobre la elección de ReactJs para desarrollar los tres *frontends*. Uno de los atributos a tener en cuenta en la tabla comparativa de la sección de tecnología fue la performance ya que ReactJs mejora la performance mediante su virtual DOM, que minimiza las

actualizaciones en el DOM real, reduciendo así el tiempo de renderizado y mejorando la experiencia de usuario.

Interoperabilidad

La interoperabilidad se refiere al grado en que dos o más sistemas pueden intercambiar información significativa a través de interfaces en un contexto específico. Esto no solo implica la capacidad de intercambiar datos (interoperabilidad sintáctica), sino también la habilidad para interpretar correctamente los datos intercambiados (interoperabilidad semántica) [57].

Para poder lograr una buena interoperabilidad entre los servicios, lo primero decidimos fue la utilización del estándar REST. Esta decisión la tomamos en base a la comunicación simple que provee el estándar REST, que además es un protocolo de comunicación ampliamente aceptado y entendido. Esto facilita la interoperabilidad con una amplia gama de tecnologías y minimiza las barreras de entrada para la integración de nuevos componentes o sistemas externos.

Por otro lado, para interconexión entre los servicios también utilizamos colas de mensajes. Las colas de mensajes permiten una comunicación asincrónica entre servicios, lo cual es bueno para operaciones que no necesitan de una respuesta inmediata. Esta flexibilidad mejora la interoperabilidad al permitir que servicios con diferentes requisitos de tiempo de respuesta y cargas de trabajo interactúen sin conflictos.

Deployabilidad

La *deployabilidad* se refiere a la facilidad con la que un sistema de *software* puede ser desplegado, actualizado o distribuido en un entorno de producción. Este atributo de calidad es crucial para evaluar qué tan rápido y eficientemente se pueden realizar cambios en el *software* y ponerlos a disposición de los usuarios finales.

Este es uno de los atributos de calidad que surgieron en base a algunos requerimientos no funcionales propuestos por todo el equipo. El cliente en el pasado nunca había trabajado con despliegue continuo y contenedores. Al desplegar cada servicio en *Elastic Beanstalk* con Docker, las bases de datos en RDS (*Amazon*

Relational Database Service), y los *frontends* en Amazon *Amplify*, conectando todo con GitHub, aplicamos varios elementos que favorecen la *deployabilidad*.

En primer lugar buscamos la automatización de la configuración del despliegue junto con la aplicación de integración y entrega continua. Al utilizar *Elastic Beanstalk* junto con *Docker* y conectar todo el sistema con GitHub facilitamos la implementación de *pipelines* de CI/CD, permitiendo la automatización de pruebas y despliegues automáticos. Con esto aseguramos que las actualizaciones puedan ser desplegadas rápidamente y con confianza, brindando mayor agilidad y la capacidad de respuesta a cambios.

Luego, tuvimos en cuenta la separación de configuración del código, esto es, separar datos importantes como puede ser la conexión a base de datos del código en sí. Para lograrlo, almacenamos la configuración del entorno de manera separada al código gestionarlas a través de *Elastic Beanstalk* lo que mejora la seguridad y facilidad de adaptar el sistema a diferentes entornos como puede ser producción o *local*.

Por último, utilizamos contenedores para encapsular los servicios. Docker proporciona un entorno de ejecución consistente para cada servicio, independientemente del entorno de despliegue. Esto mejora la portabilidad y reduce los problemas relacionados con la frase "en mi máquina sí funciona", facilitando así el despliegue.

En la sección de guía de *deployment* en el anexo se encuentra una guía detallada de los pasos que hay que seguir para poder desplegar todo el sistema.

5. Proceso Gestión

5.1. Introducción

Dentro de lo que es el cronograma de trabajo, realizamos una división de las actividades en dos grandes grupos, a los cuales denominamos *Discovery* y *Delivery*.

Discovery fue el conjunto de actividades con las que buscábamos reducir la incertidumbre sobre los requerimientos del producto, sobre el uso de las metodologías y sobre el uso de las tecnologías y herramientas. Una vez reducida dicha incertidumbre nos encontrábamos en condiciones de adentrarnos en las actividades de *Delivery*, actividades como construir el producto, aplicar las metodologías y usar las tecnologías y herramientas investigadas. Realizábamos reuniones de retrospectiva donde evaluábamos la incertidumbre. Una vez que detectamos que se redujo, comenzamos con la etapa de *Delivery*.

A continuación, en la ilustración 5.1.1 podemos encontrar un eje de tiempo que representa este cronograma primario. Cabe destacar que la fecha 31/07/2023 es simbólica, ya que la transición de *Discovery* a *Delivery* fue más bien un proceso y no un punto en el tiempo. Una vez comenzadas las actividades de *Delivery*, continuamos con otras actividades de *Discovery* en simultáneo.



Ilustración 5.1.1: Cronograma de trabajo

5.2. Discovery

En la etapa de *Discovery*, tuvimos como objetivo reducir la incertidumbre e ir respondiendo incógnitas, para luego poder pasar a *Delivery*. Esta etapa constó de una serie de actividades como la definición de requerimientos del producto, la selección de las metodologías y la selección de las herramientas y tecnologías a usar en *Delivery.* Para la definición de requerimientos, como metodología de trabajo optamos por *Design Thinking*. Para ordenar nuestras actividades diarias usamos

Kanban, facilitando así la organización del trabajo. Por último, semanalmente llevábamos a cabo reuniones de retrospectiva para analizar el avance de las tareas y definir que podíamos cambiar, mejorar y a que áreas dedicarle más o menos tiempo según la experiencia de la última semana de trabajo. Además, en estas retros medíamos y evaluamos el nivel de incertidumbre que teníamos hasta el momento.

Cabe destacar que dicha medición de la incertidumbre no fue necesariamente cuantitativa, pero buscamos medios para hacerla lo más objetiva posible. Un ejemplo de una forma de medir fue lo realizado respecto del lenguaje de *frontend* a utilizar. En un comienzo evaluamos diferentes opciones, para las cuales realizamos una Matriz de Pugh, lo que nos permitió decidirnos por una herramienta, lo cual coadyuvó a bajar la incertidumbre. Luego comenzamos a probarla mediante *Spikes* y corroboramos que habíamos tomado una buena decisión, lo cual fue un segundo paso de dicha reducción de la incertidumbre. Podemos encontrar más información sobre dichos *Spikes* en la sección 5.8 Capacitaciones.

Utilizamos desde el comienzo un tablero Kanban en la herramienta *Click Up*, como se muestra en la Ilustración 1.2. Este tablero contaba con las 4 columnas *Open, In progress, Blocked* y *Closed* donde creábamos tareas a realizarse en la etapa de *Discovery*. Dichas tareas recorrían cada una de las columnas hasta llegar a Closed. La columna *Blocked* fue agregada para aquellas tareas que fueron creadas, estuvieron en *In progress* pero por algún motivo externo o interno a nosotros, decidimos no continuar con la misma. Este tablero nos permitió determinar el estado del trabajo, el progreso, medir el esfuerzo invertido en cada una de las tareas y nos organizó como equipo. Este último punto nos pareció sumamente importante dado que este tablero fue creado al comienzo de todo este proyecto, donde aún estábamos en un periodo de adaptación y de pruebas para lograr encontrar la mejor forma de trabajo para todos y en conjunto.

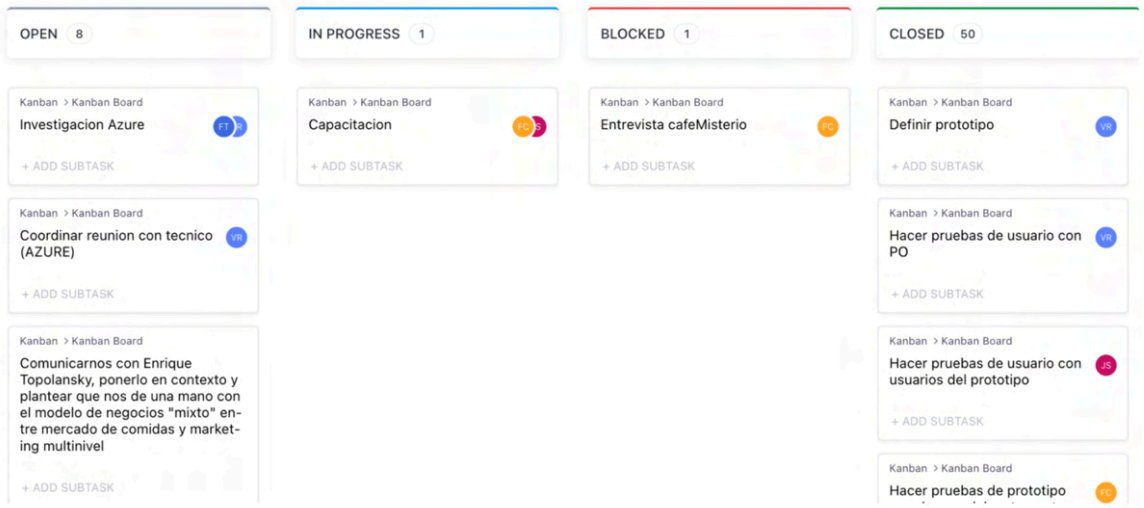


Ilustración 5.2.1: Tablero Kanban para tareas de Discovery

A pesar de que en la ilustración 1.1 vemos que hay una fecha de “fin” para la etapa de *Discovery*, esto no fue exactamente así. Estas dos etapas podemos decir que no son excluyentes entre sí, no fue de un día para el otro que dimos por finalizada una etapa y comenzamos otra, sino que tuvimos un periodo de tiempo en que trabajamos en las dos en simultáneo. Durante el desarrollo del sistema, seguimos manteniendo tareas de descubrimiento, análisis y pruebas que nos ayudaron a reducir la incertidumbre en el desarrollo.

5.3. Selección de metodología

Para la selección de metodologías, nuestro punto de partida fue el conocimiento y la experiencia que poseíamos como equipo en lo que respecta a metodologías ágiles. A eso le sumamos el análisis en profundidad de las características de nuestro proyecto, del cliente y del equipo. Luego de esto, pudimos probar y comparar las opciones a modo de reducir así la incertidumbre, hasta llegar a conclusiones y decisiones finales, con la certeza de que eran las acertadas.

Como primer paso para adoptar y adaptar la metodología más acertada, analizamos las características principales del proyecto, del cliente y del equipo:

- **Relación con el cliente**: con el cliente tuvimos la posibilidad de comunicarnos directamente y con frecuencia. Además, desde el comienzo nos solicitó realizar entregas regularmente de los avances que fuéramos

teniendo para mantenerlo actualizado a modo de que se reduzcan los costos de posibles errores o discrepancias.

- **Fecha de finalización inamovible**: al ser un proyecto de tipo con la industria, la fecha de finalización está estipulada ya y no se puede cambiar, realizamos un plan de gestión de riesgos para el caso en que no lleguemos a la entrega, donde inevitablemente el alcance del proyecto se vería comprometido, pero la fecha de entrega no era un cambio posible.
- **Uso de tecnologías específicas:** por su experiencia y utilización de algunas tecnologías, el cliente nos solicitó si podíamos usar alguna como lo es Azure para alojar la aplicación web, por lo que realizamos la investigación necesaria para poder cumplir con esta solicitud. Esto necesitaba de una metodología que nos permitiera aprender mientras trabajabamos dado que ninguno de los integrantes tenía experiencia con la misma.
- **Requerimientos inestables:** este proyecto surgió para que el cliente pueda validar el modelo de negocio planteado con nuestro POC, por eso los requerimientos fueron cambiando.
- **El proyecto es una prueba de concepto (POC):** en relación al punto anterior, este proyecto surgió como una prueba de concepto, con todas las características que esto conllevaba que mencionaremos más adelante.
- **Horarios de trabajo flexibles y disjuntos:** si bien tenemos algunos horarios fijos semanales para trabajar en el proyecto, estos fueron variando según la disponibilidad de cada integrante. Dicha disponibilidad estaba atada a que los 4 integrantes trabajamos 6 y 8 horas, así como también algunos siguen cursando materias de la carrera.
- **El equipo de trabajo es de 4 integrantes:** esta característica también era parcialmente fija, tuvimos que planear la metodología teniendo en cuenta que el número de integrantes no iba a variar, independientemente de que algún integrante pudiera darse de baja.
- **Relacionamiento del equipo:** buen relacionamiento y comunicación diaria del equipo. Todos los integrantes ya nos conocíamos, teníamos buena relación, habíamos realizado trabajos previos juntos y teníamos una buena dinámica de trabajo lo cual era muy favorable para el proyecto.

En base a estas características mencionadas, llevamos a cabo una

investigación para determinar cuál era la metodología de trabajo más adecuada para cada etapa de nuestro proyecto. Para ello, decidimos realizar una matriz de Pugh que podemos ver en detalle en el Anexo A.9. Matrices de Pugh - Metodología.

Como ya mencionamos, para la etapa de *Discovery* decidimos utilizar Kanban. Tomamos la decisión en base a que creímos que fue la más acertada ya que es una metodología ágil, flexible y centrada en la optimización del flujo de trabajo. Además, como podemos ver en la ilustración 5.3.1, donde se muestran las principales diferencias entre Kanban y Scrum, Kanban es muy útil, principalmente para estar al tanto de las tareas de todo el equipo. Semanalmente teníamos un espacio en el cual hacíamos retrospectivas, las cuales utilizamos para reflexionar sobre el trabajo realizado y buscar mejoras en el proceso. Estas nos permitieron analizar lo que venía funcionado bien, identificar problemas y proponer soluciones para optimizar el flujo de trabajo dado que usamos el método DAKI que será explicado más adelante en la sección 5.10. Ejecución. Estas instancias promovían la colaboración y la transparencia, y fomentan la búsqueda constante de mejoras en el proceso de trabajo.

En esta instancia, decidimos no empezar con *Sprints* porque debíamos tener paquetes de tareas completadas y además implicaba desarrollar y generar ejecutables, lo cual aún no era posible. Decidimos que era conveniente utilizar Kanban y no poner más de una semana de duración a las tareas, de ese modo, haciendo las retro cada semana, no era necesario armar un *Sprint*. Kanban exige retrospectivas una vez por semana para mantener cierto ritmo pero no te exige *reviews* entonces en este periodo decidimos mantenernos en esta metodología.

Kanban nos resultó ideal para la primera etapa del proyecto porque nos brindó flexibilidad y mediante el tablero nos permitió una gestión visual del trabajo. Pudimos visualizar y priorizar eficientemente las tareas que tenían que ver con investigación y el descubrimiento sin tener que marcar plazos fijos y estrictos o iteraciones predefinidas. Kanban es una metodología muy relacionada al flujo continuo de solicitudes ya que se basa en entrega y desarrollo continuo, nos fue útil para reducir la complejidad en el área de gestión de esta etapa.

Teniendo lo mencionado en cuenta, comenzamos eligiendo qué metodologías

queríamos comparar, para ello tuvimos en cuenta que nos sentíamos cómodos utilizando metodologías ágiles dada la experiencia laboral y académica que tenemos con estas. Además, creemos que El Manifiesto Ágil [12] establece una serie de valores que encajan con nuestro equipo. Entre estos valores, destacamos aspectos como la retroalimentación, la mejora continua y, especialmente, la transparencia. Es por esto que la investigación de metodologías para el proceso de gestión en la segunda etapa, nuevamente tenía nuestro foco sobre Scrum y Kanban.

Como siguiente paso, decidimos cuáles eran los atributos que nos interesaban analizar de estas metodologías y qué prioridad le dábamos a cada uno de estos. La prioridad la clasificamos en tres categorías: 1 baja, 2 media y 3 alta.

A raíz de esto realizamos la matriz de Pugh correspondiente que nos permitió comparar para decidir la metodología que más se adecuaba a esta etapa del proyecto y al equipo. En conjunto a esta matriz, para ahondar aún más en ambas metodologías y los atributos seleccionados, decidimos ampliar la investigación y análisis creando una tabla comparativa, como resultado obtuvimos la tabla 5.3.1.

Atributo	Kanban	Scrum
Entregas en tiempo fijo	No tiene tiempos fijos para entregas, entrega continua de trabajo.	Se basa en entregas fijas al final de cada *Sprint*.
Fecha de finalización inmovil	Las fechas de finalización se pueden ajustar según necesidades.	Fechas inamovibles de los *Sprints*, aunque se pueden ajustar entre *Sprints*.
Baja burocracia	Tiende a tener menos burocracia dado que está centrado en la gestión visual y la optimización del flujo de trabajo.	Si bien promueve la autoorganización, implica más burocracia por sus roles y ceremonias
Nivel de control	Es más flexible, se adapta mejor a situaciones cambiantes y permite ajustes en tiempo real.	Es más estructurado, tiene roles y eventos definidos que brindan claridad sobre quién hace qué y cuándo.
Experiencia del equipo	Los 4 integrantes conocemos por facultad pero nunca lo hemos usado en la práctica.	Los 4 integrantes lo usamos a diario en nuestros trabajos.
Fácil de implementar	Relativamente fácil de implementar debido a su enfoque simple y flexible. Puede adaptarse según necesidad.	Puede requerir más esfuerzo inicial para implementar correctamente, ya que implica establecer roles, eventos y artefactos definidos.

Tabla 5.3.1: Tabla comparativa entre Kanban y Scrum

Por lo tanto, nuestra selección de la metodología comenzó con la identificación de las necesidades clave del proyecto, seguido por la evaluación de las dos opciones utilizando la matriz y la tabla comparativa como guía y por último un análisis de todos los datos obtenidos para tomar la decisión. Este enfoque nos permitió comparar objetivamente las metodologías entre sí. A continuación, planteamos cuál fue nuestra decisión junto con una breve descripción de por qué tomamos dicha decisión.

Al investigar Scrum, nos dimos cuenta que nos proporcionaría una estructura más rígida en cuanto a las entregas y los plazos, lo que creímos muy útil dado que teníamos un conjunto bien definido de funcionalidades a desarrollar y un plazo fijo inamovible de entrega. Scrum cuenta con *Sprints* que nos permiten como equipo poder establecer ciertos objetivos claros en un plazo corto y fijo. Además también permite poder trabajar de manera iterativa y entregar valor incremental en periodos de tiempo definidos al cliente. Scrum le llama *User Story* a los requerimientos y a su vez utiliza una subdivisión de una user story en tareas, lo que facilita a visualizar el trabajo. Esta metodología nos permitía agregar y entregar valor al cliente al finalizar cada iteración acordada, esto nos parecía muy útil para poder mostrar nuestro avance y la revisión del proyecto en periodos cortos de tiempo, además de poder desplegar a ambientes de producción sin tener que esperar al final.

En relación a la rigidez, Scrum plantea que si surgen cambios en el transcurso de un *Sprint*, esos no se realizan en el *Sprint* actual, sino que son postergados para el siguiente. En cuanto a Kanban, esta plantea una postura un poco más flexible, ya que es posible agregar cambios que surjan en cualquier momento del transcurso del desarrollo. Dado que una de las características del proyecto era que tenía requerimientos inestables, nos parecía un poco más práctica la postura de Kanban, aunque la de Scrum creímos que podía facilitarnos la organización como equipo y para tener un seguimiento de lo planteado originalmente y de los cambios requeridos.

Así mismo, otro factor importante a tener en cuenta es que Scrum, a diferencia de Kanban, cuenta con una distribución formal de roles: *product owner*, *scrum master* y desarrolladores. Esto fue un punto de interés para nosotros dado que siendo 4 integrantes, nos resultaba conveniente contar con roles definidos y

responsabilidades claras para poder trabajar de forma más estructurada y organizada. Además de estar acostumbrados a esa modalidad de trabajo por nuestros respectivos trabajos.

Por último, en cuanto a las reuniones que plantean ambas metodologías, Kanban cuenta con las reuniones diarias llamadas *stand-up* que se podrían comparar con las dailies que presenta Scrum pero además de estas, esta última metodología cuenta con lo que denominan ceremonias que son reuniones que se llevan a cabo en momentos claves de la iteración. Estas ceremonias creímos que serían muy útiles para nosotros dado que al ser 4 integrantes y cada uno trabajando y estudiando con sus disponibilidades horarias probablemente disjuntas, era muy posible que se nos dificulte coordinar una hora todos los días para las stand-up, sin embargo, poder tener las ceremonias por ejemplo los fines de semana unicamente o un dia puntual de la semana que todos pudiéramos nos iba a favorecer.

En base a toda la información obtenida y presentada, nuestra decisión fue utilizar una variante personalizada de Scrum. En nuestro enfoque mantuvimos todas las ceremonias del marco mencionado, pero con la particularidad de ser flexibles con la frecuencia y estructura de las reuniones. Teníamos fijos los días de las reuniones pero no las horas, sobre todo por un tema de estudio y trabajo lo que nos resultaba más práctico era fijar al comienzo de la semana las reuniones internas y después en caso de tener que realizar algún ajuste lo comentábamos con todo el resto del equipo. Además, dada la disponibilidad del *product owner*, si bien mantuvimos constante comunicación con el mismo, definimos que el participara únicamente de las *Release Plannings* y *Release Reviews* y no necesariamente de las *Sprint Plannings* y *Sprint Reviews*. Por último, para el proceso de desarrollo investigamos TDD y BDD, *Test driven development* y *Behavior driven development* respectivamente y terminamos decidiendo usar BDD, o algunas de sus prácticas.

5.4. Definición de requerimientos

Para definir los requerimientos del sistema, utilizamos la metodología *Design Thinking* seguido de una serie de pasos que se detallan más adelante. Esta es una metodología iterativa que se centra en entender las necesidades reales de los usuarios para poder así diseñar soluciones innovadoras con un importante respaldo

en la prototipación y prueba con dichos usuarios. Si bien seguimos todos los pasos, cabe aclarar que el cliente siempre tuvo la última palabra en la definición de todos los aspectos.

El proceso de *Design Thinking* cuenta de cinco etapas conectadas entre sí que nos ayudaron a definir el problema. Estas etapas las podemos ver en la ilustración 5.4.1.

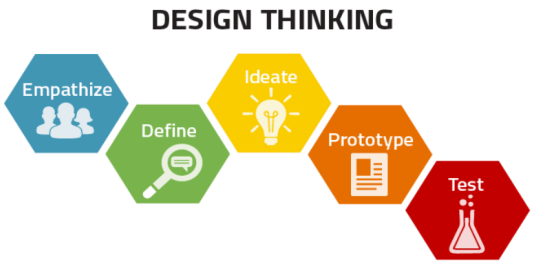


Ilustración 5.4.1: ciclo de design thinking

Empatía

La empatía es la primera etapa en el proceso de *Design Thinking.* En esta etapa obtuvimos la información a través de entrevistas, inmersiones y encuestas con ciertas personas que creímos útiles para comenzar con este proceso. Esta etapa nos ayudó a comprender y compartir sentimientos, pensamientos y perspectivas de los usuarios para poder llegar a la solución adecuada.

Entrevistas

Para comenzar realizamos tres entrevistas a dueños de restaurantes de sushi en Montevideo, quienes nos compartieron sus experiencias vendiendo sushi en las diferentes plataformas de entrega de comida existentes hoy en día en el país. Todas las entrevistas nos fueron útiles, pero hubo una en particular que fue muy interesante, ya que se trataba del dueño de un restaurante que comenzó a vender sushi en una plataforma en los inicios de la misma, por lo que supo transmitirnos con lujo de detalle ciertas cosas que si o si deberíamos tener en cuenta a la hora de empezar y otras que le parecieron secundarias. Para las entrevistas utilizamos un guión establecido por nosotros mismos que reflejaba las necesidades del cliente.

Además, realizamos una entrevista para conocer un poco más del *marketing* multinivel, más allá del sector de comida rápida. El PO fue quien nos introdujo el tema pero creímos necesario ahondar un poco más. Para ello, nos contactamos con la encargada de ventas de las ollas Essen, ya que investigando supimos que utilizan el *marketing* multinivel para vender sus productos. Esta entrevista fue útil para entender el funcionamiento, dado que ninguno tenía experiencia en el tema.

Para finalizar con las entrevistas, quien nos brindó más de una fue el PO que nos expresó su visión del problema y cómo se imagina él y su equipo la solución, así como también nos explicó en profundidad el concepto de Marketing Multinivel y como aplicarlo ya que ninguno de nosotros tenía experiencia en el tema y era crucial para este proyecto. En esta etapa realizamos también dos inmersiones, en la primera realizamos *benchmarking* de otra herramienta, accediendo a una plataforma de venta de comida desde el perfil del dueño de un restaurante para así poder revisar en profundidad las funcionalidades de la misma.

Por otro lado, en la segunda inmersión hicimos uso de dos plataformas existentes, poniéndonos en el lugar de usuarios para poder seguir el paso a paso de la compra. Este proceso lo repetimos reiteradas veces en ambas plataformas para poder probar todas las configuraciones, los medios de pago, los envíos, y demás. Nuevamente fue muy útil tanto para ver algunas cuestiones del diseño y estructuración como de detalles sobre funcionalidades clave que quizá no habíamos analizado hasta el momento.

Por último, para seguir reduciendo la incertidumbre acerca de los requisitos del proyecto, realizamos una encuesta a actuales consumidores de sushi, por ende, posibles consumidores de nuestra plataforma, para poder conseguir más información acerca de este nicho y las funcionalidades que les gustaría ver en nuestra solución. Tuvimos una gran cantidad de respuestas en el cuestionario, 37 personas jóvenes de entre 20 y 30 años nos dieron su opinión acerca del consumo de sushi hoy en día y la visión que tienen acerca de nuestro proyecto.

Podemos ver las conclusiones a las que llegamos en cada instancia a continuación, y para ver las entrevistas, inmersiones y sus respuestas obtenidas ver

Anexo A.10. Entrevistas e inmersiones y resultados obtenidos. En las siguientes imágenes podemos ver a nuestro equipo llevando a cabo entrevistas.



Ilustración 5.4.2: analizando estadísticas restaurante de sushi

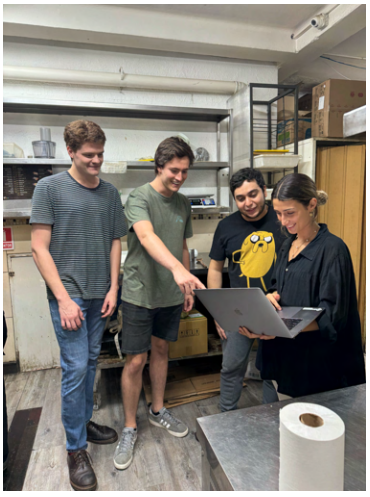


Ilustración 5.4.3: con el gerente del restaurante

Entrevista dueño de Restaurante 1

Esta entrevista nos brindó la oportunidad de presenciar cómo eran otras plataformas de venta de comida en el periodo en el que estábamos nosotros: al comienzo del proyecto, y compararlo con lo que conocemos en la actualidad de la aplicación. En este periodo la comunicación entre los locales y la aplicación era mayoritaria por mail, era difícil establecer comunicación fluida. Pudimos concluir que la mayoría de los problemas que se tenían era debido a la falta de comunicación, sin embargo la aplicación significa un gran aumento de las ventas para el local. El dueño consideraba difícil lidiar con la plataforma pero necesario ya que le aportaba muchas ventas.

Entrevista dueño de Restaurante 2

Pudimos observar que uno de los principales problemas que existen entre los restaurantes y las aplicaciones de *delivery* de comida es el costo que esto implica. El entrevistado señala múltiples veces que su mayor problema es la comisión grande que plataformas existentes hoy en día se quedan, las cuales andan alrededor de un 23%. Uno de los principales objetivos de la aplicación TuBocado es poder optimizar los costos para poder ofrecerle a los dueños de los restaurantes un incentivo a unirse.

Por otro lado, se puede observar como al día de hoy, las plataformas que usa decidieron quitar la opción de *delivery* propio lo que hace que si o si los restaurantes están obligados a pagar un porcentaje mayor de las ventas sin tener la opción de usar el *delivery* propio que usualmente utilizan. Por lo que estaría bueno que TuBocado pueda brindar flexibilidad en el uso de *deliverys* propios. Esto permitiría a los restaurantes tener un mayor control sobre el proceso de entrega y la calidad del servicio además de que permitirá reducir los costos.

El entrevistado también comentó que el 50% de sus ventas son mediante llamadas telefónicas por lo que no presentaría un problema solicitar a los restaurantes que utilicen su propio sistema de *delivery*, sino que al revés, daría la opción de reducir aún más los costos de repartición de comida mediante la aplicación. Además esto hace que si no se utiliza el *delivery* no tengan que pagar “extra” si un usuario selecciona la opción de “retirar por el local”, ya que hoy en día las aplicaciones al tener integrado el *delivery*, cobran lo mismo por llevar el pedido que si un cliente retira por el local el sushi.

Con lo que respecta a las demoras de entrega, el entrevistado indica que la penalización hace que se pierda un porcentaje muy importante de ventas. Esto es porque generalmente las penalizaciones se dan cuando hay picos de demanda por lo que cuando los penalizan es cuando hay más demanda y se pierden más clientes. Alguna de las cosas que se pueden considerar es, en primer lugar establecer tiempos de entrega realistas y comunicarnos de manera clara a los clientes al realizar los pedidos. También buscar estrategias para hacer frente a las demandas pico, esto puede ser ofrecer reportes y estadísticas de horas pico para sugerir incluir la contratación adicional de personal de entrega, entre otras cosas.

Otro punto es la importancia de la calificación, no solo para los clientes sino también para los restaurantes. Cuando se habló con el *product owner* de TuBocado indicó que no eran necesarias las calificaciones de un restaurante para la versión inicial. Luego de esta entrevista se pudo entender que no solo era importante para los consumidores sino también para los dueños de los restaurantes ya que les da una noción de cómo le está yendo en los casos que no puedan asistir regularmente y vigilar de cerca los pedidos.

Por último, existe una alta demanda de sushi los fines de semana y una menor demanda durante la semana, se puede concluir que existe una estacionalidad que puede afectar la rentabilidad y eficiencia de los restaurantes y servicios de entrega. Para abordar este problema se le planteó al dueño la opción de programas de fidelización basados en puntos, es decir, una manera de fomentar el consumo de sushi durante la semana implementando un sistema de puntos. Esto le pareció una muy buena idea y dijo que estaba interesado.

En conclusión, mejorar las aplicaciones de entrega convencionales de sushi implica abordar los puntos negativos existentes, como las altas comisiones de pago, la restricción de utilizar *deliverys* propios, la falta de diferenciación en los precios para recoger en el local y las penalizaciones por demoras. Además, es fundamental tener en cuenta la estacionalidad en la demanda de sushi. Para abordar este desafío, programas de fidelización basados en puntos sería el diferencial de la aplicación, que no solo permite reducir los costos e incentivar a los usuarios a comprar siempre sushi, sino que genera viralidad en el consumo lo que hace que no se necesite “marketing convencional” para la venta de los productos.

Entrevista dueño de Restaurante 3

Esta investigación nos brindó una visión detallada de los restaurantes los cuales tienen pocas ventas a través de plataformas de pedidos, especialmente en el contexto de restaurantes de sushi. Esto significó, que no pudimos obtener gran feedback sobre su experiencia con las mismas, pero sin embargo, nos dio información sobre las razones por las cuales están en esa situación. El entrevistado comentó que la razón principal es la gran comisión de las opciones que manejan actualmente, dejándolos casi con nulo margen para las ganancias, además los pedidos que les llegan por este tipo de aplicación son pocos.

Entrevista vendedora de Ollas Essen

Mediante esta entrevista, notamos que era fundamental contar con un sistema de registro y seguimiento de ventas para acumular los puntos por cada transacción, si bien la manera de acumular no es la misma, se debe mostrar al usuario claramente los puntos y como fue acumulandolos para poder canjearlos.

Otro punto relevante era el de objetivos mensuales y las iniciativas a los usuarios a aumentar sus ventas y obtener recompensas adicionales, si bien el sistemas de recompensa o incentivos mensuales no llegó a ser implementado, fue importante visualizar que existía para pensar una arquitectura y diseño para que esto sea posible en el futuro de forma sencilla.

Por último, nos parece clave para desarrollar la aplicación tener en cuenta toda la logica de la acumulacion de puntos para el éxito del *marketing* multinivel, además fue importante observar cómo era su sistema de pagos y comisiones lo cual garantiza que los puntos sean transferidos correctamente.

Entrevistas *Product Owner*

Las entrevistas con el *product owner* al inicio del proyecto fueron un paso crucial en esta etapa. De estas varias entrevistas fuimos incorporando aún más el concepto de *marketing* multinivel, logrando profundizar nuestro entendimiento y comprendiendo un poco más su funcionamiento. Así también nos sirvieron para entender qué es lo que el cliente estaba buscando de este proyecto, las necesidades y expectativas y por donde pensaba encaminar la solución.

Cabe destacar que fueron muy útiles también para conocer más al cliente y poder ir generando un vínculo y una confianza muy importantes para poder llevar a cabo todo el proyecto. Este vínculo fue fundamental para el éxito del proyecto, ya que nos ayudó en la colaboración efectiva y en obtener una comprensión compartida de desafíos y oportunidades que enfrentamos. Además, nos permite iterar y refinarnos continuamente en nuestras ideas y propuestas, asegurándonos de que la solución final no solo fuera innovadora y técnicamente sólida, sino también alineada con la visión estratégica y los objetivos de la empresa.

Por último, estas entrevistas fueron clave para llegar al diseño de una solución innovadora y centrada en el usuario que realmente agregan valor al proyecto de *marketing* multinivel. Al comprender en profundidad las necesidades y objetivos del cliente, así como las expectativas del mercado y las tendencias del sector, pudimos diseñar una solución que no solo abordará los requisitos específicos del proyecto, sino que también agregará un valor significativo para los usuarios finales. Gracias a la información que obtuvimos, pudimos identificar oportunidades

de mejora en cuanto a lo que ya hay hoy en día en el mercado y las áreas clave en las que enfocar nuestros esfuerzos de diseño. Además, al tener en cuenta todo lo hablado y con el conocimiento ya obtenido por otras entrevistas, logramos en conjunto integrar de forma eficaz los requisitos del negocio con las necesidades del usuario, creando una solución que equilibraba la viabilidad comercial con la experiencia del usuario.

Inmersión - *Benchmarking* de plataforma de venta de comida

Hicimos uso de esta inmersión para poder adoptar ciertas características del diseño y estructura de la información para favorecer a la usabilidad, como por ejemplo la barra navegadora del lado izquierdo como se ve en la ilustración 5.4.4.

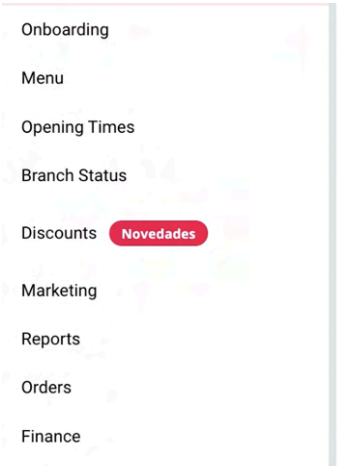


Ilustración 5.4.4: Barra de navegación de aplicación de venta de comida 1

Usamos además mucha información acerca del dashboard para priorizar qué datos íbamos a mostrar y de qué forma estos se iban a ver. Podemos verlo en la ilustración 5.4.5.

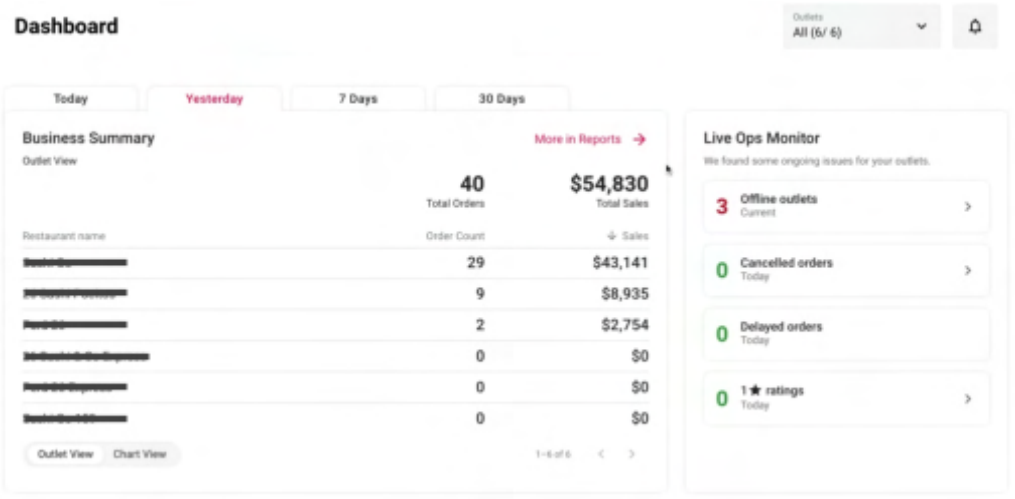


Ilustración 5.4.5: Dashboard de aplicación de ventas de comida 1

Ver Anexo A.11. Benchmarking aplicación de ventas de comida 1 para tener en análisis completo realizado.

Uso de una plataforma nosotros mismos

Una de las prácticas que aplicamos fue la de nosotros mismos intentar el flujo de pedir sushi mediante aplicaciones conocidas, si bien frecuentemente pedimos comida en aplicaciones, nunca prestamos verdaderamente atención a los flujos y experiencia de usuario de los mismos. Al analizar la experiencia de usuario pidiendo sushi mediante una aplicación, pudimos ver la importancia de una interfaz fácil de usar y navegar. La simplicidad en la estructura de la aplicación nos parece fundamental para garantizar una experiencia satisfactoria para el usuario. Además, pudimos ver la importancia de la rapidez en el proceso de las órdenes para el usuario. Los usuarios valoran la eficiencia al realizar sus pedidos, por lo que optimizar este proceso nos pareció crucial y fue una de las cosas que anotamos a tener en cuenta en nuestros requerimientos no funcionales. Por último, notamos la importancia de hacer énfasis en las reseñas, para el usuario es muy importante tener la opinión y experiencia de otros a la hora de elegir un restaurante para comer.

Encuestas

Otra de las actividades que realizamos fueron encuestas a usuarios tipo. Para realizar las preguntas de las encuestas, nos basamos en algunas de las problemáticas identificadas en las entrevistas. Decidimos realizar preguntas para

poder entender hacia dónde va el mercado, en que se centran los usuarios al comprar comida y que podría hacer que ellos compraran más.

En primer lugar, podemos ver que el consumo de sushi es primordialmente de forma ocasional, no se acostumbra a pedir todas las semanas aunque si varias al mes u “ocasionalmente” que fue la respuesta que más se votó. Esto, en conjunto con algunas respuestas siguientes que serán comentadas nos parece algo a explotar ya que podemos fomentar el consumo y lograr que el número de votos semanal o mensual crezca. Esto lo usamos como una validación para la utilización de puntos (puntos) en el sistema, ya que al existir este beneficio para los usuarios, podríamos fomentar el consumo más días a la semana.

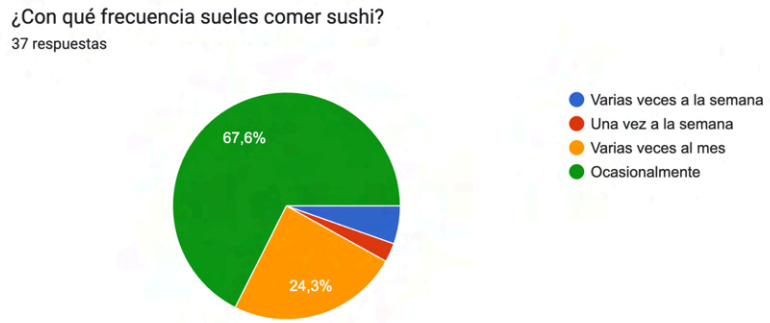


Ilustración 5.4.6: Resultados pregunta 1 encuesta

En cuanto a los días de la semana, está claro que gana los fines de semana, lo cual se ratifica con lo que estuvimos analizando de otras entrevistas. Como dijimos en la pregunta pasada, este también es un punto de enfoque, manteniendo las ventas los fines de semanas y aumentando la venta entre semana, es un camino para aumentar las ventas, que es uno de los principales objetivos del proyecto.

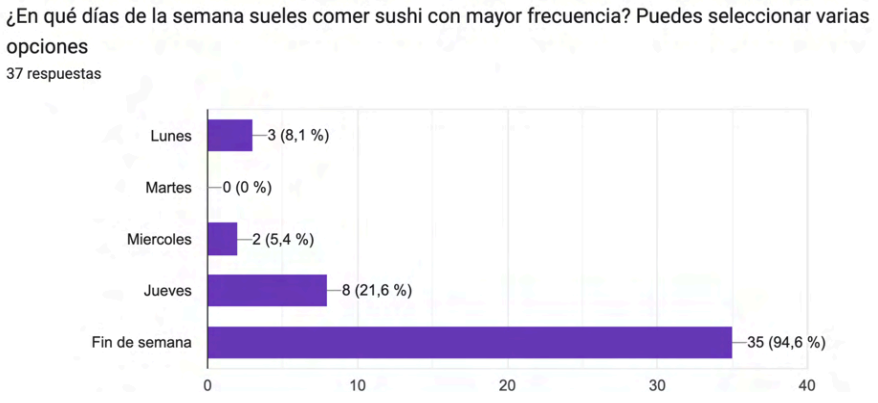


Ilustración 5.4.7: Resultados pregunta 2 encuesta

También, podemos ver que la mayoría compra sushi por una aplicación y no en el local físico, lo cual nos indica que estamos en lo correcto al pensar que una aplicación para pedir sushi es acertada y útil para el público al que apuntamos. Hoy en día la gente acostumbra a pedir y no ir al local, por esto un enfoque importante del sistema tiene que ser el envío a domicilio.

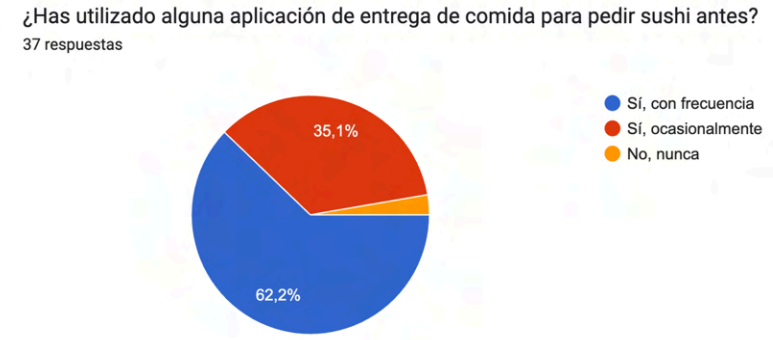


Ilustración 5.4.8: Resultados pregunta 3 encuesta

A modo de introducir vagamente las características de nuestro sistema, preguntamos cuáles son las características que les gustaría que nuestra aplicación tenga, dentro de estas opciones introdujimos el concepto de puntos y promociones y descuentos exclusivos y según los resultados obtenidos les pareció una buena idea. En general al consumidor le suena atractivo comprar si es con descuento y eso se ve reflejado en las respuestas. Uno de los principales factores de por que el consumo de sushi no es semanal ni a veces mensual es por el hecho de que es un producto costoso reduciendo la regularidad con la que se pide.

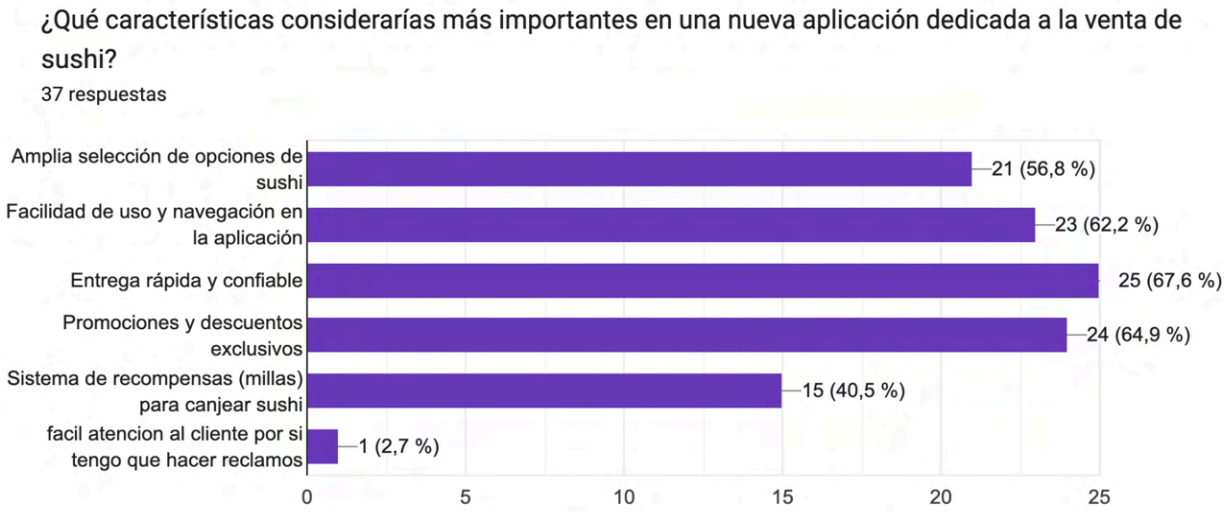


Ilustración 5.4.9: Resultados pregunta 4 encuesta

Para continuar indagando en el uso de puntos para la compra de comida, quisimos asegurarnos de que los clientes tuvieran un motivo real para sentirse atraídos por dicho concepto. El motivo real que encontramos es el elevado precio que tiene el sushi hoy en día en nuestro país. Al ser un producto caro pero muy popular, los puntos podrían beneficiar nuestro sistema y eso lo corrobora la siguiente imagen. La mitad de las personas encuestadas decidieron no consumir sushi dado su precio, y un 30% más aún alguna vez le ha pasado.



Ilustración 5.4.10: Resultados pregunta 5 encuesta

Solo el 1 voto de todos los encuestados no le llamó la atención la propuesta de los puntos, a pesar de que la propuesta fue sin explicaciones sobre el funcionamiento ni detalles de su uso, prácticamente el 100% de los encuestados les pareció una idea realmente interesante. Sumado a lo mencionado antes, el uso de los puntos y los descuentos además les pareció una propuesta que ayudará a fomentar el consumo entre semana, aumentando las ventas totales y así aumentando las ganancias.



Ilustración 5.4.11: Resultados pregunta 6 encuesta

En conclusión, la etapa de empatía nos dio una muy buena primera impresión, que cumplió con lo que veníamos pensando e investigando acerca de cuán útil y exitoso puede ser el sistema y si verdaderamente será utilizado de forma eficiente. Además, nos ayudó a construir una hoja de ruta que sería usada como guía, para poder entender cual es verdaderamente el enfoque de las funcionalidades del producto que estábamos comenzando a desarrollar. Por último, analizar las plataformas existentes nos ayudó a definir cuál debía ser nuestro diferencial para que pueda tener éxito en un mercado donde ya existen plataformas que tengan el mismo fin que nosotros: venta de comida online.

Definir



La etapa de definición de *Design Thinking* es donde sintetizamos toda la información recopilada durante la etapa de empatía para definir claramente el problema a resolver. En este caso identificamos las necesidades e insights de cada perfil de usuario y con esa información nos reunimos con el cliente. Luego, fue el cliente quien marcó el desafío al explicarnos por qué camino quería que fuera el proyecto. Como resultado del extenso análisis llevado a cabo, llegamos al siguiente enunciado en formato “cómo podemos”.

Inicialmente llegamos mediante *Design Thinking* al siguiente desafío:

¿Cómo podemos diseñar la experiencia de un servicio de comida ágil que permita a los restaurantes, aumentar su rentabilidad y al mismo tiempo, brindar a los usuarios, de todas las edades en Uruguay, la posibilidad de recibir recompensas tangibles por referir a sus amigos y familiares, incentivando así la lealtad y la expansión del servicio, al mismo tiempo que aprovechamos las ventajas del marketing multinivel?

Luego de varias iteraciones llegamos a la siguiente sentencia de desafío:

¿Cómo podemos ayudar a mejorar la experiencia de compra de comida online para los clientes de los restaurantes de sushi de Montevideo y aumentar la rentabilidad

para los restaurantes de sushi de Montevideo, teniendo en cuenta que existe competencia desigual, que hay dificultad para fidelizar a los clientes, que la comisión que cobran las aplicaciones existentes son muy elevadas y que hay necesidad de diferenciación entre restaurantes?

Idear



La etapa de ideación es donde generamos ideas creativas y exploramos diferentes enfoques para resolver el problema definido en la etapa anterior. En esta etapa, pensamos en diferentes ideas junto con el cliente para abordar el desafío identificado teniendo como referencia a las necesidades e insights que obtuvimos. Lo que surge de esta etapa será combustible y material para construir los prototipos en la siguiente etapa.

Obtuvimos una lista de funcionalidades, subsistemas y vistas que creíamos serían necesarias. Este documento lo dividimos en dos secciones, funcionalidades básicas y funcionalidades avanzados y a su vez, cada sección la dividimos en secciones más pequeñas que caracterizan un conjunto de funcionalidades. Podemos ver a continuación la ilustración 5.4.12 el ejemplo de una subsección.

3. Funcionalidad básica de los restaurantes:

3.1. Los restaurantes podrán registrar platos con su precio.

3.2. Los restaurantes podrán registrar categorías de comida.

3.3. Los restaurantes podrán activar o desactivar un plato de comida así como editar sus precios y otros campos.

Ilustración 5.4.12: Ejemplo de descripción de funcionalidades

Luego, a raíz de esta lista logramos llegar a la lista de funcionalidades del sistema un poco más detallada. Así como también, decidimos que nuestro sistema lo dividiremos en tres aplicaciones separadas. Esta división fue explicada en la sección 3. Solución. En resumen, una aplicación es para los **Consumidores Finales** que consta del mercado online mediante el cual piden comida, otra para los **Administradores de Restaurantes** que consta de un *backoffice* para manejar su

restaurante y los pedidos y por último, otro para **TuBocado Manager** que corresponde al admin de la aplicación general, donde el *product owner* o alguien que él decida podrá hacer manejo del sistema. Podemos ver un ejemplo en la ilustración 5.4.13. Cabe destacar, que esta lista era una versión original, que luego sufrió modificaciones.

1. Registro de restaurantes: Deben registrarse en el sistema proporcionando toda su información y un administrador debe aprobar el restaurante para que comience a funcionar.
2. Permitir a los restaurantes recibir los pedidos y procesar los pagos.
3. Los restaurantes podrán registrar platos con su precio.
    a. Pueden explicitar si tiene promoción o no con Bocados cambiando la cotización de la relación Boccados - pesos, siempre respetando la cotización base.

Ilustración 5.4.13: Ejemplos de las funcionalidades de la aplicación restaurante

Para ver más detalles de la lista completa de funcionalidades obtenidas en la etapa de ideación véase el Anexo A.16. *Product Backlog.*

Prototipar



La etapa de prototipado consiste en crear representaciones tangibles y simplificadas de las ideas generadas durante la etapa de ideación. En esta etapa realizamos lo que es la pretotipación y la prototipación del sistema. Con el pretotipo llegamos a una primera versión, sin énfasis en el *look-and-feel*, de lo que nos imaginamos como solución final. La idea fue realizar dibujos a mano de las vistas principales que pensamos incluir y esto lo cargamos en una herramienta llamada ProttAPP donde le dimos interacción a dichos dibujos. Con este pretotipo, la idea fue responder la pregunta: si construimos esto, ¿lo usarán/comprarán/consumirán?, y en ese sentido consideramos que tenía que ser construido rápido y sin costos. Podemos ver en la ilustración 5.4.14 el ejemplo de una vista creada en el pretotipo. Para ver el pretotipo completo véase la sección A.12. Pretotipo del Anexo.

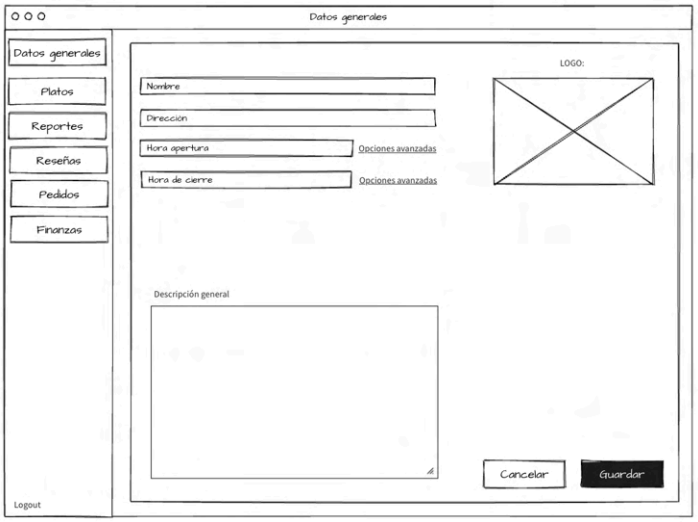


Ilustración 5.4.14: Pretotipo de la vista: Restaurant - Datos generales

Luego de la validación con el cliente y varias pruebas con usuarios que podemos ver en el Anexo A.14. Validación con el cliente y pruebas con usuarios - pretotipo y prototipo, aplicamos los cambios, correcciones y sugerencias que habíamos obtenido y pasamos a la prototipación en donde usamos la herramienta Figma para obtener algo más sofisticado en *look-and-feel* con una UX/UI que sea casi definitiva o casi idéntica al producto final. Un ejemplo del prototipo creado a raíz del pretotipo presentado se puede ver en la ilustración 5.4.15

Dicho prototipo fue construido por dos de los integrantes del equipo en base a su experiencia y su conocimiento brindado por sus respectivos trabajos y sus experiencias como usuarios. Una vez culminado el *Figma* definitivo, decidimos compartirlo con un diseñador gráfico de confianza para poder tener un *Feedback* al respecto. Si bien no hubo correcciones mayores, nos sugirió aplicar ciertos cambios como puede ser el de tipografía o diseño de botones.

Con esto nos volvimos a reunir con el *product owner* hasta que llegamos a la versión final con la cual pudimos especificar las User Stories para luego poder tener un Product Backlog para gestionar el proyecto.

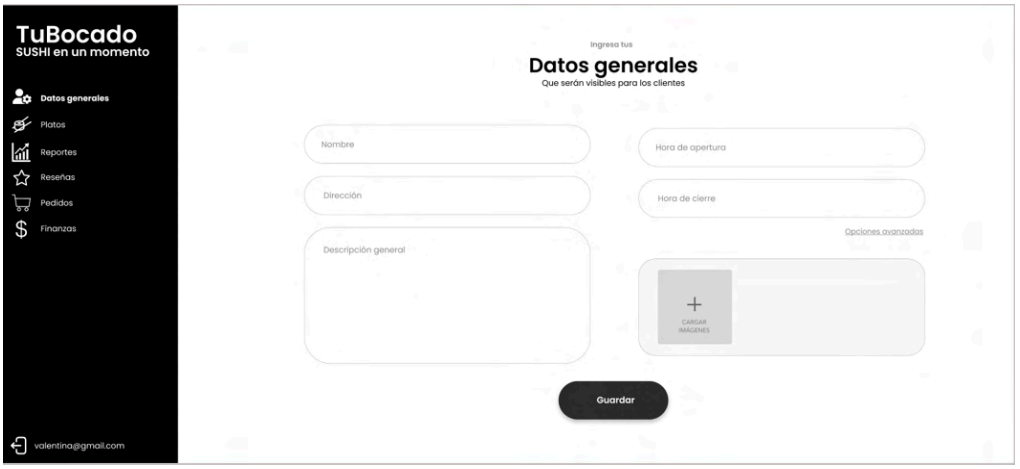


Ilustración 5.4.15: Prototipo V1 de la vista: Restaurante - Datos generales

Prueba



La etapa de prueba implica poner a prueba los prototipos creados con usuarios reales para obtener retroalimentación y validar las soluciones propuestas. Pudimos mostrarle el pretotipo a varios usuarios de diferentes perfiles como son: dueños de restaurantes, usuario típico que pide comida por aplicaciones y al PO. En estas instancias le presentamos lo diseñado y le otorgábamos un tiempo en donde sin explicarles mucho, ellos observaban, intentaban entender y conectar cada una de las secciones. Luego, comentaban su análisis con nosotros y comenzaba una instancia de preguntas tanto de ellos para con nosotros y viceversa. En base a su feedback fue que implementamos algún cambio y mejora, para luego tener una segunda versión del pretotipo y repetir el proceso. Finalmente llegamos a una versión más completa del pretotipo. Lo mismo lo repetimos para la prototipación.

Como mencionamos anteriormente, dentro de la etapa de *Delivery* entra el prototipo probado creado en el proceso de *Design Thinking* que contiene los requerimientos visuales y orales del producto. En este caso, el ciclo de vida es evolutivo en donde por cada *Release*, luego de reunirnos con el *product owner*, vamos a tener versiones adaptadas y mejoradas de lo que es el prototipo. Es decir, por cada *Release*, tenemos una versión del prototipo. A continuación, en las

ilustraciones 5.4.16, 5.4.17, 5.4.18 y 5.4.19 podemos ver un ejemplo de vista creado en Figma en la versión 1, versión 2 y versión 3 respectivamente. En el Anexo A.13. Prototipo podemos ver el prototipo completo con todas sus versiones.

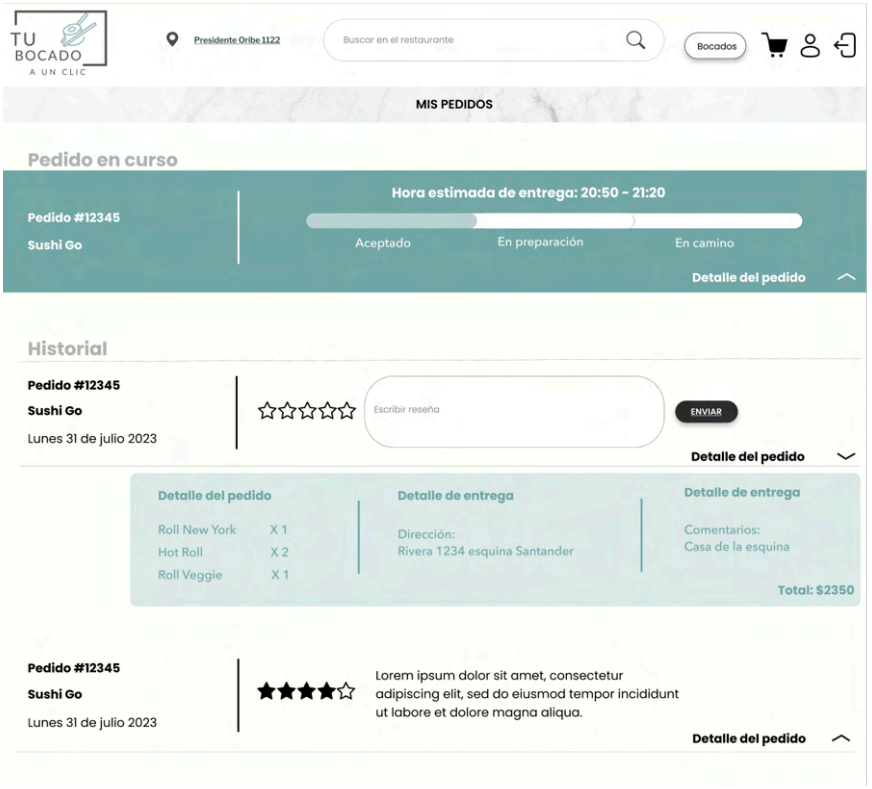


Ilustración 5.4.16: Prototipo V1 historial de pedidos

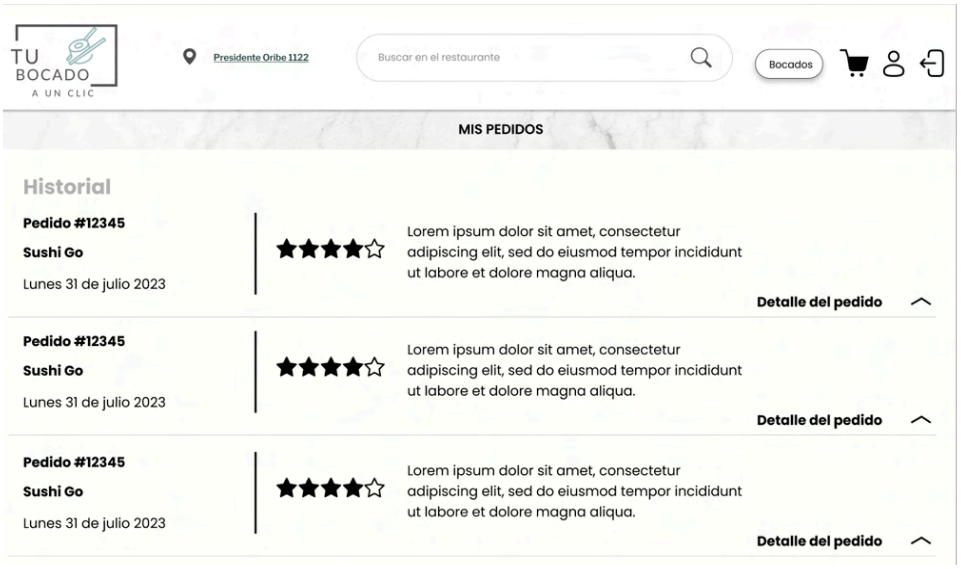


Ilustración 5.4.17: Prototipo V2 historial de pedidos

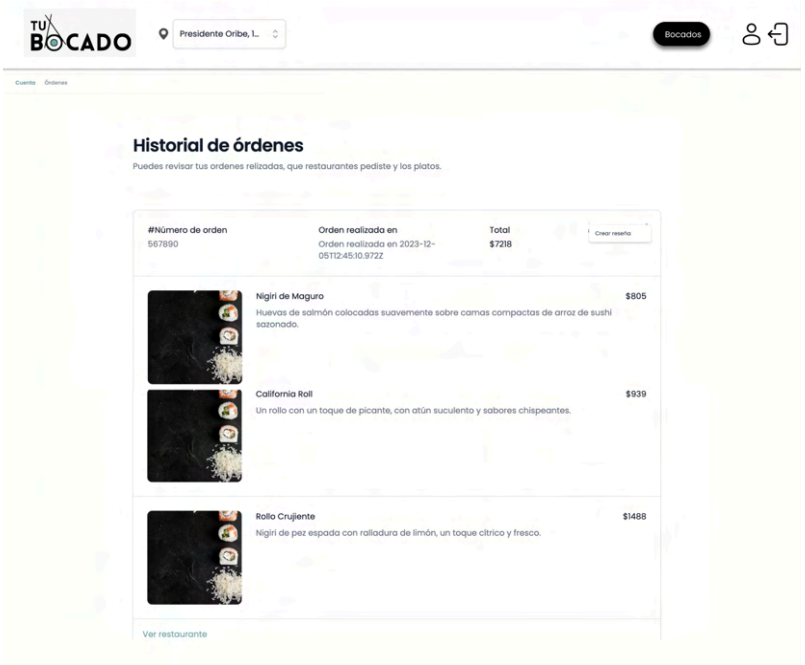


Ilustración 5.4.18: Prototipo V3 historial de pedidos

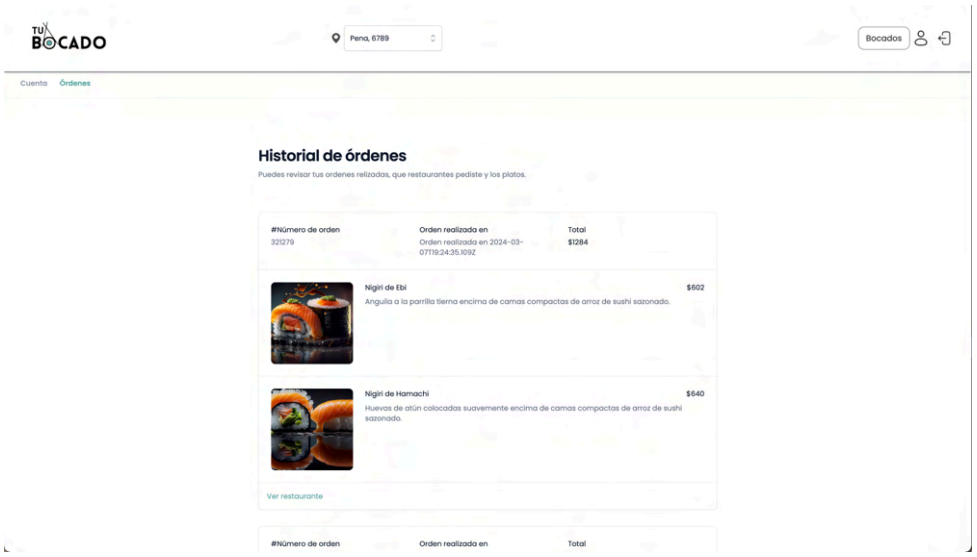


Ilustración 5.4.19: Prototipo V4 historial de pedidos

Una vez finalizado el proceso de *Design Thinking* y con el prototipo que obtuvimos, hicimos una serie de pasos que nos llevó a obtener el *Product Backlog*. En primer lugar creamos un tablero en la herramienta Miro para poder comenzar a escribir y organizar la primera parte de las *User Stories* correspondientes a los requerimientos obtenidos mediante el prototipo y a lo planificado con el cliente, la segunda parte de las mismas corresponde a un nivel más detallado, donde se definen los criterios de aceptación. Creamos entonces una lista preliminar, donde priorizamos las historias de usuario y las organizamos en un estimativo de los

Releases correspondientes, así como también decidimos que en todos los *Releases* íbamos a trabajar en los tres sistemas mencionados: Administrador de Restaurante, TuBocado Manager y Consumidores, para así poder ir avanzando en paralelo y poder probarlos en conjunto. Estas las escribimos con el formato sugerido por BDD:

As a [user role], / **Como** [rol de usuario],
I want to [perform an action], / **Quiero** [realizar una acción],
So that [I can achieve a goal/benefit]. / **Para** [lograr un objetivo/beneficio].

Podemos ver en la ilustración 5.4.20 el ejemplo del *Release* 1 del tablero, en el Anexo A.x. se puede ver el mapa completo.

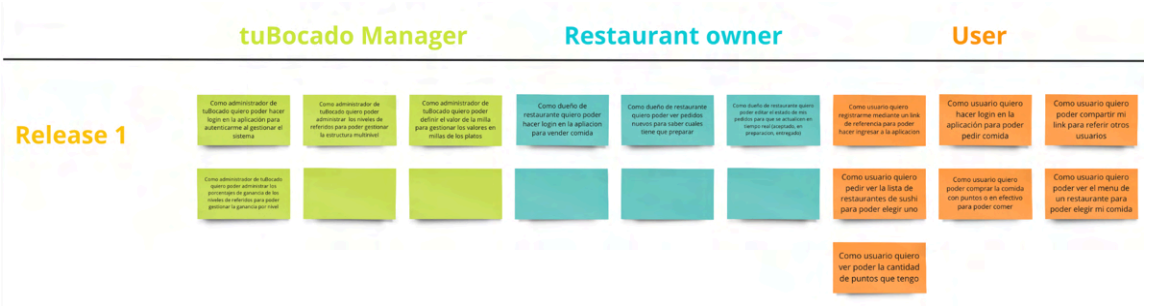


Ilustración 5.4.20: Sección del User story map

Para completar con esta investigación, hicimos un refinado de las historias de usuario obtenidas, este proceso lo llevamos a cabo en conjunto con el tutor y el *product owner*, decidiendo así el alcance del proyecto. Estas las alojamos en un *Product Backlog* en *Click Up*. En la ilustración 5.4.21 podemos ver una breve parte de dicho backlog, para ver el backlog completo véase Anexo A.16 Product Backlog.

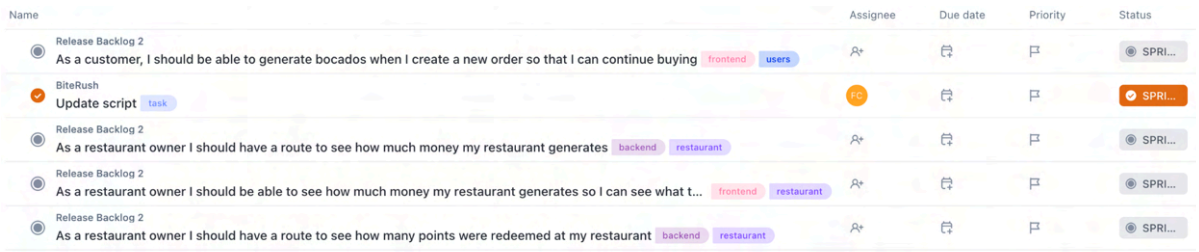


Ilustración 5.4.21: Ejemplo de User Stories en el backlog

Dentro del filtrado de requerimientos, junto con el cliente, decidimos dejar por fuera del alcance algunas funcionalidades, ubicadas en una sección “funciones extra” del Miro, que involucra aquellas funcionalidades que para el cliente no tenían prioridad, pero en caso de tener que ampliar el alcance serían útiles. Un ejemplo de

esto es la incorporación de un nuevo rol dentro del sistema que es *“Delivery”,* que correspondería al encargado de hacer la entrega del pedido a quien lo pide, es decir, en vez de encargarse el restaurante del envío de pedidos, que TuBocado tenga sus propios repartidores de comida, con toda la lógica que eso conlleva. Más adelante en la sección 12. Pasos a seguir hablaremos en detalle de las cosas que quedaron fuera del alcance y que planeamos incorporar en un futuro.

Es importante mencionar que en paralelo al proceso de *Design Thinking* realizamos un relevamiento y definición entre el equipo y el cliente para poder obtener una lista de los requerimientos no funcionales, escritos como texto y guardarlos en el *Product Backlog*. Dichos requerimientos no funcionales podemos encontrarlos en el capítulo 3.1 Requerimientos.

5.5. Criterios de aceptación vs prototipo

En esta sección vamos a comentar sobre dos enfoques en el desarrollo que son: los Criterios de Aceptación y el Prototipo. Tuvimos que tomar la decisión de elegir uno de los dos para que sea la guía del desarrollo.

En nuestro proyecto, llevamos a cabo un proceso donde a raíz del uso de *Design Thinking* obtuvimos un *Product Backlog*, el cual contenía historias de usuario escritas con BDD para luego poder comenzar con los pasos del proceso de ingeniería, el cual incluye la definición de requerimientos, el desarrollo, las pruebas y el despliegue. Cabe destacar que, al usar una escritura tipo BDD para un requerimiento funcional, a la vez se están especificando los casos de prueba de dicho requerimiento, dado que BDD incluye la especificación de los escenarios para definir los criterios de aceptación. Llevar a cabo dicho proceso nos fue muy útil ya que es una forma de facilitar la realización de pruebas.

Inicialmente el uso de los criterios de aceptación para guiar el desarrollo mediante BDD nos resultó positivo. Este método lo llevamos a cabo los primeros tres *Sprints*, es decir, el primer *Release*. Luego, notamos como equipo y en conjunto con el tutor, que estábamos realizando una gran inversión en *testing* cuando quizá no era necesario, dado que el proyecto se trata de una POC. Por ende, decidimos cambiar y probar utilizar la validación directa con el prototipo en los siguientes *Sprints* con el objetivo de comparar ambas formas de trabajar. Durante el proceso,

notamos que el enfoque del prototipo nos resultaba más efectivo en términos de visualización y validación de conceptos de diseño y funcionalidad, además de ganar tiempo para invertirlo en otras actividades. Esto nos llevó a optar por continuar el resto del desarrollo utilizando prototipo.

Utilizar el prototipo nos pareció más eficiente ya que no dedicamos tiempo a pensar y escribir los criterios, sino que usamos algo que ya tenemos. Además, mediante dicho tiempo de prueba notamos que todos somos más visuales al momento de entender el producto final, al igual que nos pareció más sencillo de presentar y validar con los usuarios.

Los pasos a seguir en cada uno de los enfoques eran los siguientes:

Criterios de aceptación

1. Para los desarrolladores, se les dan los criterios de aceptación, lo leen y programan.
2. Para la prueba exploratoria, se les dan los criterios de aceptación al explorador, el va mirando y registra lo que se le ocurre.
3. Para las pruebas guiadas, arman la planilla de excel en base a los criterios de aceptación y después el tester ejecuta según la planilla de excel.
4. Para la *Release Review*, vamos leyendo los criterios de aceptación y en paralelo vamos ejecutando el *software*, según lo que digan los criterios. Si coincide me lo tiene que aceptar, si no lo hace es un *bug*.

Prototipo

1. Para los desarrolladores, les da el prototipo, lo mira y programan.
2. Para la prueba exploratoria les dan el prototipo al explorador, el va mirando y registra lo que se le ocurre. Comparar prototipo contra producto final.
3. Para las pruebas guiadas, arman la planilla de excel en base al prototipo y después el tester ejecuta según la planilla de excel, es decir, va ejecutando el prototipo y va armando la planilla.
4. Para la *Release Review*, vamos ejecutando el prototipo y en paralelo vamos ejecutando el *software*. Si coincide me lo tiene que aceptar, si no lo hace es un *bug*.

Así todo, si bien decidimos hacer uso del prototipo para llevar a cabo el desarrollo del sistema, decidimos continuar utilizando los escenarios para cada user story para realizar las pruebas con usuarios. Estos escenarios los usamos de guía para llevar a cabo dichas pruebas, teniendo una hoja de ruta para saber que probar y qué resultado esperar. Mencionamos un poco más en profundidad este procedimiento en la sección 5.10 Proceso de desarrollo.

5.6. Roles

En esta sección explicaremos los roles que nos dimos en el proyecto. Como ya mencionamos, utilizamos una personalización de Scrum, y esta metodología propone algunos roles que decidimos incorporar. Dichos roles son s*crum master*, *product owner* y equipo de desarrollo.

Como ya sabemos, el equipo constaba de 5 integrantes (4 estudiantes más el tutor). Decidimos utilizar los tres roles antes mencionados, *scrum master*, *product owner* y equipo de desarrollo, y agregar los roles de Ingeniera en Requerimientos, Arquitecta y DevOps. En la tabla 5.6.1 podemos ver cómo dividimos esos roles entre los integrantes y una breve descripción de cada uno.

Decidimos incorporar estos tres roles dado que Scrum no nos brindaba uno específico que cubra las tareas y especializaciones que necesitábamos. En cuanto a ingeniería de requerimientos, lo incorporamos dado que decidimos utilizar *Design Thinking* por lo que necesitábamos un rol que se haga cargo de su correcta aplicación. El rol de arquitecta tampoco lo cubre Scrum y creímos necesario un rol enfocado en diseñar y elegir las tecnologías más acordes. Por último DevOps es una parte del proceso que Scrum no cubre y prescindimos de alguien que se encargue de las planificaciones, controles y políticas a cumplir.

El rol de *scrum master* fue variando a medida que avanzaba el proyecto y nuestro dominio de Scrum. Dado que al principio del proyecto nuestro conocimiento sobre Scrum era bajo, este rol fue cubierto por el tutor, para luego pasar a un estudiante. Por último, el *product owner*, que en este caso es también nuestro cliente. El fue con quien nos comunicamos constantemente y fue quien determinó el alcance, los *Releases* y otras decisiones importantes del proyecto.

Rol	Descripción	Responsable(s)
Product Owner	Representar al cliente a través de la toma de decisiones sobre el producto definiendo características y funcionalidades.	Nicolás Rostán
Ingeniera en requerimientos	Aplicación de *Design Thinking*	Jimena Sanguinetti
Arquitecta	Diseño de la arquitectura y elección de las tecnologías utilizadas	Valentina Rostán
DevOps	Llevar a cabo la planificación, control de la configuración, versionado y políticas de deployment	Federico Torres y Federico Charbonnier
Desarrolladores	Diseño detallado, programación y pruebas unitarias	Jimena Sanguinetti, Valentina Rostán, Federico Torres y Federico Charbonnier
Scrum Master	Asegurar que el equipo siga los principios y prácticas de Scrum y alcance sus objetivos	Valentina Rostán

Tabla 5.6.1: Tabla de roles

5.7. Selección de tecnologías

En relación a las tecnologías que decidimos usar, tuvimos que tener en cuenta que el cliente nos sugirió algunas, pero en todo momento nos aclaró que no estaban cerrados a que el equipo plantee alternativas de manera justificada y cambiar así la selección que ellos sugirieron, por lo que como equipo aumentamos nuestra capacidad de decisión. Por ejemplo, el cliente nos informó que tenía como preferencia el uso de ReactJs para el *frontend,* principalmente, por la facilidad de mantenimiento, pero también incluimos otras opciones para evaluar si alguna era una alternativa mejor.

Teniendo esto en cuenta fue que realizamos una investigación de tecnologías a usar comparando las sugerencias del cliente y algunas otras opciones que creímos que podían aplicar. En primera instancia decidimos cuáles eran las tecnologías que queríamos comparar en cada sección, además de los atributos que nos interesaban comparar dentro de cada uno y el peso que cada uno tiene, es decir, su prioridad: 1 baja, 2 media y 3 alta.

A raíz de esto, creamos una matriz de pugh para cada área, la cuales nos permitieron analizar las tecnologías y compararlas entre ellas para ver cual se

adecuaba más al proyecto y al equipo. En conjunto a esto, este proceso implicó una investigación y análisis comparativo de las opciones disponibles, utilizando para ello tablas comparativas basadas en los mismos criterios que habíamos utilizado para las matrices de Pugh. Estas tablas, diseñadas para facilitar una visión general de las ventajas y desventajas de cada tecnología, se explican en detalle en el capítulo 4. Arquitectura y Diseño.

Por lo tanto, nuestra metodología de decisión de tecnologías comenzó con la identificación de las necesidades tecnológicas clave del proyecto, seguido por la evaluación de diversas opciones utilizando las matrices y tablas comparativas como guías y por último un análisis de todos los datos para tomar las decisiones. Este enfoque nos permitió comparar objetivamente las tecnologías entre sí. A continuación, planteamos cuáles fueron nuestras decisiones respecto a cada una de estas tecnologías junto con una breve descripción de por qué tomamos dicha decisión.

Adentrándonos en el *backend*, desde un comienzo los integrantes nos inclinamos por NodeJs dado que es el lenguaje que usamos hoy en día en nuestros trabajos y que además adquirimos en facultad pero el cliente nos propuso usar Laravel por lo que tuvimos que investigarlo. Con el hosting sucedió algo similar, el cliente sugirió Azure pero luego se vio interesado en cambiar a AWS, por lo que al momento de tomar la decisión tuvimos que informarnos, comparar, analizar y con eso planteárselo al cliente. Por otro lado con lo que respecta al *frontend* decidimos utilizar ReactJs dado que tiene un buen ecosistema y al tener tanto *backend* como *frontend* basado en JS la curva de aprendizaje era mucho menor. Luego con lo que respecta a las bases de datos, decidimos optar por bases de datos relacionales con motor en base de datos Postgresql, esto fue mayoritariamente por petición del cliente pero igualmente investigamos diversas opciones.

En el Anexo A.17. Matrices de Pugh podemos ver todas las matrices que realizamos sobre tecnologías y a continuación se presenta la tabla 5.7.1 con un ejemplo concreto para lenguaje de *frontend*.

Atributo	Peso	ReactJs	AngularJs	VueJs
Definición de la tecnología	1	1	1	1

Reutilizacion de codigo	2	1	0	1
Performance	2	1	0	1
Lenguaje de programación	2	1	1	1
Seguridad	3	1	1	1
Soporte	2	1	0	1
Costo	1	1	1	1
Curva de aprendizaje	3	1	0	1
Fiabilidad	2	1	1	1
Experiencia del equipo	3	1	0	-1
Suma		19	9	13
Ranking		1	3	2

Tabla 5.7.1: Ejemplo de Matriz de Pugh para lenguajes de *frontend*

En esta tabla podemos observar cómo realizamos una parte del análisis para seleccionar las tecnologías de *frontend* más adecuadas para nuestro proyecto, eligiendo como opciones a evaluar tanto ReactJs como AngularJs y VueJs. En este proceso no solo nos basamos en la evaluación de atributos técnicos definidos previamente, sino también en considerar la relevancia de cada uno de estos atributos en relación con los objetivos específicos de nuestro proyecto y nuestras competencias. La ponderación y calificación de cada criterio muestran nuestro compromiso con una toma de decisiones informada y basada en evidencia, asegurando que las tecnologías seleccionadas sean la mejor elección para nuestro proyecto.

Las decisiones y justificaciones detalladas de cada una de las tecnologías, así como los análisis realizados los detallamos en mayor profundidad en la sección 4.3. Stack Tecnológico, donde profundizamos en las razones específicas que nos llevaron a elegir por NodeJs para el backend, ReactJs + Typescript para el *frontend*, Postgres para la base de datos, y AWS para el hosting. Mientras que, a través de esta sección, intentamos proporcionar una visión clara de todo el proceso de selección tecnológica que pasamos.

5.8. Capacitaciones

Para continuar con las tecnologías, decidimos tomar ciertas capacitaciones en diversas tecnologías. Dos de los integrantes no contaban con mucha experiencia usando *ReactJs* por lo que tomaron un curso de seis semanas para poder estar más

preparados al comenzar el desarrollo. Estos cursos a su vez constaban con una instancia de pruebas y simulaciones las cuales fueron muy útiles para ampliar el entendimiento del lenguaje.

Todos los integrantes recibimos una breve capacitación sobre Azure en donde aprendimos lo que el cliente deseaba que supiéramos, investigamos las características, sus servicios de computación, almacenamiento, redes y servicios de aplicaciones, así todo no llegamos a hacer pruebas dado el cambio de parecer del cliente.

Esto fue así ya que en un principio, el cliente nos pidió que alojáramos la aplicación utilizando Azure y así fue que investigamos, pero luego, una vez que íbamos a deployar, el product owner nos comunicó que deseaba cambiar a utilizar los servicios de AWS. En este sentido, fue que nos adaptamos a sus necesidades de cambio e investigamos sobre AWS.

Por último, uno de los integrantes tomó un curso de *Peregrinus* sobre Scrum de 3 días de duración, 6 horas cada día. Por más que todos estamos familiarizados con la metodología, nos pareció oportuno que al menos uno tenga un amplio conocimiento de la misma.

Pruebas de tecnología (*Spikes*)

Además de las capacitaciones mencionadas, decidimos realizar pruebas de las tecnologías para explorarlas y entenderlas un poco más antes de comenzar con la etapa de *Delivery*. En nuestro caso decidimos hacer "*spikes*", que son investigaciones o pruebas rápidas para evaluar la viabilidad y comprensión de tecnologías antes de su implementación completa. Estos spikes se enfocan en funcionalidad, seguridad y costos.

- Investigación de pasarela de pagos:

Evaluamos algunas opciones de pasarelas de pagos existentes, como Mercado Pago, que nos permitieran integrarlo con la aplicación y así facilitar las transacciones en la compra y venta de comida. Como mencionaremos en el capítulo de Arquitectura, comenzamos investigando el funcionamiento de PayPal ya que es muy popular y creímos que se ajustaría bien a nuestro proyecto, además

investigamos MercadoPago y realizamos una tabla comparativa entre cada uno para poder analizar las ventajas y desventajas de cada uno. Luego el cliente nos mencionó Mercado Pago y no nos impuso esta tecnología, pero nos la sugirió de forma contundente, por esto nos pusimos a investigar esta herramienta también.

Una vez inclinados por la utilización de Mercado Pago (tanto el equipo como el cliente), nos pusimos en marcha para realizar un spike del mismo. Nos encontramos con que Mercado Pago contaba con una implementación bastante sencilla, que además nos sería muy útil para el proyecto. Mercado Pago además contaba con credenciales de prueba que podemos ver en la ilustración 5.8.1 y tarjetas de débito/crédito de prueba, vistas en la ilustración 5.8.2, lo cual nos permitieron realizar pequeñas pruebas antes de hacer la implementación. También contaba con una gran documentación, pudimos analizar las características y los requisitos técnicos y cuando terminamos la spike nos encontrábamos con que teníamos una buena comprensión del uso de la misma y creímos que era adecuada para nuestro proyecto, por lo que decidimos seguir adelante con esta.

Tarjeta	Número	Código de seguridad	Fecha de caducidad
Mastercard	5031 7557 3453 0604	123	11/25
Visa	4509 9535 6623 3704	123	11/25

Ilustración 5.8.1: Tarjetas de prueba brindadas por Mercado Pago



Ilustración 5.8.2: Credenciales de prueba brindadas por Mercado Pago

- Investigación de tiempo real del estado del pedido y órdenes:

En este caso, el objetivo estuvo en explorar y evaluar las tecnologías que permiten ofrecer a los clientes actualizaciones en tiempo real sobre el estado de sus pedidos así como también las órdenes que llegan al restaurante. Lo principal que evaluamos acá son los websockets mediante alguna librería que permitiera llevar a

cabo la concurrencia entre los sistemas, ampliaremos sobre su uso en la sección de sockets. Antes de comenzar a implementar la solución debimos investigar la librería Sockets.io para ver cómo era su funcionamiento, exploramos su documentación y cómo sería la integración con nuestro sistema. Mediante este spike pudimos analizar que tan viable técnicamente era integrar la librería y si está efectivamente serviría para lo que nosotros deseábamos usarla. Además nos permitió adelantarnos e identificar posibles desafíos con los que nos podíamos encontrar.

- Investigación de AWS:

En cuanto a AWS, en la Universidad ORT se nos brindó a los estudiantes un breve curso donde todos los integrantes pudimos hacer pruebas pequeñas con varios de los servicios brindados. De este modo se nos facilitó el uso de esta tecnología, además de recibir ayuda cuando nos surgieran dudas. En conjunto con esta capacitación sumamos una exhaustiva investigación sobre qué servicios deseamos usar, cuáles podrían sernos útiles y cómo utilizarlos.

5.9. Delivery

Una vez culminada la etapa de *Discovery*, con el prototipo pronto y validado, la lista de *user stories*, las tecnologías y la metodología decidida nos adentramos en la etapa de *Delivery*, en donde íbamos a comenzar el desarrollo del proyecto.

En relación a esta etapa, en un principio teníamos indicios de que íbamos a utilizar alguna metodología ágil, y a medida que avanzó el proyecto fuimos descubriendo cuál era la mejor y más óptima forma de trabajo. Además, definimos la implementación de BDD para algunas de las iteraciones de los *Sprints* y luego solo hicimos uso de algunas prácticas de BDD que explicaremos más adelante.

Ciclo de vida

Respecto al ciclo de vida correspondiente al marco de trabajo, en esta etapa optamos por un enfoque iterativo, donde cada una de las iteraciones realizadas eran de igual longitud y tenían como salida un incremento de valor para sustentar el valor funcional del producto y su potencial de mercado.

En estas además fuimos modificando el prototipo, obteniendo así diferentes versiones del mismo. Es decir, actualizamos y extendimos el prototipo por cada *Release* en donde luego de obtener feedback del PO y los resultados de las pruebas con usuarios, actualizamos y adaptamos la última versión a una nueva con mejoras. De esta forma estaríamos estirando D*esign Thinking* hasta *Delivery*, y en lugar de poner una etapa detrás de otra en realidad se superponen un poco.

Teniendo en cuenta que decidimos finalmente utilizar una versión personalizada de la metodología Scrum, decidimos hacer uso de las iteraciones denominadas *Releases* y *Sprints*. La división fue de la siguiente forma: 4 *Releases*, de 1 mes y medio de duración cada uno siendo el primero agosto-septiembre y el último diciembre-enero. Además, cada *Release* lo dividimos en *Sprints*, siendo cada *Release* compuesto por 3 *Sprints* y cada *Sprint* 2 semanas, sumando un total de 12 *Sprints*. Sin embargo, en el *Release* 1 tuvimos una falla en la estimación de tiempos al comienzo y esto nos llevó a tener que crear un nuevo *Sprint*: *Sprint* 4 de una semana de duración para poder llevar a cabo el proceso de *Deploy,* en la parte de

ejecución, específicamente en el proceso de *deployment* lo explicaremos en más detalle.

Inicialmente elaboramos un *Release Plan* que comprendía 4 entregas, correspondientes a cada uno de los *Releases*, y en este plan identificamos de manera preliminar las historias de usuario que entregaremos en cada una de ellas.

5.10. Ejecución

Como ya mencionamos, para la gestión del proyecto decidimos utilizar la metodología Scrum personalizada, así como también hicimos uso de algunas prácticas de BDD para el desarrollo de ingeniería de *software*. A continuación se muestra la ilustración 5.10.1 que presenta el ciclo de desarrollo de ingeniería de *software*.

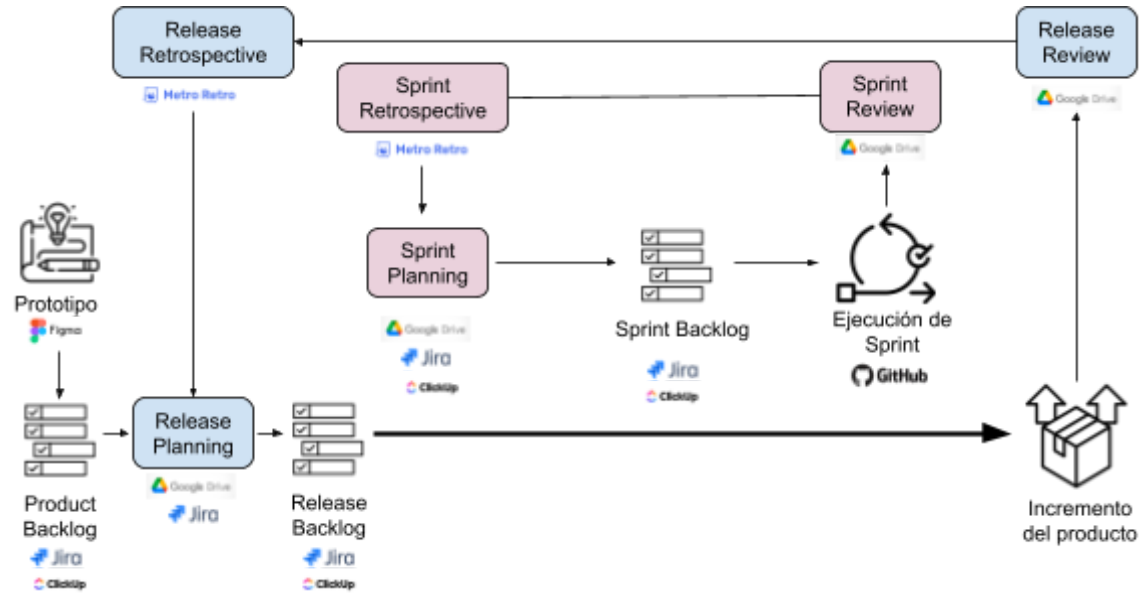


Ilustración 5.10.1: Ciclo de desarrollo

Como se detalla en la imagen trabajamos en ***Releases*** los cuales se componen de ***Sprints*** de dos semanas, dividiendo funcionalidades en requisitos discretos y pequeños, llamados *user* stories. Este enfoque nos permitió llevar a cabo un desarrollo iterativo y flexible, donde priorizamos las características más importantes para cada entrega, realizamos revisiones y ajustamos el trabajo en función de la retroalimentación recibida, lo que nos condujo a una evolución constante y a la mejora continua del producto.

En primer lugar obtuvimos el ***Product Backlog,*** que surge a raíz del proceso ya mencionado, y consta de un listado de todos los requerimientos que el *software* en desarrollo debía cumplir. Estos requerimientos estaban escritos en formado de ***User Story***.

Release Planning Meeting

Comenzando a describir el ciclo de desarrollo, se puede ver que cada *Release* comenzaba con esta ceremonía donde participábamos todo el equipo y el cliente. Aquí planificábamos y definíamos qué funcionalidades del producto y, en caso de existir, *fixes* que apliquen incluiríamos en ese *Release* específico y así obteníamos lo denominado *Release Backlog* con las *User* Stories seleccionadas.

Como ya hemos mencionado, nuestro sistema se basa en la cooperación entre tres sistemas funcionando en paralelo: el mercado donde los consumidores pueden ver los productos, el *backoffice* para el manejo de los restaurantes y el *backoffice* para todo el manejo de la lógica del *marketing* multinivel y del sistema en sí. Para poder dividir las funcionalidades de modo de favorecer el desarrollo del producto y además ser coherente con el avance de los tres sistemas decidimos dividir las *User Stories* en estas tres categorías. Esta clasificación la hicimos mediante las *Labels* que nos brinda tanto *Click Up* como *Jira*, que es la otra herramienta utilizada para registrar el tablero, mencionaremos más sobre el cambio de herramienta más adelante.

En las ilustraciones 5.10.2 podemos ver los tres tipos de tags distintos en tres *user stories* diferentes, tanto en *Click Up* cómo en *Jira* respectivamente. La ilustración muestra por un lado la *Tag* de *Consumers* que corresponde a aquellas historias de usuario relacionadas a los consumidores finales del producto en desarrollo, es decir, quienes acceden al mercado para comprar comida. Por otro lado está *Manager* que corresponde a las historias de usuario relacionadas con el *backoffice* que utilizará ya sea el dueño de TuBocado, o quien él decida que hará manejo del sistema. Dicho manejo hace referencia a definir los parámetros que el sistema tiene, tener control de los restaurantes, ver un *dashboard* de datos útiles, entre otras. Por último, la tercera etiqueta corresponde a *Restaurant,* que trata de las historias de usuario relacionadas nuevamente con el *backoffice* pero en este

caso el que usará el administrador de cada restaurante. Este sistema incluye la carga de información acerca del restaurante, los platos, las categorías, entre otros. Ver más sobre estos sistemas en la sección 3. Solución.

Esta división nos fue sumamente útil para poder tener una trazabilidad acerca del desarrollo, poder analizar cómo venía cada sistema independiente y además poder mantener una buena organización. Este último punto fue muy importante ya que por más que cada sistema funcione independiente, algunas funcionalidades dependen de otras, correspondientes a sistemas diferentes. Por ejemplo, para poder realizar una orden siendo un consumidor, es necesario que existan platos disponibles en el sistema, y eso solo es posible si un administrador de restaurante cargó alguno en el *backoffice.*

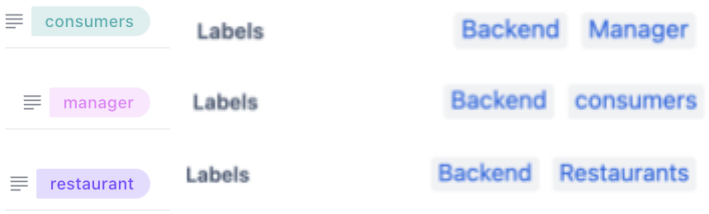


Ilustración 5.10.2: Los tres tipos de Tags disponibles en Click Up y Jira

Decidimos incluir en cada *Release* las funcionalidades que estén relacionadas con todos los sistemas, manteniendo una coherencia entre lo que hacíamos en cada uno, además de alcanzar construir un MVP con esas funcionalidades que permitan hacer pruebas con usuarios en ambiente de producción.

Para poder hacer esta selección de funcionalidades nos basamos en nuestra experiencia tanto laboral como de toda la carrera universitaria. Esto se debe a que, la estimación de cada una de las historias de usuario puntualmente la llevábamos a cabo al comenzar un *Sprint*, por ende, al momento de seleccionar las historias para el *Release Backlog* no contábamos con una estimación formalmente realizada, y por eso nos basamos sobre todo en lo que íbamos aprendiendo y en nuestro conocimiento previo. Así todo, sin ser por el *Release* 1, para los demás ya contábamos con una experiencia y un conocimiento basado en lo que ya habíamos desarrollado por lo que pudimos estimar con confianza. Dado que en el *Release 1*

no contábamos con esta confianza, decidimos no estimar y solo generar información para luego poder estimar con éxito en los siguientes *Releases*.

Por último, el *product owner,* que estaba presente en estas reuniones, nos guiaba en lo que le parecía que tenía mayor prioridad dentro del *Release*, lo cual tendríamos en cuenta en las planificaciones de los *Sprints*.

Al concluir cada *Release* con dicho subconjunto de funcionalidades, se generaba un incremento que lograba cumplir con las necesidades básicas de los usuarios respecto al problema planteado al comienzo, aunque no con la totalidad de las funcionalidades, a no ser que sea el último *Release*. Es por esto que planeamos cada *Release* de forma que compongan un MVP. Una vez obtenido el *Release Backlog* completo, pasábamos a identificar los riesgos que aplicaban a ese *Release*, y con ellos creábamos un *Risk Release Backlog*, se puede ver más información sobre los riesgos en la sección 6. Gestión de Riesgos.

Por último, hacíamos una validación de la *Definition of Done* que ya teníamos para determinar si requería actualizar su especificación. En el caso de la primera *Release Planning Meeting* lo que hicimos fue, justamente definir que implicaba la *Definition of Done.* Esta es una lista de características que debe cumplir el producto que estamos desarrollando para poder clasificarlo como "*Done*", es decir, terminado. Este listado nos brinda un panorama general sobre el estado completo del producto, favoreciendo así la calidad integral de todo el sistema, facilitando la entrega del mismo y permitiéndonos medir el progreso global. Véase el Anexo A.18 *Definition of Done* de cada *Release* y la sección *Definition of done* para más detalle.

Cabe destacar que las reuniones de planificación de *Releases* las realizamos en el contexto de la primera *Sprint Planning Meeting* de cada *Release*.

Sprint Planning Meeting

Las *Sprint Planning Meetings* las realizábamos al comienzo de cada *Sprint.* En nuestra personalización de Scrum decidimos que el *product owner* no participaría de estas reuniones dada su disponibilidad limitada. Además, el PO estuvo de acuerdo con esta decisión ya que todos consideramos que ya teníamos la suficiente experiencia y conocimiento del dominio como para tomar decisiones

dentro de los *Releases* gracias a las investigaciones previas y la aplicación de *Design Thinking.*

Optamos por *Sprints* de dos semanas de duración, ya que buscamos un balance entre la agilidad necesaria para responder rápidamente a cambios y el esfuerzo requerido para un enfoque detallado en cada *Sprint*. Consideramos que un ciclo de una sola semana era muy corto, especialmente dado que trabajamos y cursamos algunas materias de la carrera simultáneamente. Un *Sprint* de una semana nos habría obligado a dedicar mucho tiempo en ceremonias, haciendo que tengamos menos tiempo para dedicarle al desarrollo. Esto nos permite planificar y ejecutar de manera dinámica, adaptándonos a los cambios sin sacrificar la calidad o el alcance del proyecto por restricciones de tiempo.

En las *Sprint Planning Meeting* definíamos los objetivos del *Sprint*, el *Sprint backlog*, los recursos disponibles, así como también el ***Risk Sprint Backlog***; que es la lista de riesgos que aplican a este *Sprint*, tomada de el *Risk Release Backlog*.

Luego, pasábamos a formar el ***Sprint Backlog***, que está compuesto por un subconjunto de *User Stories* del *Product Backlog.* Este se trata del listado de requerimientos que serían desarrollados a lo largo de ese *Sprint*. Para formarlo tomábamos las *User Stories* correspondientes al *Release Backlog* y realizamos una votación teniendo en cuenta lo priorizado con el *product owner* y teniendo en cuenta la distribución de *User Stories* de forma de no bloquear entre sí mismas y entre sistemas. Además, a partir del *Release* 2 comenzamos a definir los escenarios de prueba para cada una de las historias de usuario, dado que este proceso fue eliminado de la ejecución de un *Sprint*.

Como siguiente paso, estimábamos el tamaño de las *User* Stories en *story* points. Para este proceso usamos la herramienta *Planning Poker*. Para llevar a cabo dicha técnica, hicimos uso de una herramienta online en donde cada uno debía votar los *Story Points* que estimaba, luego revelábamos los votos de cada uno y ahí podían suceder dos escenarios: o todos coincidimos o habían discrepancias. En caso de coincidir, tomábamos ese valor como la estimación pero en caso de existir discrepancias, los 2 integrantes que votaron los extremos (superior e inferior) debían dar sus razones. Una vez explicados los motivos de sus votos, sin decir más

información repetíamos el proceso: los 4 integrantes volvían a votar. Nuevamente podía suceder que hubiera discrepancias, de ser así, repetíamos el proceso anterior. Podemos ver un ejemplo de la votación en la ilustración 5.10.3.

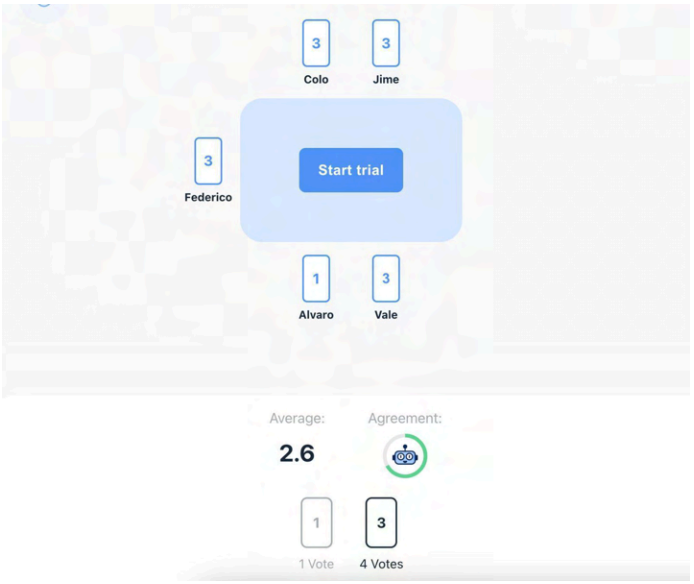


Ilustración 5.10.3: Ejemplo de Planning Poker

La primera vez que hicimos uso de esta herramienta fue al finalizar el primer *Release*. Como ya hemos mencionado, este no fue estimado, por lo que solo contábamos con las horas reales que había llevado cada historia de usuario. Estas horas fueron usadas al finalizar el último *Sprint*, dado que nos juntamos con el tutor y este nos explicó cómo se usaba la herramienta. En primer lugar, hicimos la votación y llevamos a cabo todo el proceso mencionado sobre su uso, luego, una vez que teníamos todas las historias de usuario asociadas a un número de *story points* y teniendo las horas reales de duración, pudimos obtener una relación *Story Points*-horas.

Luego, al ya tener nuestra relación definida, en la primera *Sprint Planning* que estimamos (*Sprint* 5), llevamos a cabo una estimación en *Story Points* de cada historia de usuario a realizarse del *Sprint Backlog.* Una vez que obtuvimos la tabla con este dato, dividimos cada historia de usuario en tareas, las cuales corresponden a cada uno de los pasos que debe cumplir la misma hasta llegar a *Done,* es decir, todas las columnas del tablero que utilizamos. Esta división nos ayudó a estimar el esfuerzo medido en horas-persona de cada una de dichas tareas, donde la suma

total de horas de las tareas que componen una historia de usuario, refleja el esfuerzo total de la misma.

Esta estimación de cada tarea únicamente la registrábamos en el tablero, ya que en la planilla de Excel solo registrábamos las estimaciones y los valores verdaderos de las historias de usuario. Cabe mencionar que para mayor claridad y organización del equipo, cada historia de usuario la dividíamos en *sub-user-stories* correspondientes al *backend* y *frontend*, donde a cada una le correspondía un porcentaje de *Story Points*. Esta división nuevamente la registrábamos sólo en el tablero y además no requería de un proceso formal para llevarlo a cabo, sino que lo realizábamos en base a nuestro conocimiento y experiencia.

Por último, en cuanto al manejo de los *fixes,* este era similar al de las historias de usuario, estos eran estimados en esfuerzo y velocidad, independientemente de la historia de usuario de la cual se desprendía. Esto era así siempre y cuando fuese un *fix* registrado en el *Product Backlog*, véase el manejo de fixes para ahondar en el proceso llevado a cabo para manejarlos. Sin embargo, algunos no eran registrados, dado que se trataba de errores pertenecientes al desarrollo, es decir, antes de haber llegado a estar fusionados en la rama *dev*, era este el caso en el que el *fix* no lo estimábamos, agregando esfuerzo pero no tamaño a la *User Story* de la cual se desprendía.

Todas las estimaciones recabadas las registramos en el tablero de *Jira* pero además en una planilla Excel en donde se encuentran todas las historias de usuario estimadas, véase un ejemplo de dicha tabla en la ilustración 5.10.4, para ver la tabla completa vea el Anexo A.19. Historias de usuario estimadas. Esto nos ayudaba también para poder sustentar y validar las estimaciones ya que comparábamos los valores estimados del *Sprint* que estaba siendo estimado con la velocidad y capacidad promedio de *Sprints* anteriores, siendo la velocidad la cantidad de *Story Points* que hacemos por *Sprint.*

As a customer, i should be able to cancel an order so that i can order again something different	5	5	11	9,32
As a customer, If i cancel an order I want to see my bocados return to my account	5	5	14	11,33
As a tuBocado manager, i want to get the information of a specific restaurant so that I can manage it correctly	4	4	8	4,68
As a tuBocado manager, i should be able to accept or decline restaurant requests so that they can use the system	8	5	29	23,5
As a tuBocado manager I want to have inicial information for customer, levels and parameters.	3	3	4	3,5
As a tuBocado manager I should be able to delete customer points so that we could have a correct history of the customer transactions	4	4	4,5	4,31
As a restaurant owner, I want to have the ability to view real orders for my restaurant, so that I can efficiently manage orders process.	5	5	19	17,36
As a tuBocado manager I want to have an admistrator so I can use the application	2	2	2,5	2,33
As a developer I should have images in dishes and restaurants so I can test front	2	3	3	3,12
As a developer I should be able to generate orders so I can test the front	4	2	4,5	2,34
As a developer, I want to configure S3 so i can upload images	2	2	2	1,35
As a developer, I want to create pending transactions when an order is created	4	4	6	5,67
Fix/delete database before running	2	2	1,5	2
Fix: Math floor on user bocados	1	1	0,5	0,5
Fix: Display errors on the Frontend	1	1	0,5	0,5
Fix: Get orders for restaurants to manage undefined	2	2	1,5	2
Fix, change user seed UUID and use the same instance of the database for the seeds	3	3	4	3,5
Fix seed category	1	1	1	0,5
Fix: Always generate the same information with scripts	2	2	3	2,5
Fix: Migration name orders	2	2	3	2,56
Fix: Double import error and error handling after login	2	2	2,5	2,65
Fix: add logo	1	1	1	0.5
Fix: show correct bocados	1	1	1	1
Fix: inputs order in sign up	2	1	2	1
Fix: add button to login and mach the style with the rest of the page	2	1	1,3	1
Fix: remove shopping cart from header in login, home and orders views	1	1	1	1
Fix: change colour of the bar in home view to black	1	1	1	1
Fix: in home view change style of filter	1	1	1	0.5
Fix: in restaurant login view, add an option for the rut of the restaurant	1	1	1	0,5
Sprint 5				

Ilustración 5.10.4: Planilla Excel con estimaciones

Como mencionamos, en los *Sprints* que componían al primer *Release,* optamos por no comenzar a estimar ya que no teníamos la confianza de empezar a hacerlo, y además que no teníamos un histórico para comparar y poder comenzar a estimar el tamaño. Por esto destinamos los primeros *Sprints* como *Sprints* de prueba para poder hacer que nos conozcamos mejor como equipo y conozcamos nuestras capacidades así como la velocidad y productividad. A raíz de este periodo de prueba logramos identificar y delimitar nuestra productividad como equipo.

Por otro lado, en estas reuniones llevábamos a cabo un proceso de análisis de métricas obtenidas de iteraciones anteriores, en donde analizamos los resultados obtenidos para tenerlos en cuenta a futuro, para ver más sobre estas métricas diríjase a la sección 9. Métricas.

Así también, hacíamos un análisis sobre el *feedback* recibido en el *Release* anterior en caso de ser la primera *Sprint Planning* del *Release* actual, para tenerlo en cuenta a la hora de planificar una nueva iteración.

Así mismo, revisábamos todos los *bugs* que surgieron en el *Sprint* anterior, para poder priorizarlos e incorporar a esta nueva iteración los que creyéramos necesarios, además de asignándoles estimaciones de tiempo para su resolución, al igual que con las tareas.

Este enfoque nos garantizo que estábamos abordando las nuevas tareas y características, así como también problemas pendientes del ciclo anterior, asegurando así un flujo de trabajo fluido y eficiente a lo largo del *Sprint*.

Por último, definíamos el objetivo principal de este *Sprint* que describe brevemente qué es lo que se espera del *Sprint*. Fuimos armando planillas de Excel donde registramos todos los resultados obtenidos de estas reuniones. Dentro de esos resultados se encuentran el objetivo, la duración del *Sprint* y la velocidad y esfuerzo estimados, además de cada uno mencionar cuánto esfuerzo máximo prometía cada uno invertir en dicho *Sprint*. En la ilustración 5.10.5 podemos ver el ejemplo del resultado del *Sprint Planning 1*, para ver la planilla completa véase la sección A.20 Resultado de *Sprint* Plannings del Anexo.

Sprint 1	
Fecha Inicio	07/08/2023
Fecha Fin	21/08/2023
Objetivo	Setup de Backend y Frontend. Registro y autenticación de usuarios. Ver menús.
Esfuerzo estimado	-
Velocidad estimada	-

Ilustración 5.10.5: Ejemplo de resultado de Sprint Planning

Ejecución de un Sprint

A continuación nos adentraremos un poco más en el uso de Behavior driven development (BDD), ya que para la ejecución de nuestros *Sprints* seguimos algunas prácticas de dicha metodología. Esta metodología de desarrollo se centra principalmente en cual es el comportamiento esperado del sistema pero desde la

perspectiva del usuario final. Para poder cumplir con dicha perspectiva, dejamos de lado las especificaciones técnicas y elegimos un lenguaje común tanto para la parte técnica como para la parte del negocio. Esto favorece la construcción de criterios de aceptación para los requisitos por lo que nos brindó claridad a todos sobre los mismos. Así mismo, esta no lo cumplimos durante todo el desarrollo, sino que realizamos ciertas variaciones que comentaremos a continuación.

Para llevar a cabo la ejecución del proyecto decidimos utilizar un tablero que nos ayudaba a visualizar los flujos de trabajo y dar seguimiento a la ejecución. Este constaba de siete columnas originalmente dado que después sufrió una pequeña modificación que mencionaremos a continuación.

Estas columnas eran *Sprint backlog, Requirements definition, In progress, In review, Integration testing, Deploy* y *Done.* Este tablero, como mencionamos en la sección de *Delivery,* comenzamos haciéndolo en la herramienta *Click Up,* dado que nos lo habían recomendado y era el utilizado por uno de los integrantes en su trabajo. Al cabo de unos meses, nos parecía que era complejo de manejar y no funcionaba correctamente por lo que decidimos cambiarnos a Jira, una herramienta que todos conocemos por su uso en facultad y en nuestros respectivos trabajos y nos sentimos cómodos usándola. Realizamos toda la migración necesaria de uno a otro para no perder información valiosa como pueden ser reportes. Véase la ilustración 5.10.6 que muestra una pequeña parte del mismo.



Ilustración 5.10.6: Parte del Sprint Product Backlog

En la imagen podemos ver que cada User Story del backlog contaba con su título, un identificador único conformado por el nombre “TUB-” y un número, el integrante del equipo encargado del desarrollo y por último, contaba con un *issue type* que clasifica las historias de usuario en 3 tipos: *Story, Task y Bug*, que como dicen los nombres, clasifica según historias de usuario, tareas y errores. Esto se ve representado por un icono al comienzo de cada línea correspondiente como se puede ver en la ilustración 5.10.7.

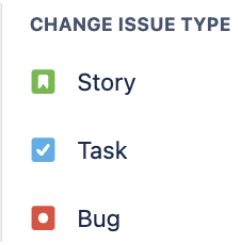


Ilustración 5.10.7: Clasificación del Sprint Backlog

La herramienta *Jira* nos permite hacer la división en *Sprints* de todo el *Backlog* donde le indicamos un conjunto de incidencias, las fechas de inicio y fin y la herramienta misma genera iteraciones del proyecto únicamente con la información específica del mismo, con su número, fechas de inicio y fin y la cantidad de *issues* correspondientes.

Para continuar con la ejecución, hicimos foco en el ciclo de BDD que consta de someter cada requerimiento a los 5 pasos: *Requirements Definition*, *Test Cases Implementation, Application Implementation, Automation Test y Refactoring.* Esto lo utilizamos para guiar la estructura de nuestro tablero ya que son los pasos que debe cumplir cada requerimiento a desarrollar durante el *Sprint*. Estas fueron utilizadas para definir las columnas principales del tablero, a pesar de que realizamos ciertas personalizaciones.

El paso de escribir las pruebas automatizadas lo tomamos como parte del desarrollo, principalmente porque eran desarrolladas por la misma persona que desarrollaba la funcionalidad en sí, por lo que dicho trabajo estaba incluido en la columna de *In Progress*. La columna *Refactoring* no la incluimos ya que la *user story* que debía ser *refactoreada* dado a fallas en las pruebas de integración, volvía a estar en *In Progress* en caso de que las pruebas fallen. Por último, las *automation test* también eran parte de la columna *In Progress* dado que las pruebas unitarias las desarrollábamos ahí y en caso de fallar las pruebas automáticas, volvían a dicha columna. Por último, agregamos una nueva columna denominada *Deploy* que comentaremos a continuación.

Viendo el diagrama en la ilustración 5.10.8, podemos encontrar como definimos inicialmente el tablero y lo que es el flujo normal de una *User Story* a lo largo de su desarrollo. La primera columna es ***Sprint Backlog***, donde se encuentran todas las user stories y posibles *fixes* que existan a realizar en este

Sprint. Estas las hemos extraído del *Product Backlog* en la *Sprint Planning* y las hemos asignado a alguno de nosotros para su implementación durante el *Sprint* en curso.

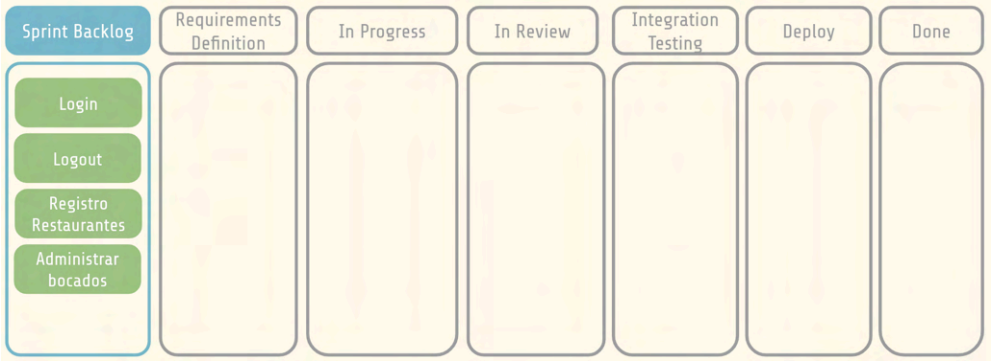


Ilustración 5.10.8: Estado inicial de un Sprint

El primer paso a realizar era tomar una *User Story*, pasarla a la columna ***requirements definition*** y tomar las historias de usuario ya definidas para refinarse en caso de ser necesario.

Para cada una, escribimos los denominados escenarios escritos con el formato "***Given-When-Then***" / "**Dado, Cuando, Entonces**" que constituyen los criterios de aceptación del requerimiento y nos ayudaron a mantener una relación entre lo que era la implementación técnica con los requisitos del negocio. Esto nos llevó a una mejoría en la comprensión de los requisitos del negocio y ayudó a lograr una implementación más precisa y eficiente. Esto lo escribíamos en el tablero de Jira como se puede ver en la ilustración 5.10.11.

Luego utilizamos el **lenguaje Gherkin**, que se basa también en la sintaxis "*Given-When-Then*" para poder definir ciertos pasos en escenarios de prueba, generando un archivo ".feature" que luego utilizaremos.

Luego, esos escenarios los convertimos en especificaciones BDD mediante la herramienta Cucumber, la cual nos ayudó a escribir las pruebas que finalmente verifican el comportamiento del sistema. Esta herramienta analiza los archivos Gherkin ".feature" y ejecuta las pruebas que estos definen, en comparación al sistema que estamos probando, brindándonos resultados claros y certeros sobre el comportamiento del *software*. Para ello, configuramos nuestro entorno de pruebas

para trabajar con Cucumber y escribimos nuestros escenarios de prueba en archivos .feature. Esto permitió automatizar las pruebas para verificar el comportamiento correcto de la aplicación en cada *Pull Request*.

A continuación vemos en la ilustración 5.10.9 un ejemplo de código escrito en Gherkin, el cual es la traducción de los escenarios de la *User Story* y es el código que permitirá llevar a cabo la prueba automática. Cabe destacar que este procedimiento lo llevamos a cabo únicamente para las funcionalidades *backend.*

Gherkin nos ayudó a garantizar que todas las partes involucradas en el desarrollo comprendamos los requerimientos del sistema y lo que hay que esperar del mismo. Por esta razón fue que nos enfocamos en hacer dichas pruebas únicamente para el *backend.* Decidimos mantener el enfoque en el comportamiento del sistema desde la perspectiva del usuario, lo que ayuda a garantizar que las pruebas están alineadas con los requisitos y objetivos del negocio.

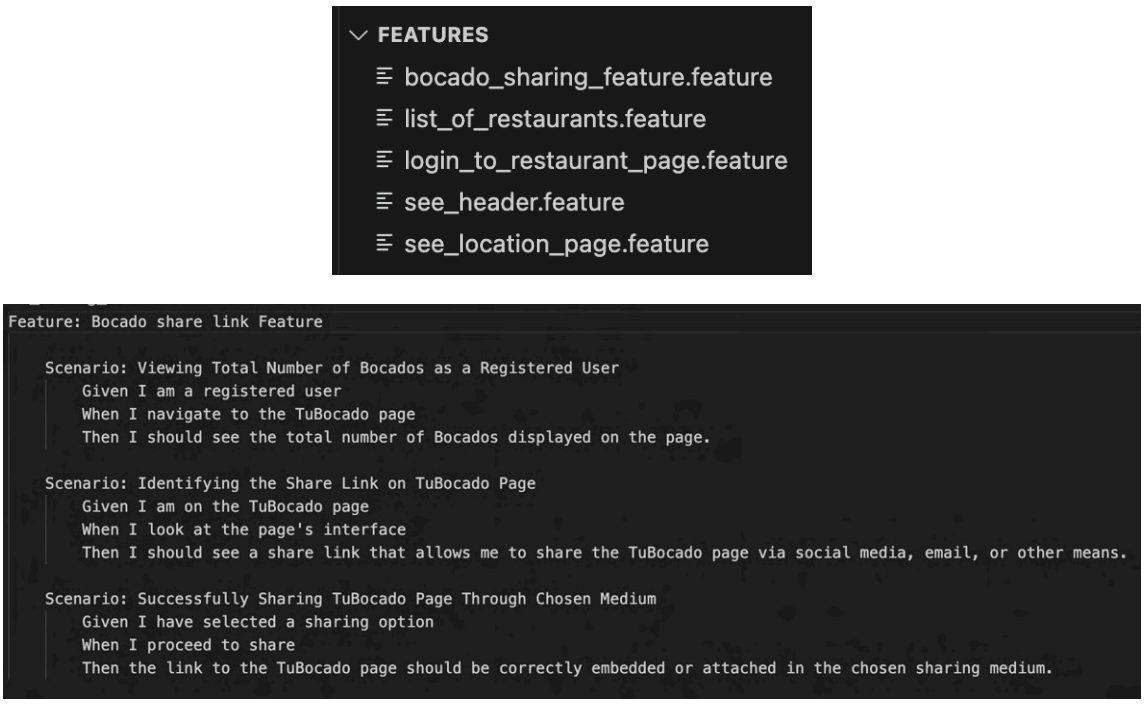


Ilustración 5.10.9: Ejemplo de archivo .feature en lenguaje Gherkin

Dicho proceso lo llevamos a cabo al pie de la letra con lo pautado por BDD únicamente durante el primer *Release*, luego, como mencionamos en la sección 5.5 Criterios de aceptación vs Prototipo, decidimos no escribir los escenarios de los criterios de aceptación y comenzar a utilizar el prototipo para guiar el desarrollo. Al

finalizar el primer *Release*, nos dimos cuenta de que estábamos invirtiendo mucho esfuerzo en escribir los escenarios sabiendo que el proyecto es una POC, por lo que decidimos quitarle complejidad a BDD, y nos basamos en este para seguir la ejecución de los siguientes *Releases* pero no lo ejecutamos al pie de la letra.

Por lo tanto, seguimos utilizando algunas prácticas de la metodología así como también seguimos escribiendo los casos de prueba basados en los escenarios de BDD para que sirvan como guía de las pruebas de aceptación y lograr así que los usuarios puedan probar la aplicación siguiendo un flujo amigable, pero no en lenguaje Gherkin, sino que en español. Por lo tanto, podemos decir que no usamos BDD para toda la ejecución del proyecto dado que no lo seguimos tal cual la metodología indica, pero si hicimos uso de sus prácticas.

En cuanto a las tareas correspondientes al *frontend*, optamos omitir el paso de implementar casos de prueba de comportamiento automatizados desde el comienzo. Lo que hicimos fue utilizar únicamente los escenarios de prueba escritos en lenguaje Gherkin o en Español dependiendo del *Release* para la realización de las pruebas comportamentales de forma manual.

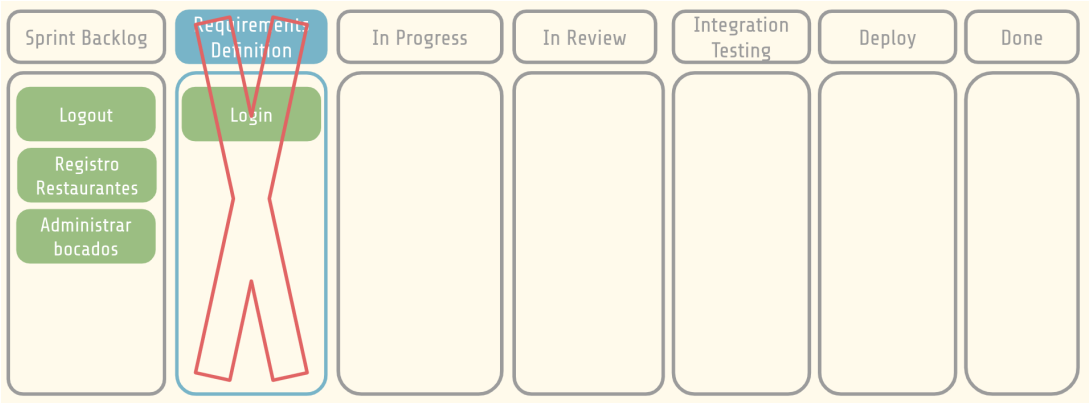


Ilustración 5.10.10: Etapa de Requirements Definition

Por lo tanto, como se muestra en la ilustración 5.10.10, al tablero inicial le quitamos la segunda columna de *Requirement Definition*.

A continuación, presentaremos cómo se ve una historia de usuario después de avanzar por la columna de *Requirement Definition*. Observaremos que en esta ilustración, la historia de usuario ya incluye los tiempos estimados y reales del esfuerzo, esto es porque estamos mostrando una historia existente como ejemplo

que ya pasó por todas las columnas. Sin embargo, es importante destacar que, en realidad, cuando una historia la movemos de la columna *In Progress,* sólo dispone de los valores estimados y los esfuerzos invertidos hasta la etapa de *Requirement Definition*.



Ilustración 5.10.11: Definición de requerimiento y escenarios en Jira

Luego pasamos a la columna *In Progress* en donde podemos ver que la *user story* la dividimos en sub *user stories* según sus componentes, es decir, entre *backend* y *frontend*. Cabe destacar que nos basamos en el principio de ***One Piece Flow*** para la ejecución, lo que significa que hasta que no terminamos un requerimiento, no comenzábamos con el siguiente, incluyendo *backend* y *frontend*. Por lo que avanzan juntas, no las tomamos de forma independientes durante el flujo pero así todo las dividimos en dos para poder ser realizadas por diferentes desarrolladores. De esta forma además, evitamos dejar mucho trabajo incompleto en simultáneo y mantuvimos una clara asignación de responsabilidades. Podemos ver dicha división en la ilustración 5.10.12.

En esta etapa realizamos la construcción del código funcional correspondiente al requerimiento, es decir a los escenarios y casos de prueba especificados para el mismo. Además, creamos las pruebas unitarias para los módulos del *backend* utilizando *Jest* para probar todas las funcionalidades desarrolladas. Mencionaremos más sobre estas pruebas en la sección 8. Aseguramiento de calidad.

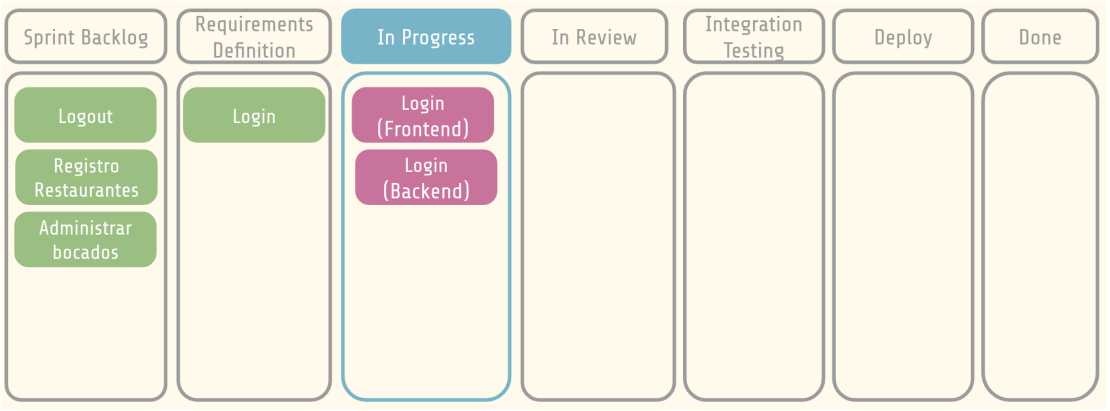


Ilustración 5.10.12: Etapa de *In Progress*

Una vez que el desarrollo estaba terminado, pasamos a la columna de *In Review*, donde realizábamos el pull request en GitHub de todas las sub historias realizadas y alguno de los demás integrantes revisaba lo realizado y hacía comentarios al respecto en caso de ser necesario. De ser así, el responsable de dicha implementación debía corregir lo planteado por su compañero hasta terminar siendo aprobado. Podemos ver dicho paso en la ilustración 5.10.13. Una vez que hacìamos el *pull request,* corríamos todas las pruebas unitarias creadas hasta el momento de forma automáticas, por lo que nuevamente probábamos todas las funcionalidades. Una vez que todas las pruebas hayan pasado y las hayamos aprobado todos los *pull requests* necesarios para la historia de usuario, el desarrollador procedía a *mergear* estos cambios a la rama *development*. Para ahondar más en el uso de *pull requests*, véase la sección 9.5 Estándares, particularmente Estándares de proceso de desarrollo.

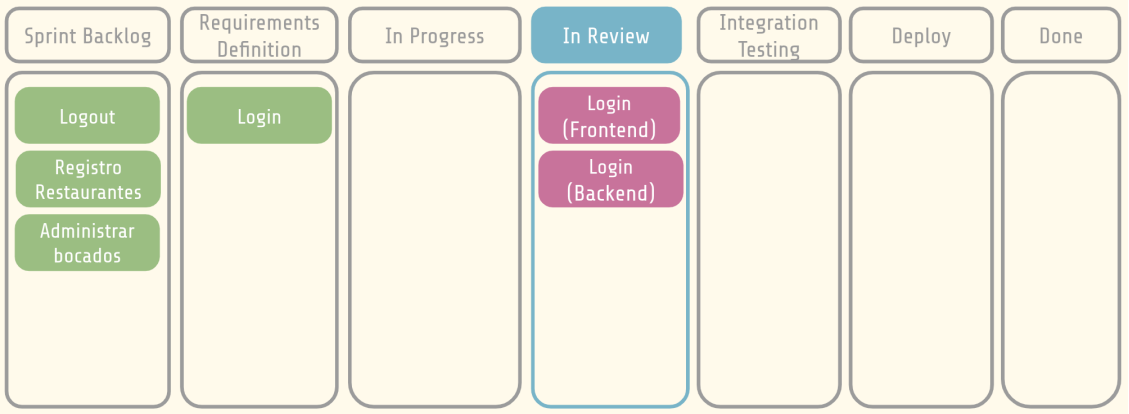


Ilustración 5.10.13: Etapa de In *Review*

Después de la revisión, como se puede ver en la ilustración 5.10.14, las user stories pasan a la columna *Integration Testing* para ser sometidas a pruebas de integración ejecutadas a mano por algún integrante. Aquí toda la nueva funcionalidad implementada ya la podemos encontrar en la rama *development* y se encuentra en conjunto con todo el resto de las funcionalidades por lo que verificamos que las nuevas implementaciones funcionen adecuadamente en conjunto con otras partes del sistema antes de ser incorporadas a la rama *main*. Asimismo, evaluamos que el *backend* y el *frontend* interactúan de forma correcta y no comprometen el resto del sistema. Podemos ver más información acerca de estas pruebas en la sección 9. Aseguramiento de la calidad, particularmente en la sección pruebas de integración. Hasta no lograr pasar dichas pruebas en su totalidad, las user stories volvían a estar en *In Progress*, repitiendo el procedimiento nuevamente hasta finalmente pasar las pruebas con éxito.

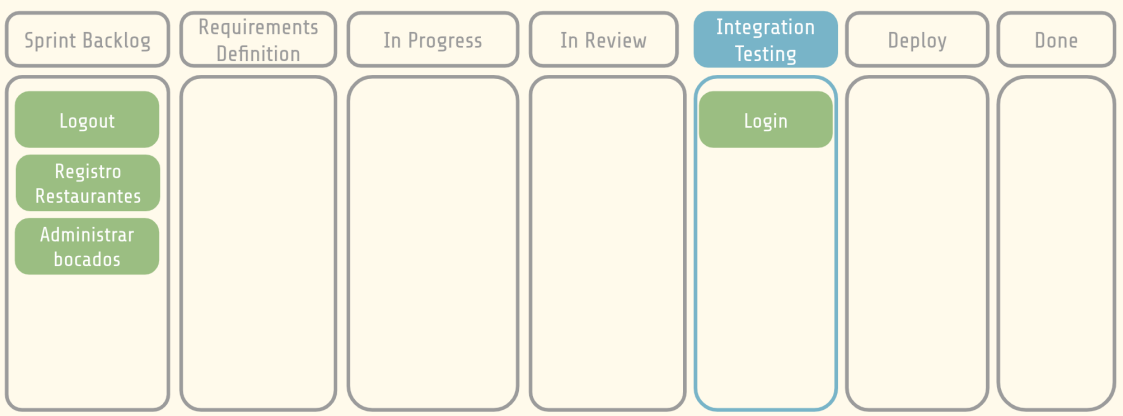


Ilustración 5.10.14: Etapa de *Integration Testing*

Una vez que han superado con éxito las pruebas de integración, podemos ver en la ilustración 5.10.15 que pasan a la columna Deploy, donde las preparamos y posteriormente, desplegamos en el entorno de producción. En nuestro caso, al no tener un ambiente de desarrollo *deployado*, las historias de usuario permanecen en la columna Deploy hasta finalizar el *Release*.

Al finalizar el último sprint de un *Release*, con todas las funcionalidades terminadas y probadas, realizábamos nuevamente un *pull request* con todos los cambios generales en donde volvíamos a correr las pruebas automáticas correspondientes y en caso de que las mismas fallen volvía al desarrollo. Por otro lado, en caso de que todas las pruebas sean aprobadas, procedíamos a realizar una

nueva ronda de pruebas de integración manual, llevada a cabo por otro miembro del equipo. El objetivo de esto era asegurar que el código listo para ser desplegado en producción no provocará ningún problema o inestabilidad en la misma. Si los resultados de las mismas eran exitosos, el miembro que realizaba la revisión procedía a aprobar el *pull request* y hacíamos un merge a main, con su respectivo tag de *Release* y la versión correspondiente en GitHub.

Luego, si todo salía correctamente, pasábamos todas las historias de usuario de todos los sprints correspondientes al *Release* a la columna *Done*. Nosotros utilizamos despliegue continuo, es decir, con cada *merge* a *main* se realiza automáticamente un *deploy*, con todo lo que esta automatización conlleva. A continuación mencionaremos como es dicho proceso de deployment y lo que incluye.

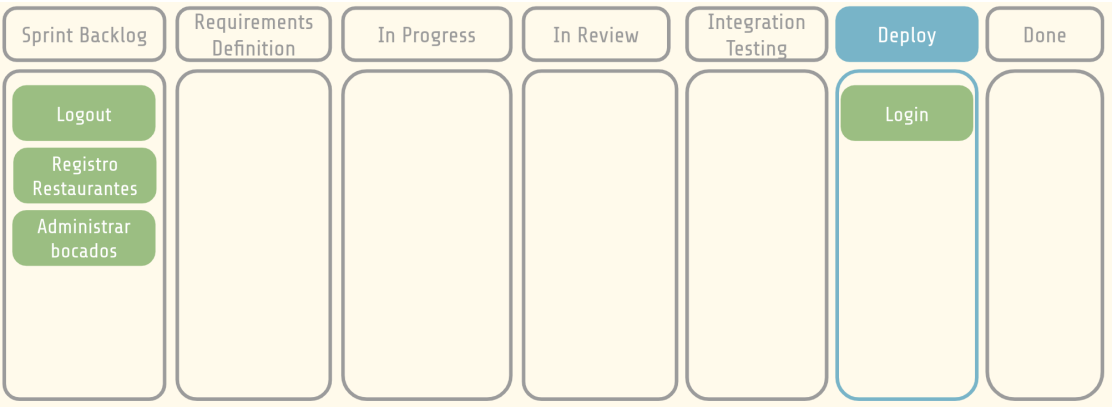


Ilustración 5.10.15: Etapa de *Deploy*

Proceso de deployment

El primer paso de dicho proceso es el *pull request* para hacer el *merge* a la rama *main*. Con este comienzan las *Github Actions* en el caso del *backend*, lo cual es un servicio de integración continua que permite automatizar tareas específicas en un repositorio de GitHub. Dichas tareas son la compilación del código, las pruebas unitarias de forma automatizada, denominado integración continua, el despliegue a entornos de producción y las notificaciones y alertas. Luego, en el caso del *Backend*, gracias a la utilización de *Pipeline* de AWS que está conectado directamente con el repositorio, al realizar el merge empieza a correr el proceso de deployment del lado de AWS. Aca automáticamente se genera la imagen en docker, se sube la imagen de Docker a Amazon Elastic Beanstalk y se finaliza con la

actualización del cluster de ECS con la nueva imagen (esto lo hace Elastic Beanstalk automáticamente).

Por otro lado, para el caso del *frontend*, utilizamos AWS Amplify por lo que conectamos la rama *main* a AMplify lo que hacía que cada vez que exista un merge a *main* se deploye automáticamente.

Por último, volviendo a la ejecución, una vez que cumplíamos con la *definition of done*, que comentaré a continuación sobre la misma, pasábamos a la columna *Done* que podemos ver en la ilustración 5.10.16. la etapa final del proceso, donde encontramos las tareas y *user stories* completadas. Consideramos terminado y cerramos formalmente el ciclo del *Sprint* actual, así como también del *Release*, dado que llegan a *Done* una vez que está todo desplegado en *main* y dicho proceso lo realizábamos únicamente al finalizar un *Release*.

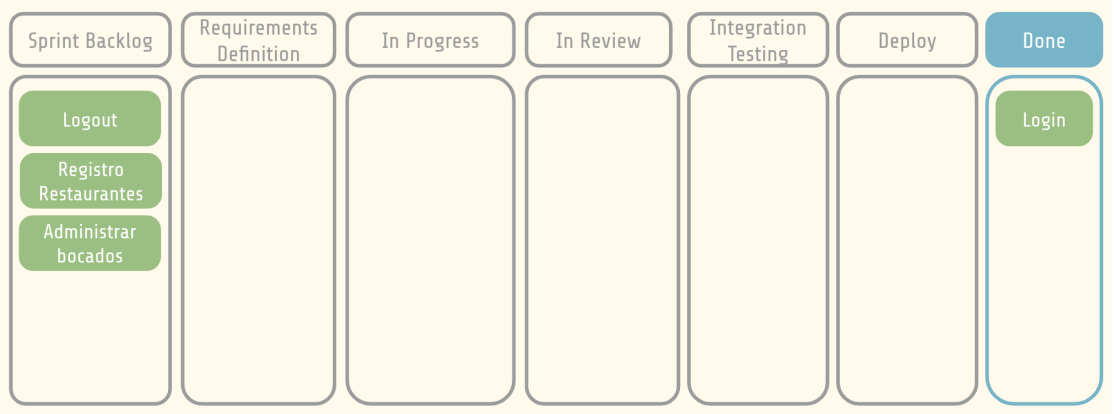


Ilustración 5.10.16: Etapa final de una *user story*

Definition of Done

Como ya mencionamos en la sección de Proceso de desarrollo, la *Definition of Done* es justamente la lista de características que debe cumplir el producto que estamos desarrollando para que podamos clasificarlo como "*Done*". Antes de comenzar el primer *Sprint*, en la primera *Release Planning Meeting*, llevamos a cabo dicha lista, que luego sufriría ciertas modificaciones al correr de los *Releases* restantes.

Al comienzo del proyecto, nos enfrentamos a la decisión de si establecer la *Definition of Done* por cada *Sprint* o por cada *Release*. Al comienzo optamos por hacerla por *Release* por una serie de factores que comentaremos a continuación. El

primer *Release* lo tomamos de prueba para ver si nos era suficiente no ahondar en cada *Sprint*, como nos sentimos cómodos con este enfoque, decidimos mantenerlo.

En primer lugar, al tener el proyecto una duración limitada y un conjunto de entregas ya definido, nos pareció más adecuado que la DoD sea por *Release*, dado que sería más eficiente y relevante para el equipo. Además, como ya mencionamos, las entregas las realizábamos al finalizar cada *Release,* siendo estas MVPs, por lo que el valor del negocio no estaba asociado a cada *Sprint*, sino que derivaba de la entrega de un conjunto completo de características al finalizar cada *Release.* Por último, buscamos optimizar nuestros procesos y simplificar la gestión de calidad del producto para poder ser más eficientes. Establecer la DoD por *Release* nos permitió tener criterios más centrados en la entrega del valor del negocio, en lugar de criterios granulares.

Al finalizar el proyecto nos sentimos conformes con la decisión que tomamos, a pesar de que haberla creado por *Sprint* tenía sus ventajas, ya que es más ordenado y podríamos hacer un seguimiento más detallado, eso tenía un costo que nos favoreció no haberlo pagado.

Una vez decidido esto, pasamos a especificar cada punto de la lista, para eso seguimos algunos pasos importantes. En primer lugar, tuvimos que identificar las necesidades y expectativas del cliente y del usuario final. Comprender estas nos ayudó a establecer criterios realistas y centrados en el valor del negocio.

Así mismo, fue esencial que todos los miembros del equipo participáramos de la creación de la DoD, incluido el *product owner.* Cada miembro aportó sus perspectivas sobre lo que interpreta por “terminado”, específicamente desde la posición de su rol.

Otro factor importante fue identificar los elementos clave que debíamos cumplir para considerar que un *Release* estaba terminado, como por ejemplo en nuestro caso, que todo el código esté fusionado en *main* y en ambiente de producción, pruebas unitarias completadas, aceptación por parte del cliente y demás.

Para continuar, debimos definir criterios específicos que indiquen cuando se cumple cada elemento clave, por ejemplo, indicar cual es el porcentaje de cobertura de pruebas unitarias al código mínimo aceptable. Este punto también está ligado con que debimos establecer estándares de calidad. La DoD debía reflejar estándares de calidad de código y de organización, como por ejemplo la inclusión de requisitos de seguridad y rendimiento.

Continuando con la composición del tablero, dada la longitud del proyecto fue de suma importancia gestionar de forma correcta los riesgos para poder tenerlos bajo control y que no impacten de forma negativa en nuestro desarrollo. Para ello, en cada ceremonia de *Sprint Planning*, seleccionábamos los riesgos que podrían afectar al *Sprint* y los alojábamos en un *Risk Sprint Backlog*. Para tener visible dicho *Backlog* en cada *Sprint*, en el tablero decidimos agregar una columna llamada *Sprint Risks*. Podemos ver en la ilustración 5.10.17 el contenido de dicha columna del tablero, para más información sobre la gestión de riesgos, véase la sección 6. Gestión de riesgos.



Ilustración 5.10.17: *Sprint Risks* representados en el tablero

Para finalizar, en relación a los ***fixes*** mencionados, comentaremos a continuación una serie de pasos que llevábamos a cabo para manejar las correcciones que debimos hacer durante el desarrollo.

En primer lugar estaba la identificación y el registro de los errores. Todos los errores encontrados debían ser registrados como un elemento del *product backlog,* estos errores podrían ser por parte de nuestro equipo de desarrollo, provenientes del *feedback* de los prototipos y de las pruebas con usuarios. Este elemento en el *backlog* debe describir el error de la forma más clara y precisa que pudiésemos, en conjunto con información adicional que sea relevante para la descripción, como podría ser la gravedad o el impacto del mismo en el sistema.

Como siguiente paso estaba la priorización de la corrección de dicho error, esta la llevábamos a cabo generalmente nosotros mismos, por más que algunos eran discutidos con el *product owner* en caso de creerlo necesario. En algunos casos era necesario que el *product owner* lo priorice en relación a otras tareas y funcionalidades, donde este debía evaluar el impacto del error en el producto y la urgencia del mismo frente a otras necesidades del negocio.

En tercer lugar, incluíamos la corrección de dicho error en el *Sprint Backlog.* Al momento de planificar un nuevo *Sprint*, seleccionamos las historias de usuario a realizarse en el *sprint* a comenzar, así como también los errores a corregir en el mismo. Estos los seleccionábamos siempre y cuando hayan sido priorizados por sobre otras historias de usuario.

A continuación, llevábamos a cabo el proceso de desarrollar los *fixes* dentro de la iteración. Estos cumplían con todo el flujo del tablero mencionado, excepto *Requirements Definition*, aún cuando la utilizabamos.

Como último paso, hacíamos una revisión para asegurarnos de que el error lo habíamos corregido correctamente y que además, dicha corrección no introdujo nuevos errores. Esta revisión en ciertos casos era partícipe el *product owner* si lo creíamos necesarios. Una vez completada dicha revisión, nuevamente tenía el mismo comportamiento que una historia de usuario, seguía el flujo del *Sprint.*

En último lugar, en el *backlog* además de historias de usuario y *fixes* podíamos encontrar tareas, estas las escribimos siguiendo el formato de *user stories* solo que comenzaba con *As a developer*. Algunas tareas eran hacer el *setup* del *backend* y *frontend* al comienzo del proyecto, configurar la base de datos, correr *Migrations*, entre otras. Un ejemplo de una tarea en el tablero se puede ver en la ilustración 5.10.18. Estas no pasaban por todas las columnas, ya que por ejemplo, no requieren pruebas unitarias ni de integración. Por esta razón, estas tareas pasaban del *backlog* a *In progress* y de ahí directamente a *Done* cuando estaba completada.

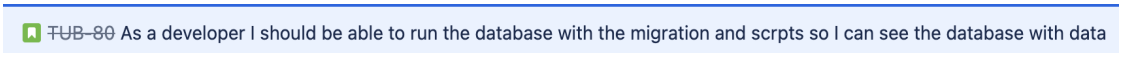


Ilustración 5.10.18: Ejemplo de tarea en el tablero

Sprint Review

Al finalizar el desarrollo de todo un *Sprint*, llevábamos a cabo la reunión denominada *Sprint Review*, como hemos mencionado, en esta reunión participaban todos los integrantes pero no así el *product owner*.

Comenzábamos haciendo un resumen de todo lo realizado durante el *Sprint* en cuestión, comentando detalladamente el valor que estábamos agregando al producto en esta iteración.

En el caso de los primeros 4 *Sprints* hicimos una revisión de los criterios de aceptación pautados al comienzo del mismo, en los siguientes *Sprints* revisamos el prototipo para corroborar que estemos cumpliendo con lo pautado.

Además, discutíamos problemas que hayan surgido durante toda la iteración y cómo hicimos para resolverlos. Esto incluye obstáculos que hayamos enfrentado y las soluciones que implementamos. Relacionado con este punto, entre los integrantes nos brindábamos *feedback* sobre el trabajo realizado, incluyendo sugerencias de mejora y reconocimientos por el trabajo bien hecho. Hacíamos una revisión de todo el progreso que hemos tenido, teniendo como base los objetivos planteados al comienzo del proyecto. Analizando cómo el trabajo realizado durante toda la iteración en cuestión contribuyó al progreso general hacia los objetivos del proyecto.

Durante la ejecución del *Sprint* registrábamos todos los datos necesarios para luego utilizarlos en las métricas, véase el capítulo 9. Métricas para más detalle. En esta reunión además, verificábamos el incremento creado durante el *Sprint*.

Por último, hacíamos una revisión de las estimaciones de todas las historias de usuario, ya que a medida que íbamos realizando el desarrollo, cada uno era responsable de agregar cuantas horas le había llevado llevarlo a cabo. Por ende, al finalizar, teníamos en la planilla las horas reales, con este nuevo dato podíamos

obtener cuál era el nuevo valor de *Story Points*-horas, promediando con el de *Sprints* anteriores.

Release Review

La *Release Review* la realizábamos al terminar cada release. Estas reuniones las hacíamos en conjunto con las *Sprint Reviews* en caso de que coincidan, es decir, en la *Review* del último *Sprint* realizábamos en conjunto con la *Review* del *Release.* Sin embargo, de esta reunión era partícipe el *product owner* por lo que además de lo mencionado, hacíamos una demostración sobre los avances al mismo. El tipo de demostración variaba según las funcionalidades que queríamos mostrar. Siempre arrancábamos con una breve demostración de las funcionalidades incluidas, luego variaba según si las funcionalidades eran más simples o no, ya que en el caso de ser simples el *product owner* interactuaba de forma libre y en los otros casos necesitaba ser guiado. Las funcionalidades más complejas o características de este sistema en sí, necesitaban de una explicación o una demostración previa para que se pueda entender. Al final la demostración dejabamos un espacio para cualquier duda que surja y además, un espacio para que pruebe libremente en caso de ser las funcionalidades que deben ser probadas de forma guiada.

Para estas reuniones hacíamos uso de la herramienta *Google Meets* a modo de poder utilizar funcionalidades como compartir pantalla en tiempo real, las cuales eran muy útiles para mostrar lo deseado. Cabe destacar que para estas demostraciones hacíamos uso del sistema desplegado en producción, una vez deployado le enviábamos los *links* al *product owner* para que pruebe la versión final hasta el momento para que pueda realizar las pruebas de aceptación para usuarios y así, hacernos un devolución final del sistema.

Sprint Retrospective Meeting

Luego de entregarle un *Release* al cliente y obtener su aprobación, sus comentarios y sus sugerencias llevábamos a cabo una retrospectiva, en la que tratamos la validación y análisis de las métricas, los riesgos y destacamos las lecciones aprendidas.

Hicimos uso de la herramienta Metro Retro donde creamos tableros DAKI (*Drop, Add, Keep, Improve*) que consisten en el armado organizado en categorías y la obtención de acciones específicas a raíz de ellas, que surgían luego de analizar lo identificado en estas 4 categorías.. Cabe destacar, que antes de llevar a cabo dicha ceremonia, actualizábamos las métricas para analizar el desarrollo, para que, al llevar a cabo la retro, ya tengamos todo lo necesario para poder discutir sobre los valores obtenidos.

Siempre teníamos un anfitrión en esa retro que era el encargado de realizar las anotaciones correspondientes, algo importante a destacar es que dicho rol iba rotando entre los cuatro integrantes del equipo. En la ilustración 5.10.19 podemos ver el ejemplo del resultado de una retro.

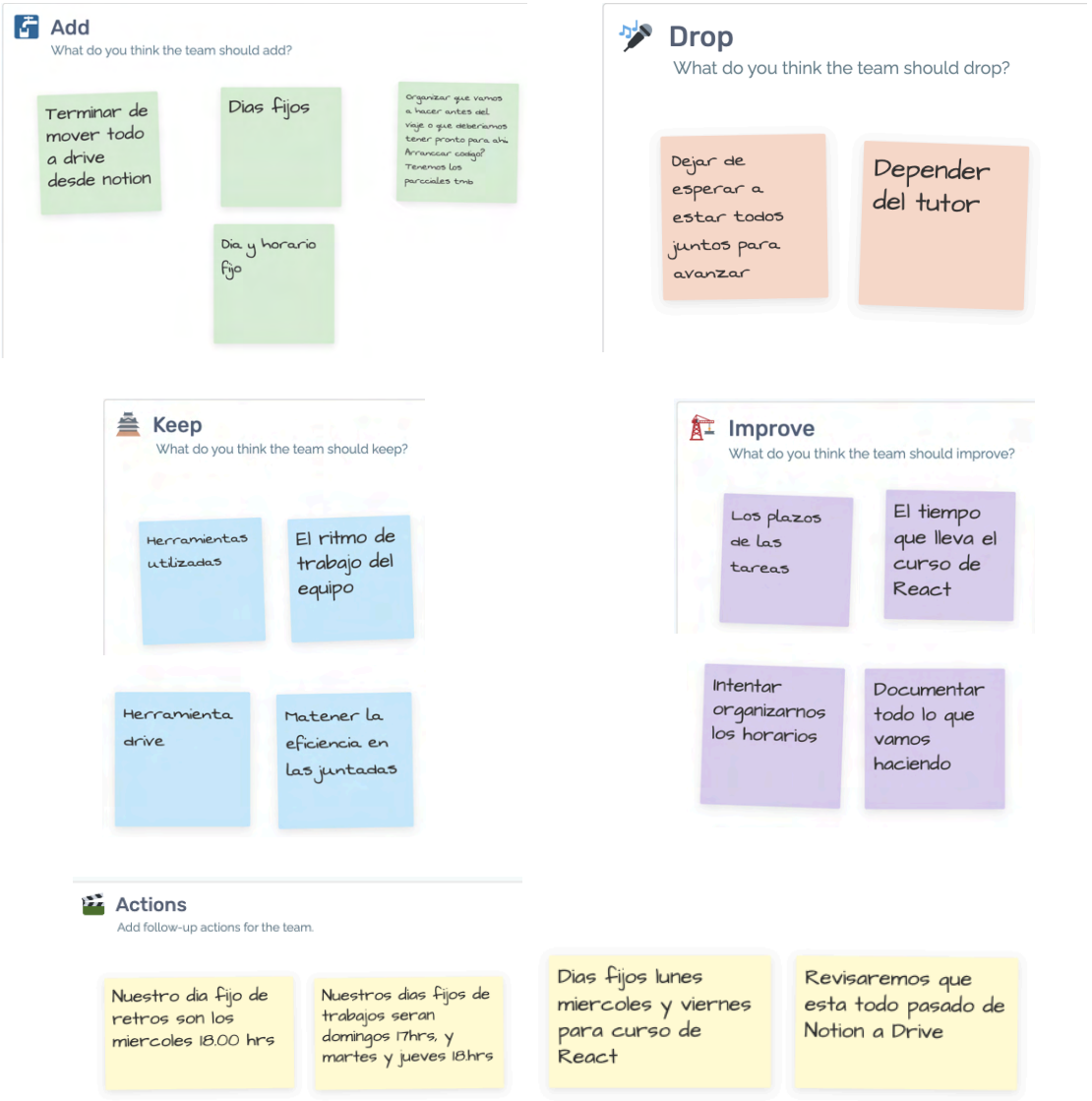


Ilustración 5.10.19: Ejemplo de tablero en Retro Metro

Tras concluir las discusiones y la generación de los *action items* pertinentes, dábamos por finalizada la reunión de retrospectiva. El anfitrión de dicha reunión, tenía la responsabilidad de elaborar un documento resumen, el cual incluye los puntos clave discutidos, así como el enlace a *Metro Retro* donde llevábamos a cabo la sesión. Dicho documento podemos verlo en la ilustración 5.10.20, en conjunto con una parte de la sección del drive que incluye todos los documentos. Este informe lo guardábamos posteriormente en el Drive del proyecto, junto con los registros de todas las retrospectivas anteriores, asegurando un acceso fácil y organizado a la información relevante para nosotros.

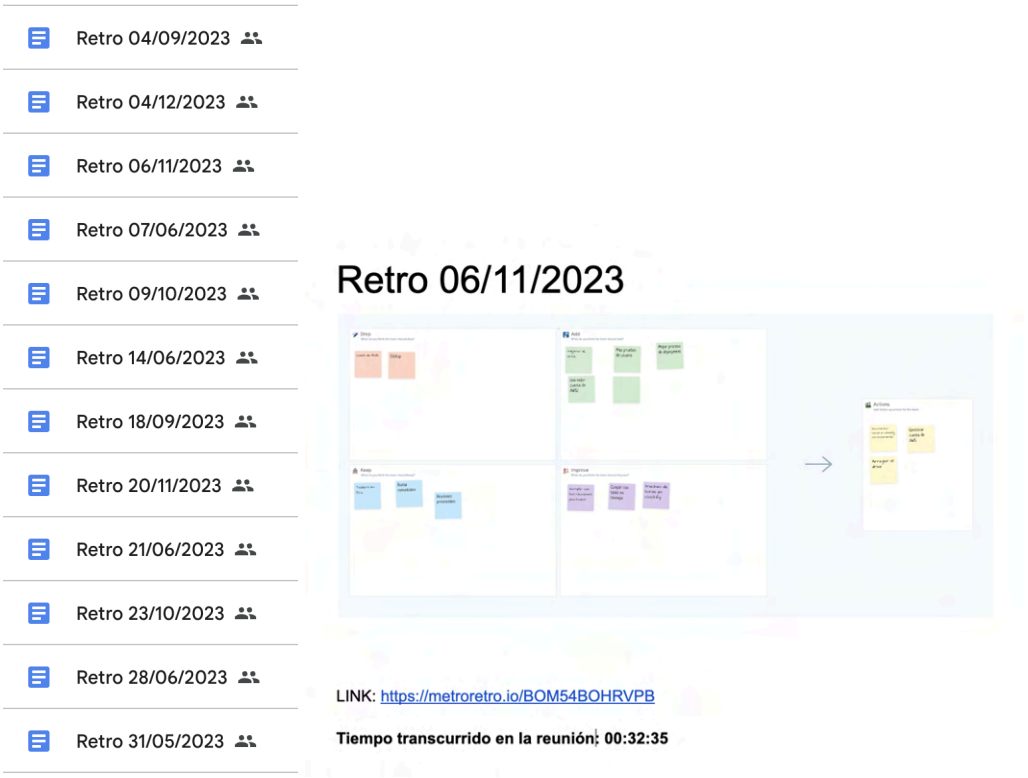


Ilustración 5.10.20: Ejemplo de documento en Drive

Por último, tenemos el *Release Retrospective*, el cual es un paso muy importante para nosotros ya que nos permite reflexionar sobre lo hecho, analizar los problemas y sus causas, de manera de corregir en los casos que es necesario y evitar continuar cometiendo errores. Estas retrospectivas son las mismas que las retros hechas en Retro Metro en el último *Sprint* de cada *Release*.

5.11. Gitflow

Continuando con el proceso de desarrollo, a continuación comentaremos acerca de *GitFlow*, un modelo de ramificación en Git que utilizamos para el desarrollo del proyecto. Este consta de dos ramas principales llamadas *main* y *develop*, que facilita la colaboración y controla el flujo de trabajo. Usamos además ramas *feature* para nuevas funcionalidades, *Release* para preparar versiones y *fix* para correcciones. Podemos ver más detalle de las ramas en la sección 7. Gestión de la configuración.

La forma de trabajo constaba en crear un *pull request* por cada *merge* a *develop* que quisiéramos realizar, el cual debía tener todos los cambios que quisiéramos incorporar aprobados por un desarrollador distinto al cual realizó el PR para ser *mergeado*. Esta forma de trabajo nos facilitó a estructurar nuestro desarrollo, asegurando así la calidad, eficiencia y velocidad de nuestro equipo.

Elegimos *Gitflow* porque creímos que es mejor cuando se necesita una estructura de ramas clara y un control riguroso del flujo de trabajo. Así todo, investigamos *Trunk Based Development* ya que nuestro tutor lo sugirió y además porque es adecuado para proyectos ágiles y pequeños, ya que prioriza la simplicidad. En este modelo se trabaja principalmente en una única rama, llamada *Trunk*, por lo que los cambios se hacen todos en dicha rama principal. Esto facilita los despliegues continuos ya que al estar todos los cambios en la misma rama, se puede desplegar rápidamente. Por último, promueve la integración continua y la comunicación fluida.

Sin embargo, creímos que el uso de las ramas de *GitFlow* nos sería muy útil dado que como equipo nos sentíamos muy cómodos con su uso y es lo que estamos acostumbrados a utilizar. El hecho de poder separar el desarrollo de diferentes funcionalidades en ramas separadas permite mayor aislamiento y facilita la colaboración en características específicas. Así mismo, *GitFlow* nos brinda una estructura para poder gestionar las diferentes versiones del sistema a desarrollar de forma controlada, dado que podemos hacer uso de ramas específicas para preparar las versiones y corregir errores. Por último, tuvimos en cuenta la popularidad de *GitFlow* y la compatibilidad que tiene con las herramientas que

utilizamos, por lo que sumando todos estos factores, decidimos mantenernos con este modelo.

5.12. *Release Plan*

El *Release Plan* lo diseñamos teniendo en cuenta el ciclo de vida mencionado. Dentro de cada uno definimos de forma primaria los MVPs (*Minimum Viable Product*) con las historias de usuario que deberían ser entregadas, ya que al finalizar cada *Release* realizábamos un despliegue a producción. Además, el *Release* 4 (último) decidimos dejarlo sin nuevas funcionalidades sino que para corrección de *bugs* y mejoras de usabilidad. Optamos por un enfoque de MVP para cada *Release* del sistema debido a nuestra prioridad en la validación del concepto y la iteración constante para ajustar el producto según la retroalimentación de los usuarios y del cliente. Esto significa que estuvimos enfocados en lanzar versiones tempranas del sistema con un conjunto básico de características para obtener retroalimentación valiosa. Esto nos permitió ser ágiles y nos aseguraba de que nuestro producto final estaba encaminado para satisfacer las necesidades del mercado y del cliente.

En relación a esto además, utilizamos la integración de principios de *Lean Startup* que implica una orientación hacia la identificación del mínimo viable necesario en cada *Release*. Aplicar *Lean Startup* nos facilitó la creación de MVPs para validar y priorizar ideas de negocio en cada etapa, asegurando que incorporábamos únicamente lo esencial para obtener un resultado útil. Utilizar este enfoque nos guió en la definición de metodologías, permitiéndonos discernir de manera efectiva la viabilidad y la utilidad de cada parte, contribuyendo a una estrategia de lanzamiento más ágil y eficiente.

La aplicación de *Lean Startup* comenzó decidiendo enfocarnos en construir el mínimo producto viable que pudiese abordar el problema, en vez de trabajar en una versión completa. Esto quiere decir que nos basamos en desarrollar solo las características esenciales necesarias para validar la viabilidad y utilidad de la solución. Otro paso importante fueron las pruebas y validaciones realizadas, ya que una vez desarrollado el MVP correspondiente, sometimos a pruebas exhaustivas con diferentes usuarios objetivos para recibir *feedback* y poder validar la solución.

Como tercer y último paso, basamos las iteraciones en la retroalimentación recibida. En base a los resultados obtenidos en las pruebas con usuarios y PO, realizamos iteraciones en el MVP para poder mejorar y refinar la solución. El ciclo de construir, medir y aprender lo repetimos reiteradas veces hasta validar la solución por completo y encontrar el producto que satisface las necesidades de usuarios y PO.

Todo esto tiene el fin de proporcionar un lanzamiento ágil, ya que mediante la validación de la solución y la incorporación del *feedback* apuntamos a que el cliente pueda lanzar el producto completo. Al hacer uso de los MVPs para validar las etapas del proceso, el riesgo de desarrollar características que los usuarios no necesitan o no valoran es ampliamente minimizado.

Primero, teniendo en cuenta la duración de cada *Release* y de cada *Sprint*, la duración del proyecto en general y el alcance del proyecto, decidimos dividir cada iteración en las fechas que se muestran en la ilustración 5.12.1.



Ilustración 5.12.1: Cronograma inicial de Delivery

Teniendo todo esto en cuenta, y debido a ciertos retrasos que comentaremos luego, decidimos que a nivel general los MVPs de cada *Release* y las fechas de principio y fin de cada uno deberían ser como se muestran en la ilustración 5.12.2.

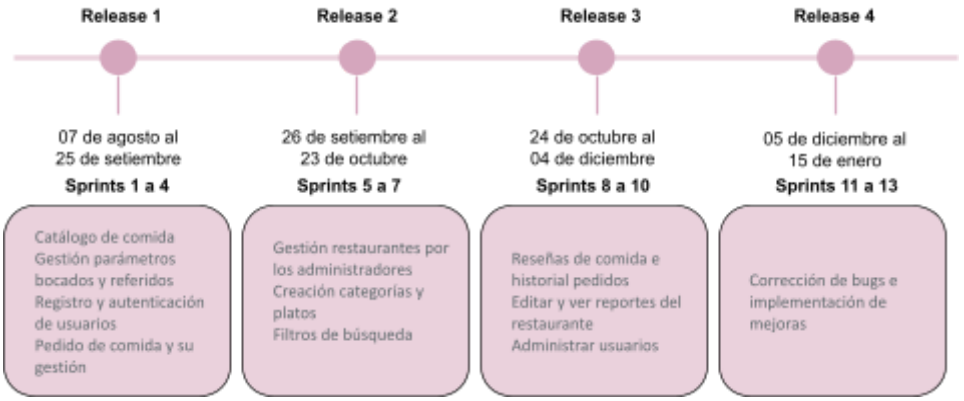


Ilustración 5.12.2: Cronograma final de Delivery

Para ver todos los requerimientos que implementamos en el sistema véase la sección 3.1. Requerimientos.

Cabe destacar que el primer *Release* fue un periodo de adaptación y aprendizaje para nosotros, dado que estábamos aprendiendo las herramientas, además de adaptarnos al trabajo en equipo. Es por esto que este *Release* no cuenta con ninguna funcionalidad muy compleja, además de no cumplir con algunos requerimientos no funcionales como performance dado que el enfoque no estaba puesto en esos puntos. A partir del *Release* 2 en adelante sumamos las funcionalidades complejas, mejoramos las ya implementadas y tuvimos muy en cuenta todos los requerimientos no funcionales que podemos encontrar en 3.1. Requerimientos, específicamente en la sección de requerimientos no funcionales.

Originalmente, el plan era comenzar el primer *Release* el 31 de julio pero llegado el momento decidimos postergar una semana dado que dos integrantes tuvieron que resolver situaciones en trabajo y facultad y no tenían disponibilidad horaria. Algo similar sucedió en el *Release* 1 que como ya mencionamos, vimos necesario sumar un cuarto *Sprint* de una semana para poder llegar a cumplir con la *Definition of Done*, véase la sección de Ejecución, específicamente en *Definition of done* para mayor profundidad. Estas situaciones sabíamos que podían suceder, se explica un poco más acerca de este riesgo en la sección 6. Gestión de Riesgos. Más allá de estos dos imprevistos, el *Release Plan* logramos seguirlo sin inconvenientes.

6. Gestión de riesgos

En este capítulo mostramos cómo llevamos a cabo la gestión de los riesgos durante el proyecto, esto es, el objetivo que nos presentamos, el proceso que utilizamos, el impacto y el resultado obtenido de los riesgos que consideramos más importantes. A su vez, mostramos cómo las metodologías proactivas que utilizamos disminuyeron el impacto y la probabilidad de eventos negativos.

6.1. Identificación de riesgos

La planificación de la gestión de los riesgos durante el desarrollo del proyecto, fue uno de los principales temas que debatimos en las reuniones de planificación de cada iteración. Lo primero que acordamos fue el formato sobre el cual especificar los riesgos. Para que la descripción de los riesgos no aparezca como ambigua, definimos que los riesgos deben contener los siguientes atributos:

- **Nombre:** breve enunciado que identifica y representa la naturaleza del riesgo
- **Contexto:** una contextualización del riesgo, con el siguiente formato:

  Dado<un contexto> (precondiciones que se cumplen en el ámbito del proyecto para que aparezca el riesgo), **Cuándo<evento>** (acción o evento que realicemos durante el proyecto), **Entonces<consecuencia>** (resultado de la acción o evento. Puede impactar sobre alguna, algunas o todas las restricciones de un proyecto como el alcance, la calidad, la duración, el costo y/o la satisfacción del cliente).
- **Acción preventiva:** posibles acciones a tomar para reducir la posibilidad de que el riesgo ocurra.
- **Acción correctiva:** posibles acciones a tomar para reducir el impacto luego de que el riesgo ocurrió y se convirtió en problema.

A continuación mostramos un ejemplo de cómo aplicamos este formato para identificar los riesgos dentro del proyecto. Se utilizó este formato para todos los riesgos. En el anexo se encuentra la lista de todos los riesgos identificados.

ID	Título	Contexto	Acción Preventiva	Acción Correctiva
R1	Falta de tiempo del equipo (estudio, trabajo y viajes)	**Dado** que el equipo no pueda dedicar el suficiente tiempo y esfuerzo al proyecto debido a otras responsabilidades como estudio de otras materias, actividad laboral de cada uno y viajes al exterior. **Cuando** se desarrolla y ejecuta el proyecto. **Entonces** puede haber retrasos en la entrega del producto, disminución de la productividad y afectación de la calidad del producto final o puede que se deba recortar el alcance.	Establecer un plan de trabajo claro, asignar responsabilidades específicas y fomentar la comunicación abierta para mantener la motivación y el compromiso del equipo.	Afectar la calidad del producto o recortar el alcance en caso de que se detecte una falta de dedicación por parte del equipo.

Tabla 6.1.1: Ejemplo de aplicación del formato para riesgos

6.2. Estrategia utilizada

En la primera etapa del proceso, llevamos a cabo la identificación de riesgos, donde registramos en un documento utilizando el formato presentado. Una vez identificados los riesgos, realizamos un análisis y evaluación, teniendo en cuenta su impacto y probabilidad de ocurrencia. Luego, desarrollamos un plan de respuesta que incluía la mitigación para reducir la probabilidad y la contingencia para disminuir el impacto en caso de ocurrir. Identificamos el tipo de respuesta deseada, ya sea para minimizar, prevenir o asimilar, y documentamos en la tabla.

Como mencionamos, desde el comienzo la gestión de riesgos fue de los principales puntos de discusión debido a la importancia que representó para nosotros. Una vez identificados los riesgos, comenzamos debatiendo el mejor proceso para manejarlos, teniendo en cuenta que utilizamos una adaptación de Scrum y la naturaleza del proyecto, optamos por utilizar tres *Backlogs*. Para esto nos basamos en la conocida técnica de "*divide and conquer*". Como es sabido, para almacenar los requisitos de un proyecto, Scrum sugiere utilizar un *Product Backlog* (PB), un *Release Backlog* (RB) para seleccionar los requisitos desde el PB que vamos a desarrollar en un *Release* y, finalmente, a nivel de *Sprint*, define un *Sprint Backlog* para alojar los requisitos, desde el RB, a construir durante un *Sprint* determinado. De manera similar decidimos gestionar los riesgos de nuestro proyecto, utilizando un *Risk Product Backlo*g (RPB), luego, por cada *Release*, creamos un *Risk Release Backlog* (RRB), para finalmente, por cada *Sprint*,

seleccionar y guardar los riesgos potenciales de ese *Sprint* en un *Risk Sprint Backlog* (RSB).

Aplicado en nuestro caso, almacenamos todos los riesgos dentro del *Backlog* de riesgos del proyecto. Para simplificar el proceso, dividimos el mismo, comenzando por *Releases*. Al inicio de cada *Release*, creamos un *Backlog* de riesgos específico para ese periodo en la *Release* planning, siendo un subconjunto igual o menor al *Backlog* del proyecto. Luego creamos una subdivisión adicional por *Sprint* en el *Sprint* planning.

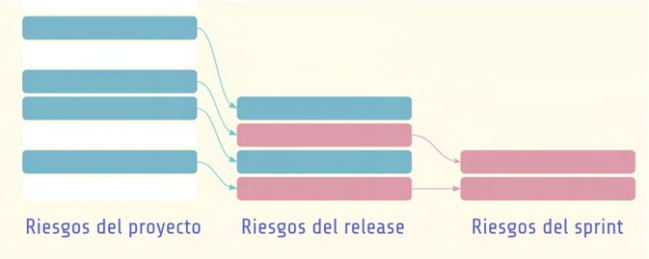


Ilustración 6.2.1: Backlogs de riesgos

De este modo, durante cada reunión de planificación de *Sprint*, abordamos la creación del *Backlog* de riesgos para el *Sprint* correspondiente. En las reuniones llevábamos a cabo las tareas de planificación de riesgos, enfocándonos exclusivamente en aquellos que afectan al *Sprint*. Para cada riesgo utilizamos el formato que detallamos previamente manejando un número de riesgos por *Sprint* limitado (no más de 3 o 4). Esta forma de registrar los riesgos, donde únicamente escribimos el nombre y el contexto del riesgo para las etapas del proyecto y *Release*, tiene como objetivo fundamental solamente identificar a los riesgos, para luego detallarlos recién en la gestión del *Sprint*. Esto brinda una gran ventaja al economizar enormemente el consumo de tiempo y esfuerzo dedicado a la planificación de los riesgos ya sea al inicio del proyecto o al inicio de cada *Release*, realizando cada inversión solo a nivel de *Sprint*.

Al finalizar cada *Sprint*, realizamos revisiones del *Backlog* de riesgos como parte de la *Sprint* Retrospective Meeting. Lo realizamos con el objetivo de validar la necesidad de incluir nuevos riesgos que podrían ocurrir, o riesgos que surgieron de forma inesperada a lo largo del *Sprint* y podrían volver a ocurrir.

6.3. Control y seguimiento

Posteriormente, llevamos a cabo el análisis y evaluación de los riesgos, considerando tanto su impacto como la probabilidad de ocurrencia. Utilizamos la siguiente planilla para facilitar las evaluaciones.

Identificacion de Riesgos			Plan y Respuesta		Nivel de Peligro		
ID	**Titulo**	**Contexto**	**Accion Preventiva**	**Accion Correctiva**	**Impacto**	**Probabilidad de ocurrir**	**Magnitud**

Ilustración 6.3.1: planilla evaluación de riesgos

El impacto como mencionamos previamente cuantifica cuánto afecta el riesgo sobre alguna de las variables de la quíntuple restricción: alcance, calidad, duración, costo y/o satisfacción del cliente. Es decir, el impacto se refiere a cómo el riesgo va a influir en el proyecto. Categorizamos la escala de impacto del riesgo con un puntaje entre 1 y 5. Siendo 1 “Ningún impacto sobre el proyecto”, 2 “Retrasa levemente el proyecto”, 3 “Retrasa sensiblemente el proyecto”, 4 “Retrasa fuertemente el proyecto” y 5 “Fracaso del proyecto”.

Por otro lado encontramos la probabilidad de ocurrencia, esta indica la probabilidad que el riesgo ocurra durante el transcurso del proyecto. Para esto evaluamos en un rango de 0.2 a 0.8, dónde 0.2 significa una probabilidad "Muy Baja" y 0.8 denota una probabilidad "Muy Alta". Excluimos los valores 0 y 1, el primero indicaría que el riesgo no podía ocurrir por lo tanto, lo descartamos, mientras que el segundo implicaba que el riesgo ya habíamos materializado, requiriendo una acción inmediata.

Una vez establecida la probabilidad y el impacto, multiplicamos estos valores para obtener la Magnitud del riesgo. Posteriormente, ordenamos los riesgos según esta magnitud, otorgando mayor prioridad a aquellos con valores más altos. Al obtener las magnitudes, nos permitió calcular la exposición del riesgo. La cual es la sumatoria de la magnitud de todos los riesgos, esta medida se utiliza como un indicador de cómo van evolucionando los riesgos durante todo el proyecto. Para monitorear y controlar estos riesgos, optamos por revisar los riesgos involucrados en cada *Sprint* en su respectiva *Sprint* Planning, evaluando si alguno se había materializado o si surgió algún cambio. Como resultado, realizamos trece revisiones de los riesgos una por cada *Sprint*, estos análisis se encuentran en el anexo.

A continuación mostramos una gráfica con la exposición del riesgo para cada *Sprint* realizado, mostrando la evolución de los riesgos dentro del proyecto. En la misma podemos ver la evolución de cada riesgo a lo largo de todo el desarrollo del proyecto. Comenzando por el primer *Release*, al inicio se mantuvo constante la exposición hasta el tercer *Sprint* donde se eleva significativamente y se mantiene para el siguiente *Sprint*. Esto se debe a la falta de tiempo que tuvimos en este periodo del desarrollo, el riesgo R1 (falta de tiempo del equipo) tuvo un gran impacto en el *Sprint* 3 y 4. Además en este periodo el riesgo R7 (requisitos cambiantes) cumplió un rol importante, el cliente pidió cambiar el servicios de infraestructura utilizado de Azure a AWS, lo cual significó gran trabajo. Si seguimos observando la gráfica del *Sprint* 5 al 11 podemos ver como la gráfica tiende a decaer, pero cuenta con algunos picos temporales. Estos picos se deben al riesgo R7, cambios repentinos en los requisitos por parte del cliente, como se puede ver en el *Sprint* 6 y 9. Por último, en el *Sprint* 12 también se puede observar un pico que se debe a la escasez de tiempo que contábamos en ese periodo del desarrollo. En el anexo, mostramos los riesgos que tomaron parte en cada *Sprint.*

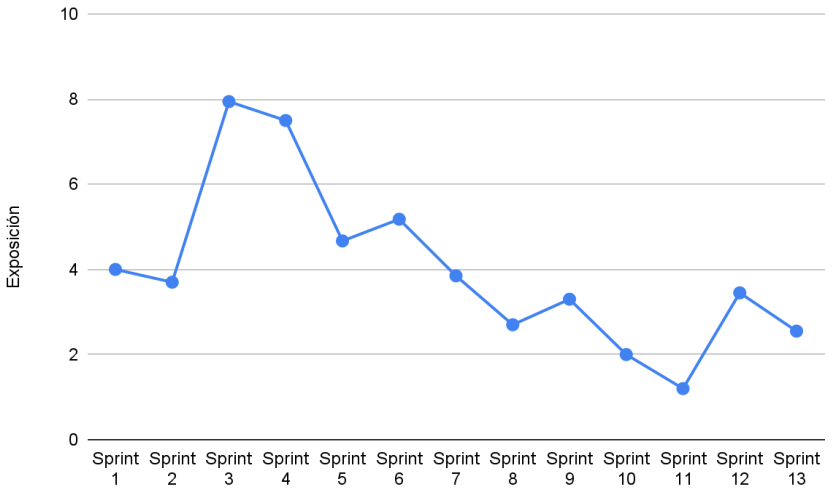


Ilustración 6.3.2: gráfica de exposición de riesgos

6.4. Evaluación de riesgos

Riesgos más relevantes

Para evaluar los riesgos más relevantes del proyecto, realizamos un gráfico donde mostramos la variación de la magnitud de cada análisis de los riesgos.

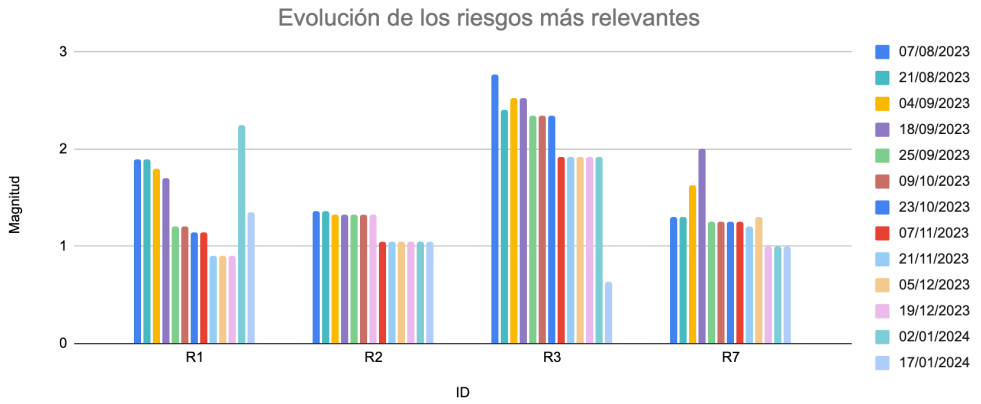


Ilustración 6.4.1: Gráfica riesgos más relevantes

Examinando el gráfico, observamos cómo los peligros más destacados experimentaron una disminución a lo largo del desarrollo del proyecto. Comenzado por el R1, podemos ver como la falta de tiempo de nuestra parte para el desarrollo tiene gran elevación al comienzo del proyecto, debido a los viajes que realizamos. Este riesgo decae a medida que avanzamos, sin embargo, se aprecia otro pico elevado al finalizar el desarrollo, que se atribuye a las fiestas festivas.

Con respecto a R2 podemos observar que a partir de noviembre la magnitud del mismo bajó considerablemente, siendo más específico a partir del *Sprint* 7. Esto se debe a que con el avance del proyecto comenzamos a estimar el esfuerzo y el tamaño con más exactitud. Si miramos el riesgo, R3 se aprecia cómo este riesgo comenzó elevado y se fue reduciendo mediante el avance del proyecto. A medida que el proyecto avanzaba comenzamos a mejorar y a capacitarnos mejor en las tecnologías que utilizamos. Además, se puede observar que en el *Sprint* 3, se eleva la magnitud con respecto al *Sprint* anterior, esto fue porque comenzamos a utilizar servicios de infraestructura en ese momento. Siguiendo el tema del uso de proveedores de infraestructura, en ese mismo periodo en el tiempo, el cliente realiza una petición de cambiar de Azure a AWS, lo cual se ve manifestado en el incremento del riesgo R7 para el *Sprint* 3 y 4. El mismo continuó casi estable por el resto del desarrollo, disminuyendo levemente.

R1 - Falta de tiempo del equipo (estudio, trabajo y viajes)

Con respecto a la falta de tiempo, identificamos que los compromisos externos del equipo, como estudios, trabajos y viajes, podrían llegar a limitar el

tiempo disponible para el proyecto, pudiendo causar posibles retrasos y disminución de la productividad. Para prevenir esto, tratamos de establecer un plan de trabajo claro, asignar responsabilidades específicas y fomentar la comunicación abierta para mantener la motivación y el compromiso. A su vez, como medida correctiva, ajustamos la distribución de tareas, re-evaluamos los plazos según era necesario y re-asignamos responsabilidades cuando detectábamos alguna falta de dedicación que pudiera afectar la calidad del producto o el alcance del proyecto.

R2 - Estimación inadecuada de esfuerzo y tamaño

Identificamos que la incorrecta estimación de las tareas representaba unos riesgos más importantes. Esto se debe a que una estimación errónea podría conducir a una planificación deficiente del proyecto, afectando negativamente los plazos y, en el peor de los casos, incumpliendo las expectativas del cliente. De esta manera, podríamos priorizar las tareas según las necesidades del cliente o agregar nuevas si disponemos de tiempo adicional.

Aunque contábamos con experiencia en la estimación de tareas individualmente, esta era la primera vez que lo abordamos de manera independiente y como un grupo. Por lo tanto, al principio, percibimos el riesgo con una magnitud alta, que fue disminuyendo con el tiempo. Este descenso se debió al mayor conocimiento adquirido, el proyecto y las tecnologías, lo que facilitó una estimación más precisa de las tareas. Sin embargo, en algunas etapas, observamos un aumento repentino debido a una mala planificación de algunas tareas. Este incidente nos llevó a revisar y mejorar nuestro proceso de estimación, y los resultados exitosos se reflejaron en la siguiente revisión, donde la magnitud del riesgo disminuyó significativamente. Gracias a la acción correctiva que teníamos planteada y la rapidez con la que actuamos pudimos mitigar el riesgo de forma segura.

R3 - Carencia de conocimiento técnico

Identificamos la falta de familiaridad con ciertas tecnologías como otro riesgo de en estas áreas podría resultar en una estimación deficiente de tareas, retrasos en el desarrollo e introducción de errores en el sistema. Por esta razón, adoptamos como parte de nuestro plan de mitigación la decisión de recibir capacitación en

aquellas tecnologías sobre las cuales no teníamos conocimientos previos. Esta iniciativa, combinada con el aprendizaje continuo durante el desarrollo con esas tecnologías, contribuyó a reducir progresivamente la magnitud del riesgo hasta alcanzar un nivel muy bajo.

Además de este enfoque, teníamos un plan de contingencia en el caso de enfrentarnos obstáculos o dificultades en el proceso de aprendizaje de alguna de estas tecnologías. En tal situación, teníamos la opción de recurrir al cliente, quien, al ser técnico y experto en esas tecnologías, podría proporcionar orientación adicional para superar los desafíos y avanzar en el proyecto.

Riesgos materializados

R2 - Estimación inadecuada de esfuerzo y tamaño

El primer riesgo en materializarse que identificamos fue la incorrecta estimaciones de esfuerzo y tamaño en la estimación de tareas. Al comienzo del proyecto definimos junto al PO que la duración de los *Releases* sean de 3 *Sprints*, los cuales duran 2 semanas cada uno. Pero al momento de transcurrir una semana del tercer *Sprint* nos dimos cuenta que no íbamos a poder cumplir con el alcance, el cual habíamos establecido previamente para este *Sprint*. Por lo tanto junto con el PO decidimos agregar un *Sprint* extra de una semana a este *Release*, para poder cumplir con el alcance establecido sin perjudicar la calidad del trabajo. De esta forma este riesgo que se materializó afectó levemente el desarrollo del proyecto, logramos manejarlo sin afectar la calidad ni el alcance.

R7 - Requisitos cambiantes

El segundo riesgo identificado fue cambios en los requisitos del proyecto por parte del cliente ya cuando el mismo está en una etapa avanzada. Originalmente, teníamos planificado utilizar Azure como proveedor de infraestructura, pero el cliente decidió cambiar a AWS cuando el proyecto ya estaba en marcha y había avanzado considerablemente.Esta modificación nos planteó desafíos adicionales, ya que implicó adaptar las configuraciones y migrar los datos y servicios a una plataforma completamente diferente. El cambio de proveedor de infraestructura puede significar un atraso en el cronograma, pero debido a que actuamos rápidamente,

pudimos realizar una evaluación y reajustar la planificación sin afectar el alcance del proyecto.

A pesar de las acciones preventivas que realizamos como mantener una comunicación abierta y constante con el cliente para anticipar posibles cambios en los requisitos y evitar sorpresas en etapas avanzadas del proyecto. Entendemos que la ocurrencia fue ajena a cualquier planificación debido que es una decisión del cliente sobre la cual no tenemos ningún efecto, pero si supimos estar bien preparados y mitigar el impacto al mínimo.

6.5. Resultados

Gracias a la gestión de los riesgos pudimos identificar riesgos potenciales y desarrollamos estrategias efectivas para mitigar su posible impacto. Evaluando la probabilidad e impacto, pudimos priorizar los riesgos más significativos y concentrarnos en abordarlos de manera proactiva. Al enfrentar riesgos como la estimación inadecuada de esfuerzo y tamaño, demostramos nuestra capacidad para responder de manera eficaz. Reajustando con flexibilidad el alcance del proyecto cuando fuese necesario, logramos las metas y objetivos que teníamos planteados. Esto se puede apreciar en las métricas de desviaciones de esfuerzo y tamaño, donde en ese *Sprint* se encuentra una mayor desviación.

Además, al enfrentarnos a desafíos como los cambios de requisitos, demostramos nuestra capacidad para adaptarnos a situaciones imprevistas, debido a la buena planificación con la cual contábamos. Aunque estos cambios podrían haber tenido un impacto significativo en el alcance y los recursos del proyecto, nuestra rápida respuesta y buena comunicación con el cliente nos permitió minimizar el impacto y llegar a un proyecto el cual tanto el cliente como nosotros estén satisfechos. Se puede apreciar en la gráfica de velocidad que el equipo pudo mantener su velocidad y completar las tareas planteadas.

7. Gestión de la configuración

7.1. Desarrollo y manejo de repositorios

Optamos por utilizar Github para manejar el código del proyecto. Creamos una organización llamada TuBocado, donde tanto los cuatro integrantes del equipo como el cliente tienen acceso de administrador para revisar el código cuando fuera necesario durante el desarrollo. Elegimos esta plataforma porque todo el equipo estaba familiarizado con ella, debido a proyectos realizados en el trabajo y en la facultad.

Creamos cinco repositorios principales para el código del proyecto:

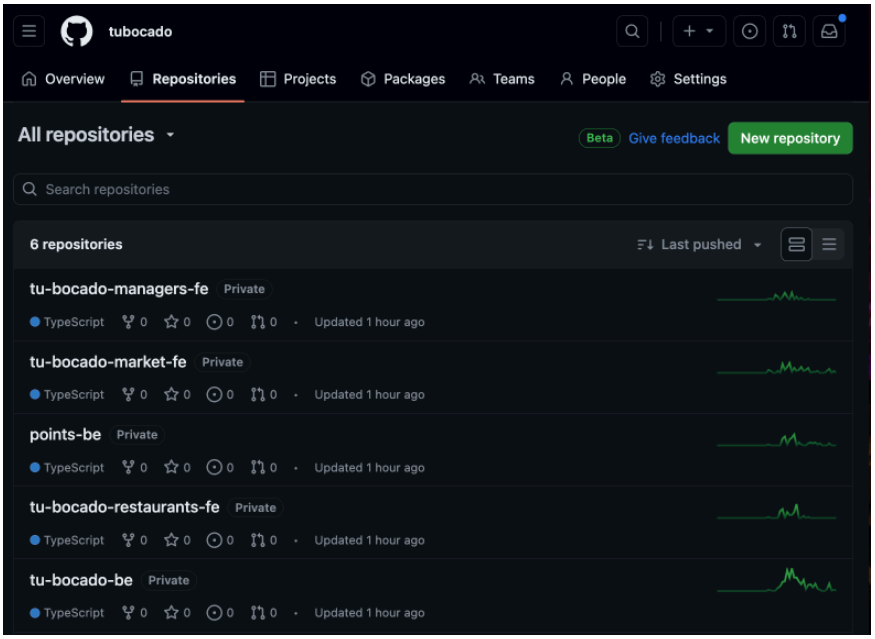


Ilustración 7.1.1: Backlogs de riesgos

Backend: contiene dos repositorios los cuales manejan la lógica del proyecto, uno para el manejo y generación de los Bocados y el otro maneja el resto de funcionalidades del proyecto.

- tuBocado/tu-bocado-be

  Este repositorio contiene el código correspondiente a una API REST que se encarga de recibir todas las peticiones de los usuarios, restaurantes y administradores de la aplicación. Se encarga de todas las funcionalidades del

sistema y de consultar y comunicarse con el servicio de puntos en caso de ser necesario. Se encuentra desarrollado en NodeJs.

- tuBocado/points-be

  Este repositorio contiene el código correspondiente a una API REST encargada de gestionar todo lo relacionado con la generación, canje y administración de Bocados de todos los usuarios. Se encuentra desarrollado en NodeJS. Además acá es donde se encuentran los parámetro de los niveles y Bocados lo cual son características muy importantes para toda la lógica que rodea a la gestión de Bocados, las cuales son definidas por el administrador del sistema.

Frontend: contiene tres repositorios, uno para cada tipo de usuario del sistema: consumidores, *managers* de restaurantes y administradores del sistema.

- tuBocado/tu-bocado-restaurant-fe

  En este repositorio vive todo lo relacionado a la aplicación web de ReactJS para los managers de restaurantes. Consume la API tuBocado/tu-bocado-be descrita anteriormente.
- tuBocado/tu-bocado-managers-fe

  En este repositorio vive todo lo relacionado a la aplicación web de ReactJS para los administradores del sistema. Consume la API tuBocado /tu-bocado-be descrita anteriormente.
- tuBocado/tu-bocado-market-fe

  En este repositorio vive todo lo relacionado a la aplicación web enfocado en diseños *mobile*, de ReactJS para los consumidores del sistema. Consume las API descritas anteriormente.

Estrategia de Branching

Git Flow es un enfoque para organizar y gestionar el flujo de trabajo en un proyecto de *software* mediante ramificaciones en Git. Se establecen dos ramas principales: "*main*" (o "*master*" en versiones anteriores) y "*develop*", junto con ramas secundarias para tareas específicas. [13]

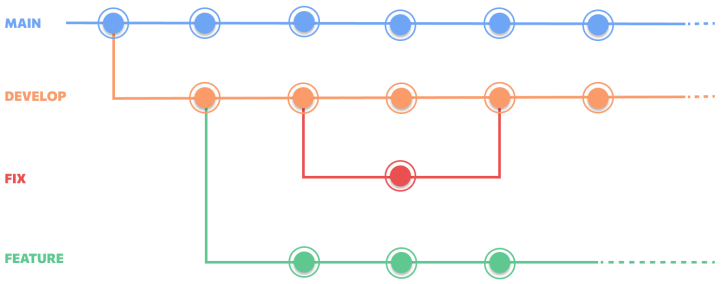


Ilustración 7.1.2: estrategia de branching

Consideramos que esta estrategia se ajustaba bien a la metodología de trabajo y por los buenos resultados obtenidos en proyectos anteriores, decidimos que este flujo era adecuado tanto para la construcción de las APIs como para las aplicaciones web, permitiendo así una guía de trabajo unificada para cada repositorio. A continuación, detallamos las distintas ramas que utilizamos en Git Flow y cómo las utilizamos en el desarrollo del proyecto.

Main. Esta rama contiene la versión estable y final del proyecto. Aquí guardamos el código que ha pasado por todas las etapas de desarrollo, revisión y pruebas. Utilizamos la rama para las versiones finales de cada *Release*, utilizamos esta rama solo para lanzamientos. Corresponde al sistema en el ambiente de producción.

Develop. Esta rama actúa como el punto de encuentro para todas las ramas auxiliares en desarrollo. La utilizamos para emplear pruebas exhaustivas (como pruebas de integración manuales, entre otras) y garantizar que todo funcione sin problemas antes de llevar el código a la rama principal (*main*). Cuando una tarea era desarrollada y aprobada por el equipo, se agrega a esta rama utilizando *Squash Merge* para mantener un historial limpio.

Feature. Estas son las ramas temporales las cuales utilizamos para desarrollar las nuevas funcionalidades del sistema. Cada rama se genera a partir de *develop*, cuando se completa el trabajo la misma se reincorpora. Cada rama creada está asociada a un *ticket* de *Jira* y a su id correspondiente, siguiendo el siguiente nombre: feature/(jira_id)-(nombre_descrpitivo). Al completar el código correspondiente y cuando la rama se incorpora a develop, la misma es eliminada.

Fix. Estas son las ramas auxiliares que son utilizadas para la corrección de errores detectados en cualquier parte del desarrollo, sin importar si son notados por el

equipo, PO, consumidores o pruebas de usuario. Cada rama creada está asociada a un *ticket* de Jira y a su id correspondiente por lo tanto, la forma de trabajo para estas ramas es la misma a las ramas feature, su nomenclatura es parecida diferenciado por: fix/(jira_id)-(nombre_descrpitivo). Además, estas ramas no incorporan una funcionalidad nueva, representan las correcciones de error de las funciones preexistentes.

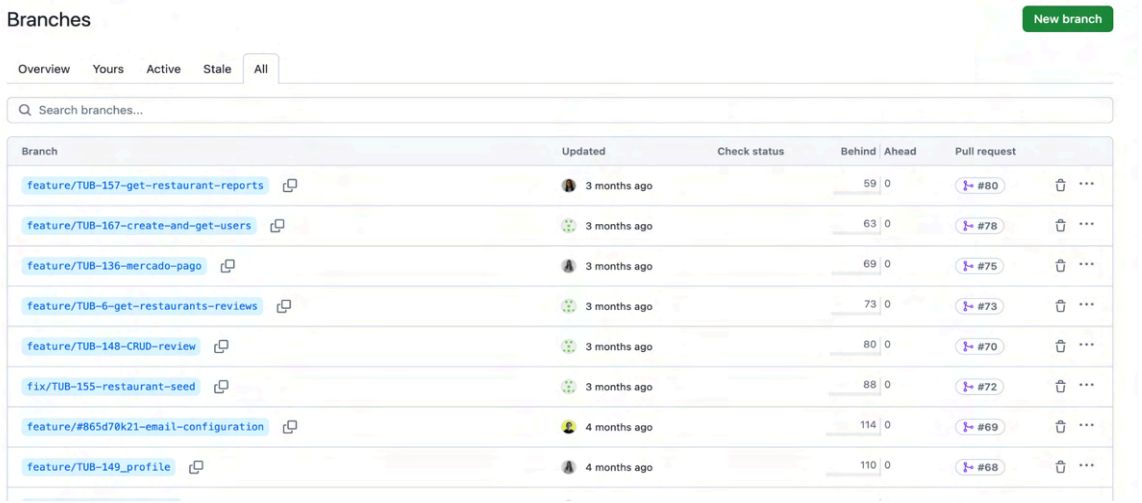


Ilustración 7.1.3: ejemplos de ramas

Cuando una tarea finalizaba, el desarrollador creaba un PR (*pull request)*. En ese momento se recorrían las tareas automatizadas con el servicio de *GitHub Actions* para asegurar la calidad del código. Finalmente para que sea posible realizar el *merge* del PR, era necesaria la aprobación de por lo menos un integrante del equipo.

En pocas palabras, adoptamos Git Flow para estructurar y controlar cómo desarrollamos el proyecto. Esto implicó crear ramas separadas para cada tarea y así garantizar que el código pasará por diversas pruebas y revisiones antes de integrarse en las ramas principales, *develop* y *main*. Este enfoque de ramificación nos ayudó a programar de manera más ordenada, minimizando errores y manteniendo un mayor control sobre la calidad del código. Detallamos más información sobre el uso e implementación de *Git Flow* en el capítulo *Delivery*.

Ambientes de trabajo

Local: sobre el cual desarrollamos las funcionalidades del sistema y corregiremos los errores que se iban generando. En el mismo trabajamos con las

ramas mencionadas anteriormente. En ejecución se encuentra información más detallada sobre el proceso incorporado en desarrollo.

Producción: este otro ambiente, cuenta con el código que se encuentra en la rama *main*. En este ambiente buscamos tener el código más estable, debido a que las funcionalidades ya pasaron por diversas pruebas. Este ambiente fue el utilizado para las pruebas de usuario y validación con el cliente. Con respecto al backend, los mismos están integrados en este ambiente con AWS Pipeline. Cada vez que realizamos un *merge* en las rama *main*, se activa automáticamente un proceso en el pipeline de AWS lo que hace que se genere el proceso de construcción y una vez que esté terminado pasa a desplegar las actualizaciones en ElasticBeanstalk. Por otro lado el *frontend*, se despliega automáticamente utilizando Amazon Amplify cuando se realiza el merge a main.

7.2. Documentación

Para la gestión de la documentación comenzamos utilizando la plataforma Notion para estructurar y almacenar los archivos. Debido a la limitación del número de carpetas del plan gratuito de la aplicación optamos por continuar el desarrollo en Google Drive.

Utilizamos Google Drive para el desarrollo del documento principal de la entrega debido a la naturaleza de open source de la aplicación y su facilidad para trabajar en tiempo real. Estas ventajas nos facilitaron el desarrollo en simultáneo, pudiendo compartir entre el equipo con facilidad los cambios de cada integrante.

En la plataforma mencionada previamente llevamos un registro detallado del conocimiento adquirido durante todo el proceso de desarrollo y la investigación (*Design Thinking*). Lo utilizamos para guardar todo tipo de información que pudo llegar a ser relevante para el proyecto, para poder utilizarla a futuro en caso de ser necesario. Inicialmente utilizamos esta herramienta para guardar todo tipo de archivos en modo borrador, no contábamos con un sistema predefinido para archivar los documentos, sin embargo nos aportó gran valor para acceder a información la cual ya había sido descubierta. Una vez que procesamos la información que se encontraba en modo borrador, redactamos la información de

manera más estructurada y formal, agregando gradualmente datos durante la investigación y el desarrollo.

Para continuar, decidimos seguir utilizando el conjunto de herramientas que Google Drive nos ofrece para el manejo de datos y documentos detallados que estábamos formando para la entrega. Esto se debió a la capacidad que tiene la herramienta para almacenar una amplia variedad de archivos y gráficos, además de ofrecer una mayor flexibilidad en términos de estilos.

Otra de las motivaciones para cambiar de herramienta fue la versatilidad que ofrece Google Drive en comparación con Notion. A diferencia de esta última, Google Drive no restringe a los usuarios a un único programa de edición de texto. En su lugar, proporciona una variedad de herramientas específicas, como hojas de cálculo para análisis de riesgos y evaluación de resultados de pruebas de usuarios, presentaciones para revisiones con correctores y formularios para evaluaciones iniciales.

Para realizar las correcciones de cada sección de la documentación con el tutor contábamos con una carpeta para cada capítulo la cual identificabamos con un número de versionado. Al finalizar el desarrollo de un capítulo un integrante del equipo, idealmente el que tuvo menos participación en este capítulo, lo leía y realiza correcciones. Una vez realizadas las correcciones y que ese integrante esté satisfecho con el resultado, notificamos al tutor que el capítulo estaba pronto para corregir. Luego, copiamos el archivo corregido sumando un valor al índice de la versión, una vez corregida la versión anterior no se modifica más. Repetimos este proceso hasta estar satisfechos tanto el equipo como el tutor.

Posteriormente, transferimos toda la información a un único archivo de Google Doc para redactar la versión final de la parte entregable del proyecto. Donde controlamos que el archivo cuente con todas las características que requiere para entregar un producto de calidad.

8. Aseguramiento de calidad

La gestión aseguramiento de la calidad es un factor importante en la creación de software y el éxito del proyecto. El objetivo de cualquier proyecto es garantizar la calidad del resultado esperado y satisfacer tanto a los usuarios como a los clientes, y todos los interesados. En este capítulo detallaremos las estrategias y las actividades que aplicamos para el aseguramiento de la calidad del software, dando una visión de su valor añadido y el impacto positivo en nuestro proyecto.

8.1. Definición de calidad

Para Crosby, un referente en el área de calidad, la calidad es "conformidad con los requisitos.“ [59]. Mientras que para Juran, otro referente, la calidad es “adecuación al uso” [60]. Por lo que, según diferentes autores, la calidad tiene diferentes definiciones pero todas conducen a la misma idea. Entregar un producto que satisfaga la necesidad y expectativa del cliente sea por la adecuación al uso o porque cumple con todos los requisitos. La calidad implica satisfacer necesidades y expectativas de los clientes a lo largo de la vida del producto.

8.2. Estrategia seguida

La estrategia de prácticas de aseguramiento de calidad que realizamos fue la de ir incorporando medidas de manera progresiva a lo largo del proyecto. Fuimos implementando las estrategias a medida que íbamos identificando las necesidades específicas de las distintas etapas tanto en *Delivery* como en *Discovery*. En ese sentido, podemos decir que optamos por tener una calidad del tipo incremental.

Entre estas prácticas, algunas se focalizan específicamente en garantizar la calidad, como las revisiones de código y la programación de a pares, mientras que otras, como la aplicación del Design Thinking y la adaptación personalizada de Scrum, hacen a la gestión y ejecución del proyecto, pero tienen un impacto positivo directo en el aseguramiento de calidad.

Si bien la calidad del proyecto tiene una importancia significativa e implementamos diversas medidas para garantizarla, dada la naturaleza ágil del proyecto y su esencia de ser una prueba de concepto, decidimos adoptar un

enfoque pragmático hacia la planificación de la calidad y no crear un plan de calidad exhaustivo al comienzo. En lugar de comprometernos con un plan de calidad exhaustivo desde el inicio, optamos por realizar una aseguramiento de calidad incremental, más flexible y adaptable, que nos permitió responder rápidamente a los cambios y aprender de la retroalimentación de los usuarios en etapas tempranas.

Al priorizar la entrega de valor y la capacidad de adaptación rápida, no solo validamos la viabilidad de la propuesta de negocio, sino que también aseguramos que el desarrollo del producto se mantenga alineado con las necesidades reales de los usuarios ya que la calidad es cumplir los requerimientos y expectativas del usuario y eso fue lo que buscamos al elegir esta estrategia.

8.3. Actividades de aseguramiento de la calidad

En esta sección listamos las actividades realizadas para asegurar la calidad del desarrollo del *software* y del producto.

Creación de prototipos

Una de las medidas que tomamos para el aseguramiento de la calidad fue la creación de pretotipos y prototipos. Como mencionamos en la sección de definición de requerimientos, realizamos varias iteraciones de las últimas etapas *Design Thinking* para poder llegar a la solución actual. El desarrollo comenzó con la versión inicial del prototipo y luego, después de cada release, tomábamos el feedback de las release reviews, encuestas y pruebas y realizabamos ajustes sobre el prototipo, obteniendo así diferentes versiones mejoradas del mismo. Este enfoque iterativo nos permitió alinear mejor nuestra aplicación con las expectativas de los diferentes usuarios, aumentando su satisfacción y usabilidad del producto final. Para poder ver en detalle el prototipo y pretotipo ir a la sección A.12. Pretotipo y A.13. Prototipo del anexo.

Capacitación

Una de las medidas que tomamos al enfrentarnos a tecnologías desconocidas fue la de capacitarnos. Algunos de los integrantes que no tenían experiencia en ReactJs o NodeJs se capacitaron tomando cursos online en

plataformas como Udemy o LinkedIn. Además, al comienzo se había decidido utilizar Azure, y luego hubo un cambio de decisión por parte del cliente hacia AWS, haciendo que nos capacitemos en principio en Azure y luego en AWS para adaptarnos al cambio. Ambas capacitaciones incluyeron lectura de documentación y realización de cursos online. Otra de las medidas que tomamos para capacitar al equipo fue la realización de *spikes* lo cual se puede ver más en detalle en Pruebas de tecnología.

Pair Programming

Para desarrollar funcionalidades técnicamente desafiantes que requerían un profundo conocimiento del proyecto y del contexto de negocio, empleamos la programación en pares. Identificamos ciertas funcionalidades, como la generación de puntos, y la trazabilidad del cambio de cotización de bocados, entre otras, que demandaban cierto grado de complejidad mayor que otras y que se verían altamente beneficiadas por la utilización del *pair programming*.

Aplicación de Clean Code

Adoptamos varios de los principios de Clean Code para hacer el código legible, fácil de entender y mantener. Entre ellos están:

- Funciones con nombres descriptivos, cortas y siguen un patrón coherente.
- Evitar tipos de retorno en los nombres de las funciones y mantener un único nivel de abstracción.
- Las variables tienen nombres cortos, descriptivos y nemotécnicos.
- Formato vertical y horizontal, con líneas en blanco cuando es necesario y líneas de menos de 100 caracteres.
- Espacios antes y después de caracteres como “=” e indentación correcta.
- Las excepciones se manejan centralmente, con mensajes definidos por separado para facilitar futuras modificaciones.

Consultas con expertos

Dentro de nuestras iniciativas para garantizar la calidad, realizamos consultas a especialistas. En la etapa inicial de investigación, nos apoyamos en expertos en

marketing multinivel que nos brindó el cliente para confirmar que la aplicación cumplía con los estándares requeridos. Adicionalmente, llevamos a cabo revisiones de la arquitectura en más de dos ocasiones con el experto Gastón Mosqués como se explica en el capítulo de arquitectura en las validaciones con el experto.

Heurísticas de Nielsen

Para poder cumplir con la calidad esperada del producto, tuvimos en cuenta la Usabilidad. Con el cliente buscamos priorizar lo que el usuario podría “ver”, incluyendo lo relacionado a la experiencia del mismo. Para esto utilizamos las heurísticas de Nielsen como un marco de calidad para medir el cumplimiento de una interfaz intuitiva y fácil de usar. Para profundizar en esta medida, véase la sección de Usabilidad dentro del capítulo arquitectura, tácticas.

Ceremonias

Sprint y *Release Reviews*

Como mencionamos anteriormente en el capítulo de *Delivery*, realizamos *Sprint Reviews*, al final de cada *Sprint*, y *Release Reviews* al culminar un *Release*. Estas reuniones son fundamentales para el aseguramiento de la calidad, ya que proporcionan una instancia para evaluar el trabajo completado y garantizar que cumple con los requisitos del cliente y los objetivos del proyecto.

Retrospectives

Las retrospectivas tanto de *Sprint* como *Releases*, las cuales se explican en el capítulo 5.10 Ejecución, fueron esenciales para la mejora continua del proceso de desarrollo y por lo tanto, el aseguramiento de la calidad. En base a los análisis e intercambio de opiniones, creamos *Action Items* que buscaban reflejar los cambios a realizar en los siguientes *Sprints*, con el objetivo de adaptar y refinar los procesos de desarrollo de *software* para maximizar el valor entregado.

Release y *Sprint Plannings*

Como actividades para asegurar la calidad también incluimos las ceremonias de Scrum release planning y sprint planning. En estas instancias, se daba un

intercambio sobre la planificación del proyecto con el PO muy importante. Nos ayudaba a asegurarnos de que todos los miembros del equipo estuvieran alineados y compartieran la misma visión del proyecto. Además, en estas reuniones aprovechábamos la oportunidad para negociar con el cliente sobre posibles ajustes en el alcance del *Product Backlog*, cuando fuera necesario, lo que es crucial para el aseguramiento de la calidad.

Métricas

Creemos que el uso de métricas es una herramienta fundamental para el aseguramiento de la calidad, como dice Peter Drucker, "*Si no puedes medirlo, no puedes controlarlo; si no puedes controlarlo, no puedes gestionarlo; y si no puedes gestionarlo, no puedes mejorarlo*"[62]. En ese sentido, pusimos un particular énfasis en la importancia de las métricas para lograr asegurar la calidad. Dentro de las métricas, las cuales se explican en el capítulo 9. Métricas, especificamos la medición y los resultados de los *Key Result* de cada uno de los objetivos planteados. Específicamente para promover el aseguramiento de calidad, nos enfocamos en el objetivo “Entregarle un producto de calidad al cliente y que le proporcione un valor añadido”, y los *Key Results* que lo miden.

Verificación de requerimientos del sistema

Antes de cada *merge* a *main* y finalización de un *Release*, realizábamos una revisión para asegurarnos de que todos los requerimientos del *Release* se encontraban cubiertos antes de la validación con el cliente. Esto nos permitía tener en claro cada uno de los estados en los que se encontraban las historias de usuario, lo que nos facilitaba la planificación y ajuste del trabajo necesario para alcanzar los estándares de calidad establecidos para el producto. Este enfoque no solo buscaba cumplir con los objetivos de calidad, sino que también procuraba minimizar el impacto negativo que podría surgir de la omisión de requisitos.

Lecciones aprendidas

Para poder aprender de lo que fuimos realizando y no volver a cometer los mismos errores, fuimos generando un registro de lecciones aprendidas. Con respecto a la gestión de la calidad, este registro sirve para mejorar la eficiencia y la

eficacia al gestionar la misma. Esta práctica promueve la mejora continua, lo que nos ayudó a prevenir la repetición de errores en etapas futuras del proyecto o en nuevos proyectos. Al integrar estas lecciones en los procesos y metodologías de trabajo, buscamos mejorar la calidad del producto final y asegurar una entrega más alineada con las expectativas del cliente.

8.4. Pruebas

El propósito de realizar diversas pruebas durante el desarrollo de *software* es prevenir que los errores identificados se transformen en problemas perceptibles para los usuarios finales. Con este enfoque buscamos asegurar la calidad garantizando que cualquier fallo sea detectado y corregido por el equipo de desarrollo antes de que el producto llegue a las manos de los usuarios. A continuación, vamos a detallar los diferentes niveles en los que se clasifican estas pruebas [63].

Pruebas Unitarias

Objetivo

El objetivo de realizar pruebas unitarias es probar que cada componente u unidad de código funcione de la manera esperada, de forma independientemente a las otras funcionalidades. El desarrollo de estas pruebas, es una práctica muy importante para ayudar al aseguramiento de la calidad de cualquier *software*.

Ejecución de las pruebas

Con lo que respecta a la implementación, para poder generar las pruebas unitarias del código lo que hicimos fue utilizar Jest, un *framework* conocido de NodeJs para generar pruebas unitarias. Esto implicó simular comportamientos de dependencias, asegurándonos de que cada función o componente rinda como se espera. Uno de los desafíos acá fueron los mocks de APIS externas como puede ser Mercado Pago y la conexión entre los servicios.

Para cada funcionalidad realizada creamos diferentes escenarios para poder asegurar el cumplimiento de todos los casos requeridos. Para lograr esto, fuimos creando los *tests* a medida que íbamos creando cada una de las funcionalidades.

Además nos aseguramos de realizar pruebas para cada módulo, según las guías de *Bounded Context*, buscamos validar todos los casos de usos para asegurarnos que cada componente del sistema funciona correctamente dentro de los límites y reglas del dominio al que pertenece. Esto asegura que las interacciones entre distintas partes del sistema sean coherentes y que el modelo de dominio se mantenga consistente. A continuación mostramos un ejemplo de una prueba unitaria realizada es el registro de un nuevo administrador:

Nuestro objetivo es evaluar todos los escenarios posibles. Por eso, incluimos diferentes escenarios como la existencia previa del email de un administrador y el caso contrario, cuando este no existe. Además, nos propusimos alcanzar un 85% de cobertura de código para determinar el número adecuado de escenarios y pruebas necesarias para cada funcionalidad. Cuanto mayor sea la cantidad de pruebas realizadas, mayor será la probabilidad de que el sistema funcione según lo esperado, reduciendo así la necesidad de extensas pruebas de integración. En el anexo se puede ver un A.42. Evidencia de prueba unitaria creada.

En la siguiente imagen podemos ver cómo estructuramos las pruebas de cada uno de los módulos y el resultado de la cobertura de las pruebas.

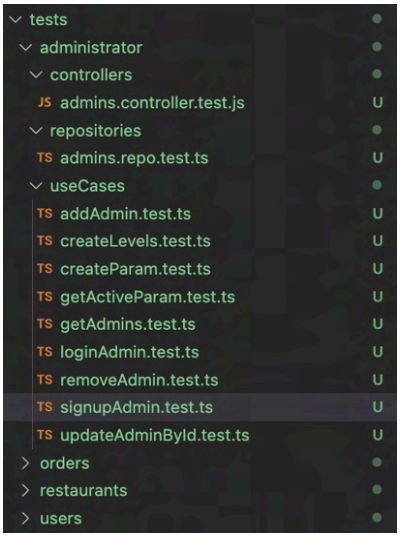


Ilustración 8.4.1: evidencia cobertura de código

Como observamos en la ilustración 8.4.1, organizamos los *tests* en módulos para reducir la dependencia entre los mismos y poder probar cada módulo aislado.

Para poder correr los *tests* utilizamos yarn run test, de esta manera podríamos verificar que todos los *tests* corrieran de forma satisfactoria. A su vez corríamos el open coverage de Jest para poder así, evaluar el porcentaje de cobertura de las pruebas. Para esto corriamos *open coverage/lcov-report/index.html.*

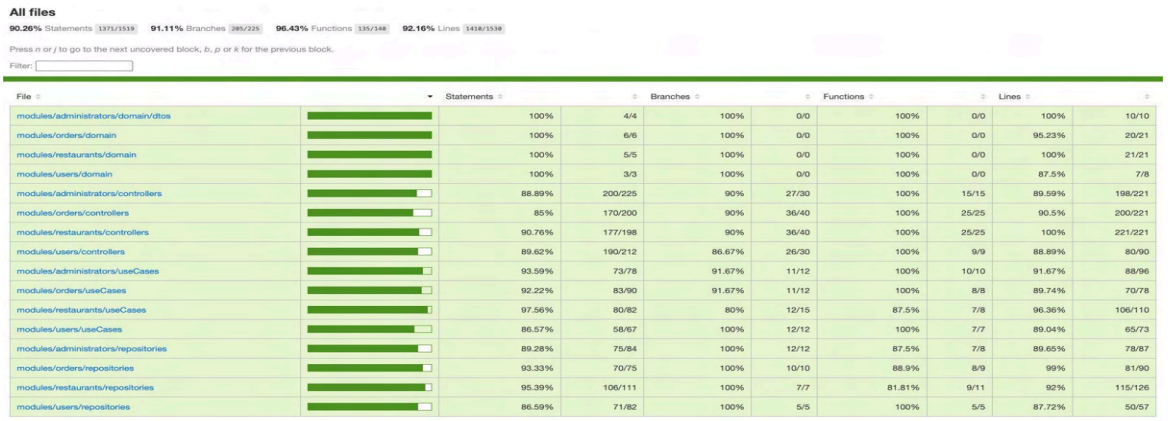


Ilustración 8.4.2: evidencia de cobertura de código

Para la cobertura de código superamos el objetivo de 85% de cobertura en todos los módulos. Se puede apreciar en la ilustración 8.4.2, se encuentra más evidencia en A.41. Evidencia de cobertura de código. Además, estructuramos las pruebas por módulo, y nivel para asegurar un mejor aislamiento, evitando generar dependencias cruzadas entre las mismas. Podemos observar la cantidad de *tests* ejecutados en el servicio del monolito modular principal y como se ve dentro de un módulo, particularmente el de administradores.

En conclusión, creemos que la aplicación de pruebas unitarias nos permitió identificar y corregir errores en las etapas más tempranas del desarrollo, simplificando la depuración y reduciendo los costos de corrección. Además, cada prueba unitaria sirve como documentación de las funcionalidades, simplificando el proceso para integrantes del equipo que quisieran comprender el funcionamiento de funciones específicas desarrolladas por los otros.

Pruebas de integración

Objetivo

El objetivo de las pruebas de integración es poder verificar que los distintos módulos o servicios utilizados en nuestra aplicación funcionan bien en conjunto. Un ejemplo

de esto es verificar que funcione bien la interacción entre los servicios (el monolito principal y el servicio de puntos) y que funcione de manera esperada.

Ejecución de las pruebas

Cuando un integrante del equipo terminaba de realizar una historia de usuario, como mencionamos en el capítulo de metodología, el responsable de dicha historia procedía a fusionar los cambios con la rama develop y luego se trasladaba la historia a la columna “Integration Testing”. En ese momento, iniciábamos las pruebas de integración manuales. Este proceso nos permitía validar la funcionalidad de la historia de usuario, ya incorporada en la versión más reciente del sistema.

Para llevar a cabo las pruebas de integración, el primer paso era seleccionar cuidadosamente los escenarios de prueba integrados en la historia de usuario realizada, basándonos en los flujos de trabajo del sistema y las interacciones entre los componentes más importantes. Esto abarcaba desde probar las integraciones con bases de datos hasta la comunicación entre los dos servicios, cuando correspondía, y la interacción con la interfaz de usuario en el *frontend* correspondiente.

Una vez seleccionado esto, un integrante llevaba a cabo dichas pruebas utilizando datos de prueba representativos. Manualmente, replicábamos las acciones que realizaría un usuario, observando y registrando los resultados obtenidos.

En cada etapa del proceso, documentábamos nuestros hallazgos. Esto incluía capturas de pantalla de los errores, descripciones de los comportamientos inesperados y registros de las interacciones con el sistema. En caso de encontrar fallos, registrábamos los problemas identificados durante las pruebas en el ticket a probar y lo movíamos nuevamente a la columna "In Progress". En la siguiente imagen podemos ver un ejemplo de prueba de integración que al realizar la prueba falló la conexión entre el sistema de principal y la conexión con el sistema de puntos.

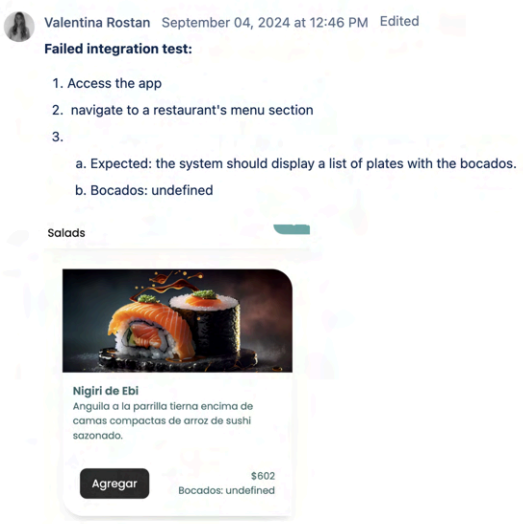


Ilustración 8.4.3: ejemplo de prueba de integración manual

Luego, una vez que las pruebas eran superadas, movíamos la historia de usuario a la columna "Deployment", donde permanecía hasta el momento el despliegue.

Por último, este método no solo lo aplicaba a historias de usuario individuales sino que también implicaba una revisión detallada y conjunta de todas las funcionalidades del sistema, especialmente antes de su fusión con la rama principal.

Este enfoque manual para las pruebas de integración, nos permitía identificar y resolver problemas que podrían no ser evidentes en las pruebas automatizadas, especialmente aquellos relacionados con la interacción entre diferentes partes del sistema.

Pruebas de aceptación

Objetivo

El objetivo de las pruebas de aceptación es que el sistema cumple con los requisitos y las expectativas de los usuarios finales.

Ejecución de las pruebas

Como mencionamos anteriormente, al completar los *Releases* y asegurar que todas las tareas cumplan con la *definition of done*, realizamos pruebas de aceptación por usuario. Esto lo realizamos en el entorno de producción, ya que,

dado que es una POC aún no está abierto al público por lo que las *keys* de Mercado Pago, Sendgrid, etc, son las de *testing*; lo que permitía a los usuarios probar todos los flujos sin problema.

Para llevar a cabo las pruebas de aceptación por usuario, el primer paso era seleccionar los participantes. Para esto seleccionamos diferentes usuarios para cada tipo de aplicación web que teníamos, particularmente 5 usuarios para la aplicación web de consumidores, 3 dueños de restaurantes para la de restaurantes y un usuario (el PO) para la aplicación web del administrador.

A estos participantes les brindábamos un documento preparado, en el cual detallábamos casos de prueba específicos organizados por funcionalidad. Cada caso incluía los escenarios de prueba (criterios de aceptación o escenarios dependiendo del *Release*) y los resultados esperados (las salidas que deberían devolver), facilitando así una evaluación estructurada del *software*. Este documento lo elaboramos utilizando hojas de cálculo, donde dejábamos un espacio para los resultados obtenidos y comentarios de los usuarios, así como un lugar para que puedan adjuntar pruebas visuales de ser necesario. A medida que iban pasando los *Releases*, se le iban agregando funcionalidades a la hoja de cálculo pero siempre se mantienen las anteriores, a excepción de necesitar realizar algún cambio pequeño.

Una vez seleccionados los participantes y creada la hoja de cálculo con las funcionalidades a probar, uno de los miembros del equipo realizaba una sesión informativa con los usuarios para explicarles cómo debían leer la hoja de cálculo y cómo debían llenar la misma. Luego de esto, sin ofrecer explicaciones previas sobre el funcionamiento del sistema, ubicábamos a los usuarios en la pantalla de inicio de la aplicación correspondiente y les proporcionamos una lista de tareas a completar.

En la sección A.26 del anexo se puede ver el *template* de las pruebas del primer Release para todos los usuarios, allí se puede observar todos los puntos que mencionamos que integraban las pruebas.

Durante la ejecución de las pruebas por parte de los usuarios, nos dedicábamos a observar detenidamente sus reacciones y la manera en que cada uno interactuaba con el sistema. Además, brindábamos apoyo para documentar sus experiencias y aclarar cualquier duda que surgiera relacionada con el proceso de

prueba, sin entrar en detalles sobre el funcionamiento específico de la aplicación. Este soporte podría ser, por ejemplo, para adjuntar evidencia visual cuando fuera necesario, facilitando así una recopilación completa y detallada de feedback.

Una vez finalizada la prueba guiada, les realizamos una pregunta específica a los usuarios para poder captar su impresión sobre la aplicación. Nos interesaba particularmente conocer su disposición a recomendar nuestro producto, formulando la pregunta: “¿Cuán probable es que recomiendes nuestro producto a un amigo o colega?”. Esto nos ayudó a recopilar algunas de las métricas y cómo hicimos para calcularlas, todo esto lo mencionaremos en el siguiente capítulo 9.Métricas, brindándonos una perspectiva clara sobre nuestro progreso en el cumplimiento de los objetivos de calidad.

Esta información, junto con el análisis de las hojas de cálculo, nos proporcionó datos valiosos sobre el tiempo que los usuarios tardaron en completar cada tarea y sus impresiones generales sobre el sistema. Además, guardabamos las respuestas de los usuarios para luego clasificarlas y evaluarlas en conjunto con el feedback del PO y así poder decidir sobre futuras mejoras o correcciones.

8.5. Estándares

Los estándares de calidad proporcionan la base para la evaluación y medición de las actividades y los productos de trabajo durante todo el proceso de desarrollo. Por eso creemos que aplicándolos generamos uniformidad, consistencia, y rigurosidad a los métodos y actividades de desarrollo de *software*.

Estándares de codificación

Estándares de código fuente

A lo largo del proyecto fuimos aplicando estándares, tanto para las tecnologías del *backend* como las del *frontend*. Para decidir los estándares, decidimos seguir recomendaciones de las comunidades para mantener uniformidad y coherencia para los siguientes desarrolladores.

En el tablero de trabajo, existe una columna que se llama “In Review”. Como mencionamos en la sección anterior de pruebas, allí realizábamos revisiones de

código y el trabajo del revisor era, entre otras cosas, asegurar que se sigan los estándares que se hayan adoptado.

Backend - NodeJs

En la tabla 8.5.1 establecemos los lineamientos y normas que definimos para el desarrollo del *backend*.

Área	Descripción	Fuente
Estructura del proyecto	Utilizamos una estructura modular, separando lógica de negocio, acceso a datos, y presentación	CleanArchitecture, NodeJS[64]
Tipado Estático	Utilizamos TypeScript para añadir tipado estático y mejorar la legibilidad y mantenibilidad del código.	Estándares de Typescript Google[65]
Pruebas	Como mencionamos en la sección anterior implementamos pruebas utilizando Jest para asegurar que el código funciona como se espera para esto seguimos los estilos que dicta Jest.	Jest testing best practices in NodeJs[66]
Convenciones de Código	Seguimos guías de estilo como ESLint con guia de estilo de Airbnb para mantener un estilo de codificación, además usamos Prittier con configuraciones específicas para NodeJs.	Airbnb Javascript Style Guide[67]
Estructuración de API	Seguimos las guías de API REST	Servicios Web API Rest Guide[68]

Tabla 8.5.1: tabla de lineamientos para backend

Frontend - ReactJs

En la tabla 8.5.2 establecemos los lineamientos y normas que definimos para el desarrollo del *frontend*.

Área	Descripción	Fuente
Estructura del proyecto	Organizamos los archivos en carpetas por componentes, pages, utilidades, y layouts.	React project structure best practices[69] y TypeScript with React best practices[70]
Tipado Estático	Usamos TypeScript para añadir tipado para mejorar la legibilidad y mantenibilidad del código. Además definimos propiedades (props) y estados (state) de componentes con TypeScript para tener un chequeo de tipos seguro.	Typescript with React[70]
Convenciones de Código	Adoptamos las convenciones de código definidas por Airbnb para React con	Airbnb Javascript Style

	ESLint. Además utilizamos como guía las mejores prácticas de React.	Guide[68] React Best Practices[71]
Estilos	Utilizamos soluciones con Tailwind y en algunos casos particulares styled-components para manejar estilos con tipos.	TailwindCSS[72]

Tabla 8.5.2: tabla de lineamientos para backend

Además de estos, decidimos utilizar React Hooks como un estándar ya que nos permiten utilizar el estado y otras características de React sin escribir una clase, simplificando el código y mejorando la legibilidad y organización.

Por otro lado, podemos visualizar en las tablas de descripción de estándares, en ambos casos utilizamos *Typescript*, tanto en ReactJs como en NodeJs. La utilización de este contribuye mucho al aseguramiento de la calidad ya que nos brinda varias ventajas para la detección y prevención de errores.

- Cuenta con tipado estricto: esto ayuda a detectar errores en tiempo de compilación, antes de que el código se ejecute.
- Documentación: los tipos funcionan como una forma de documentación, haciendo que el código sea más legible y fácil de enterner.
- Prevención de errores comunes: al forzar a los desarrolladores a ser explícitos sobre los tipos de sus variables y retornos de funciones, TypeScript previene errores comunes en JavaScript, como las operaciones en tipos incompatibles

Con lo que respecta a la utilización de Typescript, como mencionamos seguimos las guías recomendadas por Google [65]. En el Anexo se muestran ejemplos de aplicaciones de estos estándares en la sección A.25. Los estándares incluyen:

- No utilizar el tipo: any.
- Utilizar “let” y “const” en lugar de “var”.
- Nombres de variables en camelCase.
- Nombres de funciones en camelCase.
- Nombres de enumeraciones en PascalCase

- Nombre de constantes usar CONSTANT_CASE.
- Nombres de tipos e interfaces con PascalCase.
- No utilizar tipo: any
- Largo máximo de linea 100 caracteres
- Definir tipos de interfaces y tipos de retorno para funciones
- Usar *undefined* en lugar de *null.*
- Evitar comentarios obvios

También podemos observar en las tablas que agregamos extensiones a los editores de código para ayudar a cumplir estos estándares, estos ayudaban a indicar cuando exista un error de formato o estándar. Algunos de los que utilizamos fueron *ESLint* y *Prettier*. Por un lado, *ESLint* analiza de forma estática el código para encontrar problemas. Además, tiene sugerencias de posibles correcciones para ayudar a mejorar el código. Por otro lado, *Prettier* es un "formateador de código conocido que admite una variedad de lenguajes de programación diferentes". Esto nos permite establecer el estilo del código y evitar formatear el código de forma manual.

Por último, definimos estándares de codificación propios que teníamos que cumplir a lo largo de todo el proyecto o en aplicaciones y servicios particulares. A continuación, detallaremos los mismos en una tabla para que se pueda visualizar mejor.

Áreas	**Localización del proyecto donde se aplica**	**Descripción**
Idioma	Total	Inglés
Comentarios	Total	Solo incluimos comentarios en caso de: - Explicar decisiones por negocio - Explicar funciones complejas - Documentar funciones importantes
Errores	*Backend*	Usamos de clases personalizadas para manejo de errores
Pruebas	*Backend*	Cobertura de pruebas de al menos 85%(esto también definido en las *definitions of done*).
Depuración	Total	No utilizamos console.logs para el *debug* en producción.

Hooks Personalizados	*Frontend*	Todos los hooks personalizados seguían la convención de nomenclatura comenzando con "*use"* seguido de una descripción en CamelCase. Un ejemplo es *useWindowWidth* que calculaba el ancho de las pantallas.
Organizacion de constantes	*Frontend*	Decidimos organizar las constantes de los sistemas en un solo punto para que sean reutilizables y fáciles de cambiar en un futuro. Además, estas deben estar organizadas alfabéticamente en el archivo para facilitar la lectura.
Seguridad	*Frontend*	Aseguramos la aplicación contra ataques XSS utilizando correctamente dangerouslySetInnerHTML.
Mensajes de commit	Total	Seguimos un formato específico de *commits.* Esto lo hicimos para facilitar la comprensión del propósito del *commit*. Dependiendo si era una corrección o una funcionalidad, el mensaje debía arrancar con "feat:" o "fix:" y luego una breve descripción de lo realizado.

Tabla 8.5.3: tabla de estándares personales

Estándares de registro de defectos

Con lo que se refiere al registro de defectos, existe todo un proceso para el registro de estos. Cada vez que un defecto o incidente era encontrado tanto por parte de alguno de los integrantes de equipo como por parte de usuarios o del *product owner,* era registrado en el *Product Backlog* por uno de los miembros del equipo. La decisión de incluir una corrección de un incidente dentro de un *Sprint* la tomábamos durante las reuniones de *Sprint Planning*, donde, al priorizarlos, evaluábamos cuáles eran más urgentes de arreglar y cuales podían esperar para el siguiente *Sprint*. Por otro lado, la decisión de qué incidentes incluiríamos en un *Release* se toma durante la reunión de *Release Planning* con el PO, permitiendo así priorizar y elegir cuales consideran más importantes. Es importante destacar que las *Release Plannings* se realizaban después a las *Release Reviews* y a las pruebas de aceptación, tanto de los usuarios como del PO, lo que generaba una base de incidentes registrados para discutir con el cliente y así poder determinar cuáles son prioritarios.

Estándares de documentación

Para crear la documentación, decidimos seguir las normas definidas por la Facultad de Ingeniería de la ORT. El documento principal que utilizamos como

referencia fue "Guía para la entrega final - versión 2021". Además, en junto a estos, nos guiamos siguiendo los documentos 303 y 302. El documento 303 ofrece una checklist para el formato, y el documento 302 detalla los criterios de presentación para los proyectos de fin de carrera. Estas guías fueron esenciales para asegurar que la documentación del proyecto mantuviera un alto nivel de calidad y uniformidad. A continuación resumimos los estándares de documentación en las siguiente tabla:

Documento	**Estándar**	**Referencia**
Documento final	Guía para la entrega final - versión 2021	Aulas - ORT
Hoja de verificación de formato	Documento 303: Hoja de verificación de formato de trabajos finales de carreras de la facultad de ingeniería.	ORT
Normas de presentación del trabajo.	Documento 302: Normas específicas para la presentación de trabajos finales de carrera.	ORT

Tabla 8.5.4: tabla de resumen de estandares de documentacion

Estándares de proceso de desarrollo

Para los estándares del proceso de desarrollo decidimos seguir el modelo de flujo de trabajo de *GitFlow*. A pesar de evaluar otras alternativas como *Trunk Base*, decidimos optar por *GitFlow* justamente para establecer un estándar entre todos los desarrolladores, que sea conocido y así poder realizar una eficiente gestión de ramas y versiones.

Con lo que respecta al proceso de publicación e integración de nuevos cambios en el código de los repositorios, cada vez que un integrante del equipo terminaba el desarrollo de una funcionalidad o corrección, era necesario crear un *pull request* [73] a la rama *develop,* moviendo el *ticket* a la columna *"*In Review*" de JIRA*. La creación de los *pull requests* hacía posible revisar y aprobar los cambios realizados por un miembro del equipo antes de fusionarlos con la rama develop.

Además, al crear PRs, se generaba la posibilidad de iniciar una conversación en la que todos los integrantes tenían la oportunidad de seguir y proponer modificaciones en el código mediante comentarios. Esto nos ayudaba a identificar cualquier error o posible problema antes su integración a la rama *develop*,

contribuyendo a reducir el retrabajo y a mantener el código más limpio y organizado. También, en esta instancia, verificábamos que todos adoptáramos el mismo estándar de codificación mencionado anteriormente y que se evitaran errores de comprensión de negocio. El responsable de la creación de este, debía seleccionar lo siguiente en github:

- La rama de origen.
- La rama de destino.
- Una descripción de los cambios basada en un *template*, en la cual se debe especificar qué cambios se realizaron, adjuntar el ticket asociado, y proporcionar una lista de verificación de tareas que deben completarse antes de fusionar.
- Designar revisores del *pull request*

  En la siguiente ilustración mostramos el listado de los *pull request* generados en el repositorio tu-bocado-be

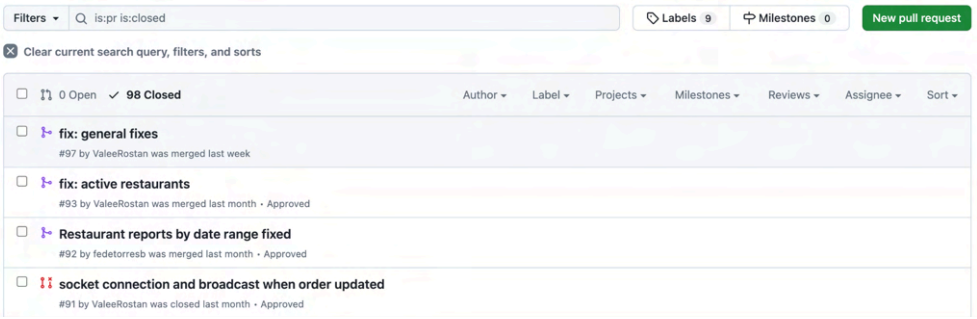


Ilustración 8.5.1: ejemplo de pull requests de TuBocado.TuBocadoBE

Ahora se presenta un ejemplo de un *pull request,* el cual incluye una descripción basada en el *template* comentado anteriormente, junto con ejemplos de comentarios realizados en él:

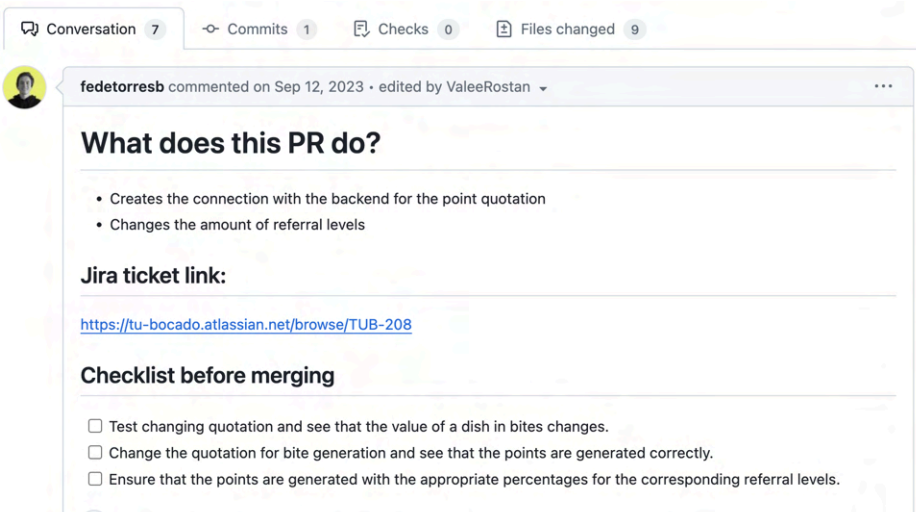


Ilustración 9.5.2: ejemplo de descripción de pull request



Ilustración 8.5.3: ejemplo de revisión de código

Una vez revisado y aprobado, el responsable de la creación del *pull request*, es encargado de fusionar la rama con *develop* mediante la técnica de “squash and merge”. Este método se basa en consolidar todos los commits en un único commit representativo y descriptivo para poder mantener un registro claro y conciso en la rama *develop*, facilitando la identificación de los cambios que serán trasladados a producción al momento de fusionar esta rama con *main*.

Por último, tras llevar a cabo las pruebas que detallamos en el capítulo de Pruebas, procedemos a aplicar el mismo procedimiento para fusionar la rama *develop* con *main*, lo que desencadena el despliegue automático una vez completada la fusión.

9. Métricas

En este capítulo, realizamos un análisis detallado de las métricas recopiladas durante el desarrollo del proyecto. Desde el inicio, definimos una serie de indicadores clave para medir el progreso del proyecto, comprender nuestra posición actual y, posteriormente, reflexionar sobre nuestros logros y la forma en la que trabajamos. Estos indicadores nos permitieron, en caso necesario, tomar alguna medida para no alejarnos del rumbo del proyecto.

La mayoría de estas métricas forman parte de la gestión de objetivos del proyecto, actuando como los Key Results de nuestros OKRs. Establecimos estos "KR" al inicio del proyecto al momento de definir los OKRs y nos permitieron monitorear y evaluar nuestros objetivos. En términos generales, utilizamos estos resultados y metas para planificar *Releases*, *Sprint* y tomar acciones correctivas. Las métricas que forman parte de los Key Results de los objetivos van desde el 10.1 hasta el 10.14 inclusive, y en cada una de ellas se encuentra el objetivo al que se asocian.

Para cada métrica, proporcionamos una descripción detallada, la meta establecida, el método de cálculo, su utilidad y si tomamos alguna medida en función de los resultados. Además, incluimos los resultados obtenidos, gráficos cuando sean pertinentes y un análisis detallado de los mismos.

9.1. Cantidad de defectos graves pendientes

Descripción: métrica que representa el número de defectos graves que se encuentran en el sistema. Como "defecto grave" entendemos que es: un error que impida el correcto desempeño de alguna funcionalidad del sistema.
Meta: 0.
Cálculo: como resultado de las pruebas de aceptación con usuario, en donde según diferentes casos de prueba los usuarios prueban el sistema, fuimos obteniendo un conteo del número de defectos graves existentes por cada release.
Uso / Decisiones en base a esta métrica: utilizamos la métrica utiliza con el fin de tener un sistema que cumpla con los requerimientos funcionales, que sea confiable

y que se comporte de la manera esperada. Por ejemplo: el flujo de compra de comida funciona correctamente.

Resultado: 0.

Gráfica:

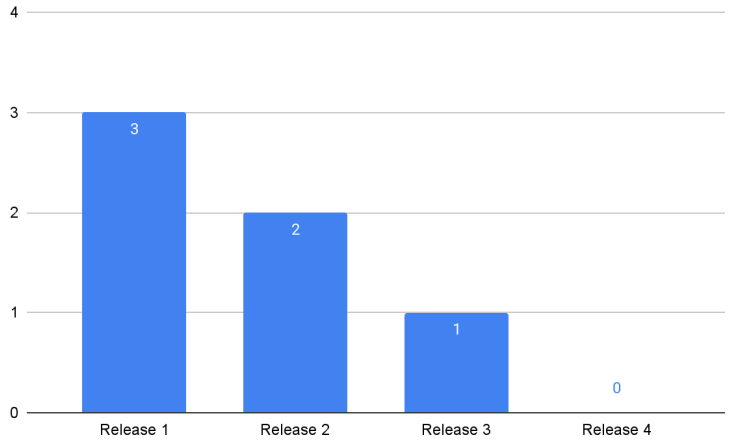


Ilustración 9.1.1: Grafica cantidad de defectos graves

Análisis gráfica: en el gráfico anterior observamos cómo el sistema comenzó teniendo en el primer release 3 defectos graves, pero que con el correr del proyecto, por cada release fuimos reduciendo esta cantidad hasta llegar a 0 en el release final. La pendiente refleja una tendencia negativa durante el desarrollo que llega hasta el cumplimiento de la meta de esta métrica. Podemos decir que estos resultados reflejan cómo pudimos tomar los resultados intermedios de 3, 2 y 1 número de defectos que se alejaban de la meta, y resolverlos permitiendo el correcto funcionamiento del sistema.

Referencia a OKR: 1. Entregarle un producto de calidad al cliente y que le proporcione un valor añadido.

9.2. Cantidad de defectos medios y leves pendientes

Descripción: métrica que refleja el número de defectos medios y leves que existen en el sistema. Como “defecto medio” entendimos que es un funcionamiento inesperado del sistema y que se aleja de los requisitos, y como un defecto “leve” nos referimos a alguna imperfección básica que no afecta significativamente a la funcionalidad sino que algo más sencillo.

Meta: menor a 10.

Cálculo: como resultado de las pruebas de aceptación con usuario, en donde según diferentes casos de prueba los usuarios prueban el sistema, fuimos obteniendo un conteo del número de defectos medios y leves existentes por cada release.

Uso / Decisiones en base a esta métrica: utilizamos la métrica con el fin de lograr un sistema que sea confiable y que cumpla estándares mínimos y generales de calidad.

Resultado: 3.

Gráfica:

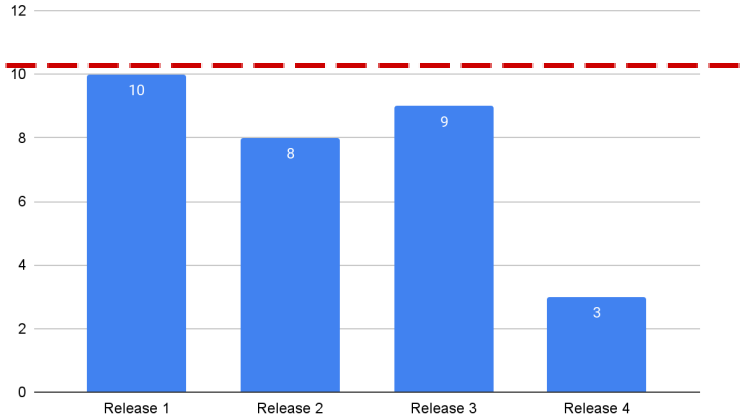


Ilustración 9.2.1: Grafica cantidad de defectos medios y leves

Análisis gráfica: en la gráfica anterior se observa como el número de defectos en cuestión va fluctuando y tanto aumenta como disminuye en el correr del proyecto. Comienza con 10 en el release 1, luego disminuye a 8 en el release 2 para luego disminuir y terminar con 3 en el último release. En todo momento logramos mantenernos por debajo de 10 cumpliendo así con la meta estipulada, y además, terminamos el último release con un valor relativamente bajo.

Referencia a OKR: 1. Entregarle un producto de calidad al cliente y que le proporcione un valor añadido.

9.3. NPS “¿Cuán probable es que recomiendes nuestro producto a un amigo o colega?”, con *Product Owner*

Descripción: esta métrica representa un indicador de la satisfacción del *product owner* con el producto desarrollado. Es un NPS calculado en base a una encuesta con la pregunta "¿qué tan probable es que recomiendes nuestro producto a un

amigo o colega?" con las posibles respuestas del 1 al 10. Es decir, el 1 representa “completamente insatisfecho” y 10 “completamente satisfecho”.

Meta: mayor o igual a 50%.

Cálculo: para calcular este indicador, utilizamos la fórmula proporcionada por la literatura de NPS (“Net Promoter Score”) en donde se define los valores del rango de la encuesta que son “detractores”, “pasivos” y “promotores” para luego restar el porcentaje sobre el total de “promotores” y “detractores”. A continuación se muestra la fórmula: NPS = Promotores(%) - Detractores(%)

En este caso, los valores del 1 al 6 son considerados “detractores”, del 7 al 8 “pasivos” y del 9 al 10 son “promotores”. Entonces, realizamos la encuesta al *product owner*, y según lo que él votó en la encuesta, calculamos el NPS. En este caso, se da una situación particular donde el 100% de los resultados depende de un único voto del *product owner,* entonces el NPS siempre va a reflejar la única respuesta del involucrado. Se puede ver más en detalle la realización del cálculo en cálculo key results NPS “¿Cuán probable es que recomiendes…” al PO.

Uso / Decisiones en base a esta métrica: utilizamos esta métrica con el objetivo de ir monitoreando y entendiendo la opinión del cliente sobre el producto que desarrollabamos. Este NPS junto con las reuniones con el PO nos guiaron para realizar ajustes necesarios según la conformidad del cliente.

Resultado: 100%.

Gráfica:

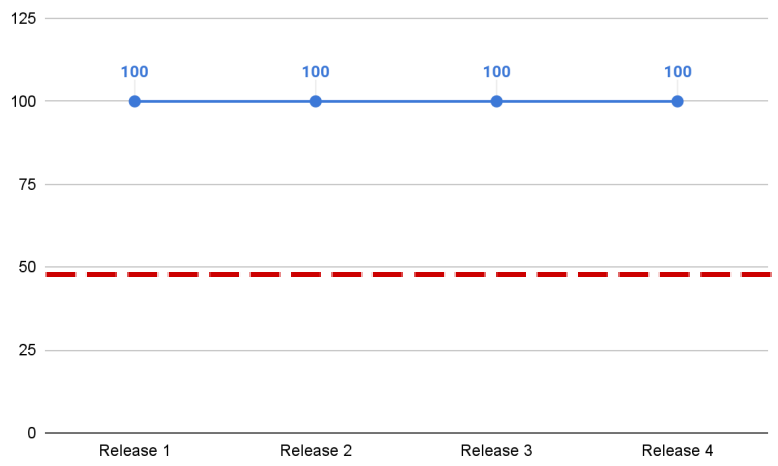


Ilustración 9.3.1: Gráfica NPS con P.O

Análisis gráfica: la gráfica muestra una meseta en el 100% que refleja cómo a lo largo de todo el proyecto, el *product owner* nos transmitió, según el NPS una satisfacción del producto muy buena al mantenerse en el valor más en cada release.
Referencia a OKR: 1. Entregarle un producto de calidad al cliente y que le proporcione un valor añadido.

9.4. NPS "¿Cuán probable es que recomiendes nuestro producto a un amigo o colega?", con 10 Consumidores

Descripción: esta métrica es una copia idéntica de la anterior con la excepción de que realizamos la encuesta a una muestra de 10 consumidores. Esta también la manejamos como NPS y mantuvimos la encuesta con la misma escala. Además, mantuvimos los valores eran "detractores", "pasivos" y "promotores" pero la única diferencia fue la muestra de 10 usuarios.
Meta: mayor o igual a 50%.
Cálculo: mantuvimos el cálculo de este NPS exactamente igual al anterior, con la misma fórmula y escalas. La muestra de 10 usuarios (5 de ellos obtenidos luego de las pruebas de aceptación con usuarios) permitió un índice más variado y abaractivo desde la perspectiva de los consumidores. Se puede ver más en detalle la realización del cálculo en Cálculo key results NPS "¿Cuán probable es que recomiendes…" a consumidores..
Uso / Decisiones en base a esta métrica: el uso que le dimos a esta métrica está relacionado con el entendimiento de la perspectiva de los consumidores sobre la aplicación. Los consumidores representan al grupo de usuarios más grande del sistema y son quienes usarían el *market* de comida, lo que es una parte muy importante del mismo. Esta métrica permitió empatizar con el consumidor y tomar algunas críticas para ajustar la aplicación a algo más atractivo y *user friendly*.
Resultado: 80%.

Gráfica:

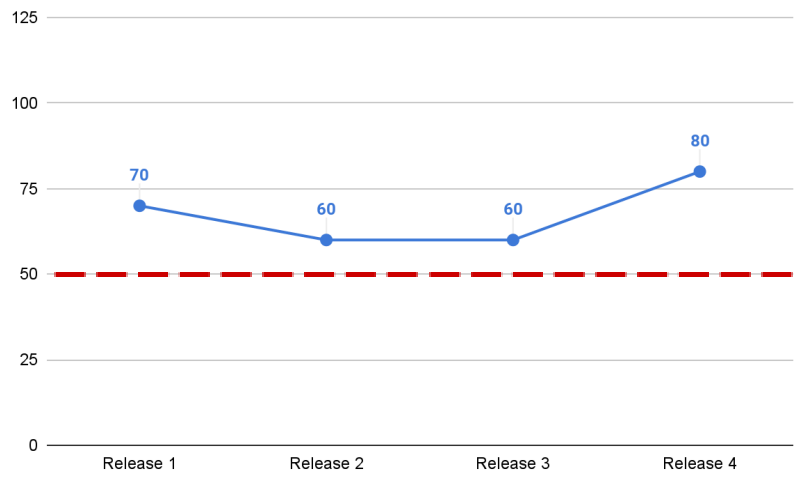


Ilustración 9.4.1: Gráfica NPS con Consumidores

Análisis gráfica: en este caso la gráfica nos permite ver como en el primer release el NPS dio un 70%, superando por un 30% la meta, pero luego, en los siguientes dos releases, bajo a un 60% reduciendo así el margen con la meta. Finalmente, en el release 4, el NPS de satisfacción del producto con clientes aumentó a un 80% superando la meta establecida y reflejando una buena valoración por parte de dichos usuarios.

Referencia a OKR: 1. Entregarle un producto de calidad al cliente y que le proporcione un valor añadido**.**

9.5. NPS “¿Cuán probable es que recomiendes nuestro producto a un amigo o colega?”, con 5 Dueños de restaurantes

Descripción: nuevamente utilizamos el NPS con la misma encuesta y escala, pero en este caso fue realizada a una muestra de 5 dueños de restaurantes al finalizar cada release.

Meta: mayor o igual a 50%.

Cálculo: la forma en la que se calcula es igual a la anterior. Se puede ver más en detalle la realización del cálculo en cálculo key results NPS “¿Cuán probable es que recomiendes…” a dueños de restaurantes.

Uso / Decisiones en base a esta métrica: este índice nos permitió comprender a alto nivel la visión de los dueños de restaurantes sobre el back office de administración de restaurantes. Entendíamos que era muy importante medir la

opinión de los dueños y empatizar con ellos teniendo en cuenta su clave participación en el sistema y que van a ser los clientes y socios clave de nuestro cliente. Junto con esta métrica y los diferentes feedbacks que nos fueron dando, tomamos diferentes medidas para mejorar la experiencia de los managers de restaurantes en el sistema mencionado.

Resultado: 100%.

Gráfica:

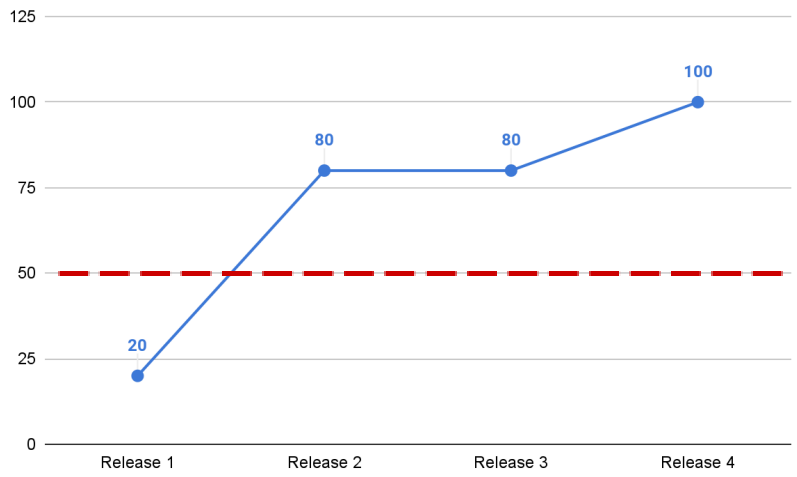


Ilustración 9.5.1: Gráfica NPS con Restaurantes

Análisis gráfica: la gráfica de este NPS se ve con una tendencia al aumento a lo largo de los 4 releases. Primero un aumento, luego meseta y finalmente un último crecimiento. Al comienzo, en el release 1 el valor obtenido fue significativamente negativo, estando por debajo de un 30% de la meta determinada. De todas formas, en el siguiente release pudimos remediar esto y logramos un salto grande a un 80% en el release 2 y 3, para finalmente en el release 4 obtener un 100%. Creemos que el bajo valor obtenido al principio está relacionado con las pocas funcionalidades desarrolladas para ese entonces y lo básico que se veía el sistema para ellos, y que luego, acompañado de una mayor implementación de funcionalidades relacionadas con el manejo de restaurantes, su opinión fue mejorando.

Referencia a OKR: 1. Entregarle un producto de calidad al cliente y que le proporcione un valor añadido**.**

9.6. Índice de satisfacción del cliente con el producto

Descripción: esta métrica es un índice que se genera a partir de una encuesta al *product owner* y está enfocada en medir el punto de vista del cliente sobre el producto que desarrollamos.

Meta: mayor o igual a 4.

Cálculo: para obtener este índice, realizamos encuestas de satisfacción al *product owner* al finalizar cada release en donde le preguntabamos "¿Qué tan satisfecho estás con el producto en cada una de estas áreas de enfoque?" siendo las áreas de enfoque: rendimiento y velocidad, facilidad de uso, cumplimiento de requerimientos y evaluación general. El posible valor de cada área es de un rango del 1 al 5 siendo 1 "muy poco satisfecho" y 5 "extremadamente satisfecho". El valor del *Key Result* que tomamos como valor final por cada release es el promedio de los resultados en cada área de enfoque. Se puede apreciar con detalle el cálculo en encuesta satisfacción del cliente con producto - 4 temáticas.

Uso / Decisiones en base a esta métrica: usamos esta métrica para entender si el producto cumplía con las expectativas del cliente y si él mismo era un producto de calidad o no. En base a los resultados fue que tomamos alguna medida de corrección tomando el feedback del PO al contestar en cada una de las áreas.

Resultado: 5.

Gráfica:

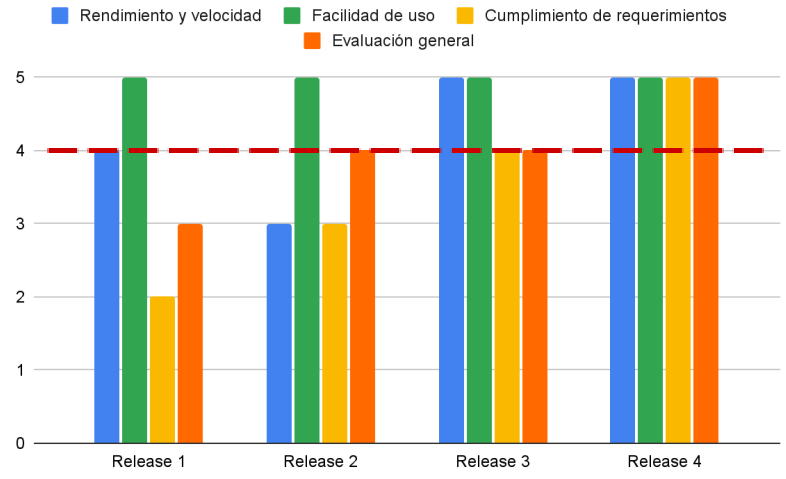


Ilustración 9.6.1: Gráfica índice satisfacción del cliente con producto

Análisis gráfica: la gráfica anterior expone cómo las diferentes áreas de enfoque de la encuesta fueron teniendo un comportamiento variado a lo largo del proyecto. Por un lado, la facilidad de uso fue estable y se mantuvo con la mejor puntuación en los 4 releases, mientras que el cumplimiento de requerimientos fue de muy bajo a terminar muy alto al final del desarrollo. Esto último es proporcional con las fuentes que fuimos realizando y que el encuestado podría observar en el momento de hacer la encuesta. Además, a partir del release 3 cumplimos con el objetivo de mínimo un valor de 4 promedio, y luego en el release 4 llegamos a un promedio de 5 superando así la meta por 1 punto.

Referencia a OKR: 1. Entregarle un producto de calidad al cliente y que le proporcione un valor añadido.

9.7. Índice de satisfacción del cliente con el equipo

Descripción: este *Key Result* representa la satisfacción del cliente con el equipo de trabajo. El concepto anterior de la realización de encuestas con diferentes temáticas a evaluar se mantiene, pero en este caso enfocadas al equipo. Realizabamos las encuestas también al finalizar cada release, y el rango era el mismo de 1 al 5. Las temáticas en este caso son: trato profesional y de respeto, puntualidad, claridad en la comunicación, compromiso del equipo y adaptación a los cambios.

Meta: mayor o igual a 4.

Cálculo: misma forma que la métrica anterior con las nuevas áreas de enfoque y la pregunta: “¿Qué tan satisfecho estás con la forma en que trabajamos como equipo de desarrollo de *software* en cada una de estas áreas de enfoque?”. Se puede observar el detalle del cálculo en encuesta satisfacción del cliente con equipo - 5 temáticas.

Uso / Decisiones en base a esta métrica: este indicador lo utilizamos para medir la forma en la que funcionamos como equipo de trabajo. Buscábamos entender si íbamos por el camino correcto en nuestra relación con el cliente y la forma en la que trabajamos o si deberíamos cambiar algo para corregir y mejorar.

Resultado: 5

Gráfica:

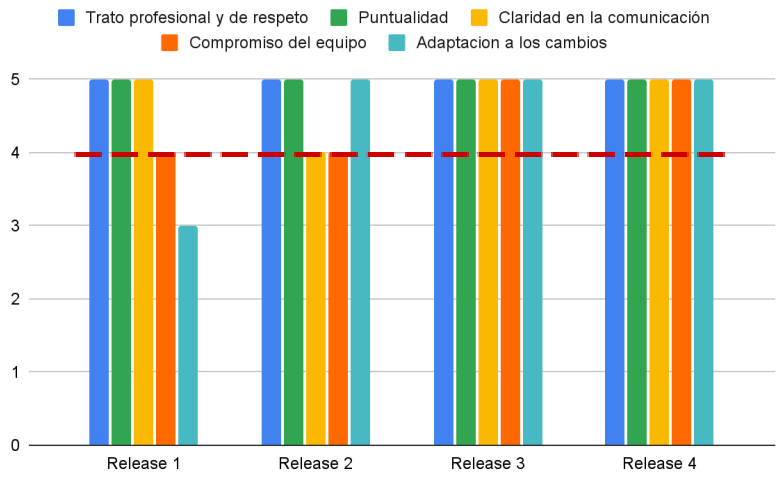


Ilustración 9.7.1: Gráfica índice satisfacción del cliente con equipo

Análisis gráfica: en esta gráfica observamos como a nivel general, tuvimos una buena calificación como equipo. El promedio de los resultados en cada temática por cada release fue superior a la meta en los 4 releases. En lo que fue el relacionamiento con el cliente, tuvimos un muy buen resultado y estable en el tiempo, mientras que en áreas como adaptación a los cambios fue algo que creció de un bajo resultado (3) en el primer release a 5 en los siguientes.

Referencia a OKR: 1. Entregarle un producto de calidad al cliente y que le proporcione un valor añadido.

9.8. Cumplimiento de las 10 heurísticas de nielsen

Descripción: por ser una POC, uno de los atributos de calidad a priorizar durante el proyecto fue el de usabilidad dado que es gran parte de lo que ven los usuarios como se explicó anteriormente en el capítulo de arquitectura. En este sentido, optamos por seguir las heurísticas de nielsen y fue de gran ayuda tener una métrica que nos permita medir y evaluar el número de heurísticas cumplidas. Esta métrica representa eso, la cantidad de heurísticas que fuimos cumpliendo en cada release.

Meta: mayor o igual a 80%.

Cálculo: luego de cada release todos los integrantes del equipo de forma asíncrona nos encargabamos de analizar el sistema con las nuevas *features* implementadas y, utilizando como *checklist* las heurísticas, marcabamos cual se cumplía o no. Después, poníamos en común entre todos y discutíamos si había inconsistencias

entre las respuestas de cada uno. De esta forma obtuvimos el número de cuentas cumplimos y cuantas no por cada release. Se puede observar en detalle el cálculo en cálculo key result cumplimiento heurísticas de nielsen.

Uso / Decisiones en base a esta métrica: esta métrica nos perimitó entender dónde estábamos parados en relación a la usabilidad. Sirvió como referencia para ver si en relación al atributo de calidad en cuestión estábamos bien posicionados o debíamos alertarnos y comenzar a mejorar la UX/UI siguiendo las reglas.

Resultado: 100%.

Gráfica:

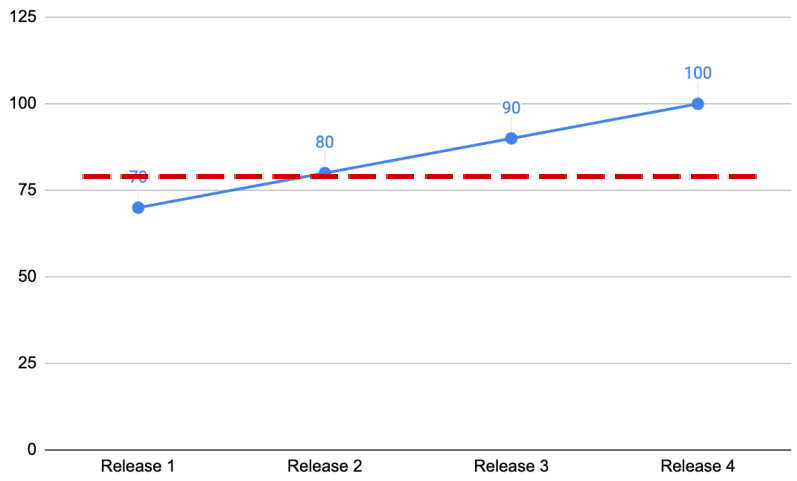


Ilustración 9.8.1: Gráfica cumplimiento heurísticas de nielsen

Análisis gráfica: en la gráfica obtenida observamos una tendencia positiva con un crecimiento lineal desde un 70% del cumplimiento de las heurísticas hasta un 100% en el último release. Esto refleja una mejora en la usabilidad y cumplimiento de estas reglas que ayudan a tener un producto de mejor calidad. La meta fue superada desde el release 2 en adelante.

Referencia a OKR: 1. Entregarle un producto de calidad al cliente y que le proporcione un valor añadido.

9.9. Desviación en la estimación del tamaño

Descripción: esta métrica refleja el porcentaje de desviación que existe entre el tamaño real que llevó realizar una iteración en *Story Points* y el tamaño que se estimó para realizar las *User Stories* el cual también está en *Story Points.* Por cada

iteración tenemos que porcentaje de desviación existe entre lo que estimamos y lo que realmente nos llevó.

Meta: tendencia no ascendente.

Cálculo: luego de cada iteración realizamos el cálculo de la desviación de tamaño utilizando la siguiente fórmula y obteniendo los datos de la suma de los *Story Points* estimados y que realmente nos llevó realizar cada *User Story*. A continuación la fórmula:

$$(1 - (\text{Tamaño Real} / \text{Tamaño Estimado})) * 100$$

Se pueden ver más detalles sobre los datos que utilizamos para el cálculo de esta métrica en A.19. Historias de usuario estimadas - hoja de cálculo.

Uso / Decisiones en base a esta métrica: la desviación del tamaño fue utilizada en cada *Release Planning* y *Sprint Planning* para analizar qué tan precisos y realistas estábamos siendo al momento de estimar y ver de una manera más adecuada que *User Stories* podíamos tomar en la siguiente iteración. En caso de tener una desviación muy grande, intentábamos identificar porque estábamos estimando mal y mejorar la estimación siguiente.

Resultado: tendencia no ascendente.

Gráfica:

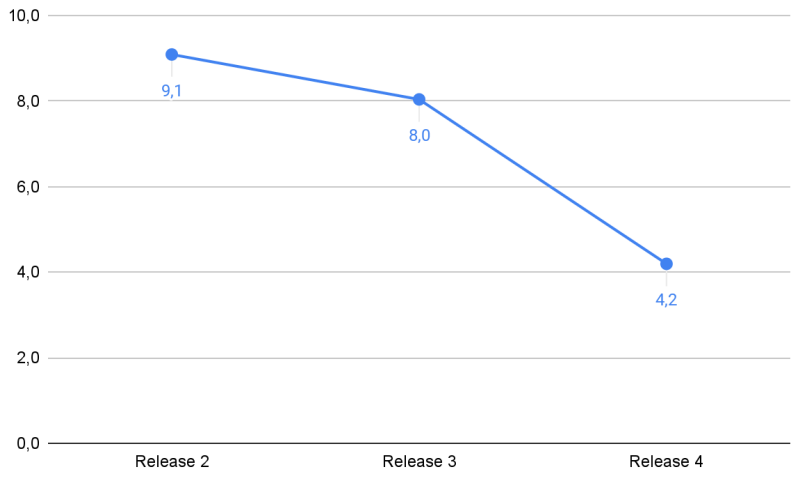


Ilustración 9.9.1: Gráfica desviación estimación tamaño

Análisis gráfica: la gráfica refleja el porcentaje de desviación entre el tamaño estimado y el real por cada release que va en caída. A medida que avanzó el proyecto fuimos mejorando nuestras habilidades para estimar y fuimos siendo más

precisos al momento de hacerlo. Esto refleja que cumplimos con la meta establecida teniendo una tendencia no ascendente que se ve con la pendiente negativa en la gráfica.
Referencia a OKR: 2. Establecer un proceso eficiente y adecuado para el proyecto.

9.10. Desviación en la estimación de esfuerzo

Descripción: esta métrica refleja el porcentaje de desviación que existe entre el esfuerzo real que tomó realizar cada iteración en horas-persona y el esfuerzo que estimamos para realizar las *User Stories* el cual también está en horas. Por cada iteración tenemos que porcentaje de desviación existe entre lo que estimamos y lo que realmente nos llevó.
Meta: tendencia no ascendente.
Cálculo: luego de cada iteración realizamos el cálculo de la desviación de esfuerzo utilizando la siguiente fórmula y obteniendo los datos de la suma de las horas-persona estimadas y que realmente nos llevó realizar cada *User Story*. A continuación la fórmula:

$$(1 - (\text{Esfuerzo Real} / \text{Esfuerzo Estimado})) * 100$$

Se pueden ver más detalles sobre los datos utilizados para el cálculo de esta métrica en A.19. Historias de usuario estimadas - hoja de cálculo.

Uso / Decisiones en base a esta métrica: la desviación del esfuerzo fue utilizada en cada *Release Planning* y *Sprint Planning* para analizar qué tan precisos y realistas estábamos siendo al momento de estimar en horas y ver, de una manera más adecuada, que *User Stories* podíamos tomar en la siguiente iteración. En caso de tener una desviación muy grande, intentábamos identificar porque estábamos estimando mal y mejorar la estimación siguiente.
Resultado: tendencia no ascendente.

Gráfica:

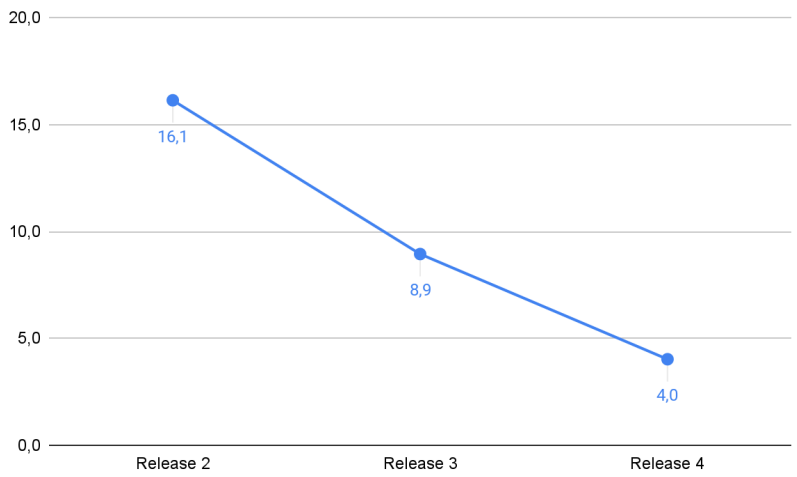


Ilustración 9.10.1: Gráfica desviación estimación esfuerzo

Análisis gráfica: la gráfica refleja el porcentaje de desviación entre el esfuerzo estimado y el real por cada release y como va en caída. A medida que avanzó el proyecto fuimos mejorando nuestras habilidades para estimar y fuimos siendo más precisos al momento de hacerlo. Esto refleja que cumplimos con la meta establecida teniendo una tendencia no ascendente que se ve con la pendiente negativa en la gráfica.

Referencia a OKR: 2. Establecer un proceso eficiente y adecuado para el proyecto.

9.11. Costo acumulado

Descripción: esta métrica indica el costo monetario del proyecto a nivel global en dólares con la relación de 1 hora igual a 1 dólar. Este es un indicador clave a tener en cuenta durante todo el proyecto ya que se debía ser muy cuidadoso con no pasarse del límite objetivo establecido al comienzo. El tema de financiación de un proyecto es un tema muy sensible y significativo en las negociaciones con el cliente y es algo que no puede tener mucho margen de error.

Meta: menor o igual a 3172.

Cálculo: para calcular el costo acumulado íbamos tomando los datos de tiempo dedicado por parte de todos los integrantes de cada mes, el cual se puede ver en el reporte de A.36.Trackeo de horas mensuales, y sumábamos al tiempo del mes anterior. De esta forma fuimos tomando los valores acumulados del tiempo, que según la relación mencionada antes equivale a el costo.

Además, es importante entender cómo se calcula el costo objetivo. Para hacerlo, en un principio seguimos la siguiente fórmula:

$$15 \text{ x } 4 \text{ x } 4 \text{ x } 12$$

(Horas semanales x Nro Integrantes x Semanas de un mes x Meses concurridos)

Las horas semanales por integrante destinadas al proyecto fue establecida en 15 horas. Para obtener el costo acumulado objetivo, multiplicamos este valor (15) por el número de integrantes (4), por la cantidad de semanas en un mes (4), por los meses transcurridos hasta alcanzar el valor deseado. Es decir, el aumento objetivo por mes se calcula suplantando el “Meses Concurridos” por 1 y eso da 240. Por otro lado, en el caso de ser 12 meses, el costo sería: $15\ X\ 4\ X\ 4\ X\ 12\ =\ 2880$.

Inicialmente, este objetivo de costo resultó en un aumento de 240 por mes, manteniéndose así hasta Agosto. Sin embargo, en ese punto notamos que los valores acumulados se desviaban de nuestro objetivo, por lo que decidimos realizar un ajuste. Calculamos el costo acumulado de horas hasta julio (1279) y lo dividimos por los 6 meses transcurridos desde el inicio del proyecto, obteniendo así el promedio de horas trabajadas por mes que fue de 214. En lugar de seguir incrementando el costo objetivo en 240 por mes, decidimos ajustarlo a 214 para reflejar una meta más realista. Esta modificación se mantuvo hasta Octubre, cuando la tendencia indicó que estábamos excediendo el costo objetivo. Por lo tanto, tomamos la decisión de volver a realizar el cálculo del promedio de los últimos 6 meses y esto nos llevó a modificar nuevamente el incremento mensual a 266 como objetivo. Además incluimos el mes de marzo de 2024 que involucró horas de documentación, por esta razón es que el costo objetivo total quedó en 3172 y no en 2880.

Uso / Decisiones en base a esta métrica: el resultado de esta métrica nos permitió ir entendiendo cuánto era el costo monetario hasta el momento y siempre ir midiéndolo en comparación con el costo objetivo. En caso de pasarnos tomábamos la decisión de reajustar el objetivo o tomar alguna medida en relación al tiempo dedicado por parte de los 4 integrantes.

Resultado: 3357.

Gráfica:

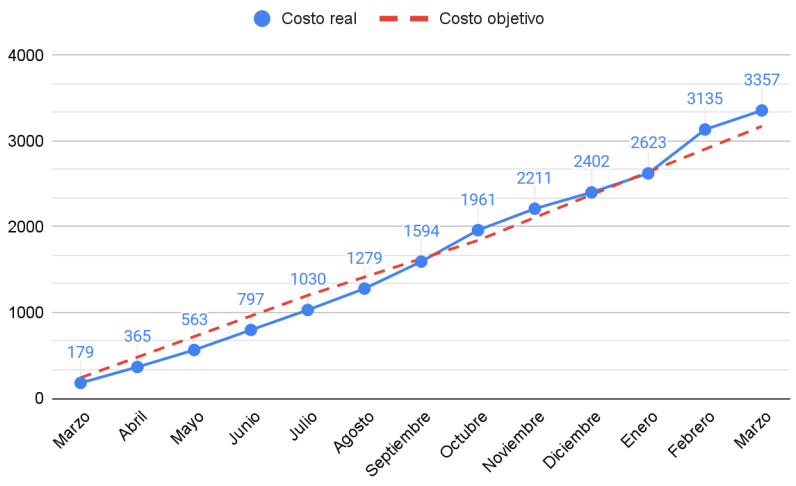


Ilustración 9.11.1: Gráfica costo acumulado

Análisis gráfica: la gráfica refleja cómo el costo real fue acompañando la tendencia del costo objetivo que calculamos en un inicio. Además, se pueden ver los leves ajustes y como hubo meses de mayor intensidad y meses de menos. Podemos ver como al final, nos terminamos pasando del costo objetivo ya que el costo objetivo era 3172 horas y terminamos realizando 3357 horas en febrero, esto se debe a que en febrero cambiamos de actividad y comenzamos a realizar la documentación final para el proyecto final lo que hizo que se genere un cambio grande en la cantidad de horas realizadas en dicho mes. En la sección de conclusiones hablamos más en detalle sobre este pico y porque realizamos los reajustes. En el anexo adjuntamos el reporte de clockify sobre la cantidad de horas mensuales.

Referencia a OKR: 2. Establecer un proceso eficiente y adecuado para el proyecto.

9.12. Grado de satisfacción (del 1 al 3) del equipo con el proceso de metodología ágil seleccionado

Descripción: esta métrica es un índice que refleja la satisfacción de los integrantes con el proceso de metodología ágil seleccionado. Expone la opinión del equipo sobre una parte significativa de la forma en la que estuvimos trabajando.

Meta: mayor a 2.

Cálculo: para calcular esta métrica, luego de cada release cada integrante del equipo nos encargábamos de contestar una encuesta del 1 al 3 enfocado con la

satisfacción con el proceso, siendo 1 “insatisfecho”, 2 “conforme” y 3 ”muy satisfecho”. En A.37. Encuestas de grado de satisfacción del equipo con metodología se pueden observar más detalles sobre el cálculo de esta métrica.

Uso / Decisiones en base a esta métrica: este indicador nos ayudó a entender que tan conforme se sentía cada uno con el proceso que estábamos desarrollando y si había una satisfacción a nivel general con la forma de trabajo que elegimos.

Resultado: 3.

Gráfica:

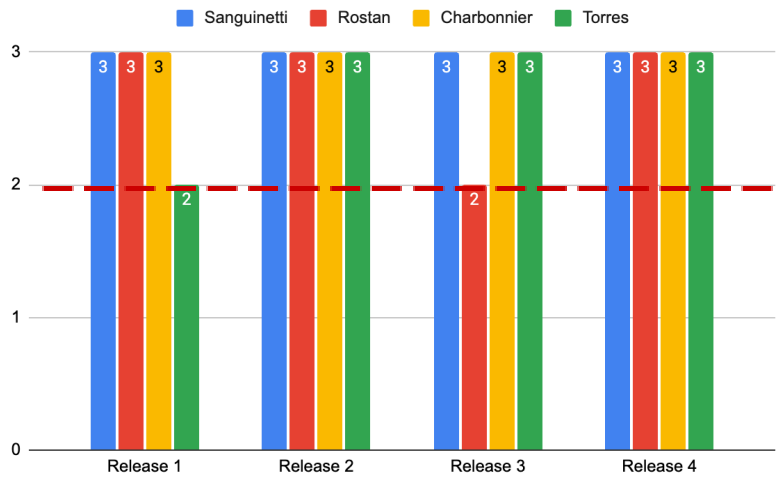


Ilustración 9.12.1: Gráfica satisfacción del equipo con metodología

Análisis gráfica: la gráfica muestra como en todo momento superamos la meta de tener un promedio de calificación mayor a 2 y como la mayoría de los integrantes y durante los 4 releases, votaron un valor alto en la escala planteada lo que demuestra una gran satisfacción del equipo.

Referencia a OKR: 2. Establecer un proceso eficiente y adecuado para el proyecto.

9.13. Aprender y aplicar correctamente nuevas metodologías claves de desarrollo durante la ejecución del proyecto

Descripción: esta métrica sirve como contador para listar el número de metodologías de desarrollo que fueron aprendidas y aplicadas durante la ejecución del proyecto.

Meta: mayor o igual a 2.

Cálculo: para obtener esta métrica básicamente identificamos cuales habían sido las metodologías de desarrollo que aprendimos y aplicamos hasta el momento.
Uso / Decisiones en base a esta métrica: este indicador permite contar la cantidad de metodologías de desarrollo clave que fueron aplicadas y aprendidas en determinado momento, y así poder planificar y ver cuánto esfuerzo le dedicamos a hacerlo.
Resultado: 4.
Referencia a OKR: 3. Aprender nuevas tecnologías.

9.14. Adquirir conocimiento y habilidades en las tecnologías clave utilizadas en el desarrollo de la aplicación, como ReactJs, NodeJs, Azure y AWS

Descripción: esta métrica es un indicador relacionado con el conocimiento y habilidades de cada integrante sobre las tecnologías que utilizamos para el desarrollo del proyecto.
Meta: mayor o igual a 4.
Cálculo: para poder obtener esta métrica, nos enfocamos en ver qué actividad o enfoque de participación podía tener cada integrante, en donde si entendíamos que alguno ya era suficientemente capaz de continuar alguna tarea relacionada con determinada tecnología significa que ya tenía el conocimiento suficiente y las habilidades necesarias. Luego de cada release, analizabamos que había realizado cada uno y cuáles tecnologías estaban involucradas, de ahí obtenemos el número.
Uso / Decisiones en base a esta métrica: este indicador nos ayudó a entender que tantas habilidades técnicas veníamos aprendiendo y si debíamos capacitarnos en alguna en específico. De este valor, fue que algunos tomaron la decisión de dedicar tiempo a la capacitación en una de las tecnologías.
Resultado: 4.
Referencia a OKR: 3. Aprender nuevas tecnologías.

9.15. Distribución del esfuerzo

Descripción: Esta gráfica muestra la distribución porcentual del esfuerzo total del equipo entre distintas áreas.

A lo largo de la duración del proyecto, llevamos un registro detallado de las horas trabajadas mediante la herramienta Clockify, donde cada miembro del equipo anotó el tiempo invertido junto con una breve descripción de la tarea y una etiqueta que representaba la categoría a la que pertenecía. Este seguimiento nos permitió llevar un seguimiento del esfuerzo que realizamos en cada área para poder luego tomar decisiones en base a estos. Posteriormente, gracias a que Clockify permite sacar reportes, exportamos un gráfico para visualizar la distribución del esfuerzo invertido en cada una. A continuación, justificamos y explicamos la contribución de cada área:

1. Desarrollo: Esta categoría hace referencia a todas las horas dedicadas a todas las tareas de programación, pruebas unitarias, correcciones de revisiones de código y demás actividades directamente relacionadas con el desarrollo del *software*, las cuales están vinculadas a las columnas del tablero mencionado en 5.9. Delivery. Es el núcleo del proyecto, donde materializamos las funcionalidades y construimos el producto.
2. Documentación: Esta categoría hace referencia a todas las horas dedicadas a la elaboración de todos los documentos para la entrega final, además de la elaboración de documentos técnicos y cualquier otra documentación necesaria para el proyecto.
3. Gestión: Esta categoría hace referencia a todas las horas dedicadas a la planificación, seguimiento y control del proyecto, incluyendo la gestión de riesgos y la coordinación del equipo, como puede ser todas las Ceremonias de Scrum y todo lo relacionado a Kanban. Es crucial para garantizar que el proyecto se desarrolle según lo previsto.
4. Gestión de Calidad: Esta categoría hace referencia a todas las horas dedicadas a las actividades de aseguramiento y control de calidad mencionadas en el capítulo de 8.3.Actividades de aseguramiento de calidad, incluidas las pruebas de integración y sistemas, horas dedicadas a la generación de pruebas de aceptación para usuario.
5. Reuniones: Esta categoría hace referencia a todas las horas dedicadas a las reuniones de equipo que no están incluidas dentro de las ceremonias de Scrum, tanto por revisión de proyecto, con el cliente, con el tutor y entre los miembros del equipo.

6. Investigaciones: Esta categoría hace referencia a todas las horas dedicadas a la investigación para el desarrollo del proyecto. Esto va desde la aplicación de *Design Thinking* hasta la investigación de metodologías para el proyecto, nuevas tecnologías, spikes, etc.
7. Arquitectura: Esta categoría hace referencia a todas las horas dedicadas a el diseño y planificación de la arquitectura del sistema.
8. Capacitaciones: Esta categoría hace referencia a todas las horas dedicadas a la formación y el desarrollo profesional del equipo.

Cálculo: Para poder realizar el cálculo de la distribución del esfuerzo lo que hicimos fue realizar la suma de las horas registradas en Clockify por categoría, con un total acumulado de 3361 horas, esto Clockify lo hace solo, además de calcular el porcentaje de cada área.

Uso / Decisiones en base a esta métrica: Estos datos permiten evaluar si estamos invirtiendo correctamente el esfuerzo realizado, además nos ayuda a tomar decisiones informadas sobre el esfuerzo dedicado a cada área e identificar áreas de mejora y realizar ajustes de ser necesario.

Resultado: Obtuvimos como resultado un 4,8% en arquitectura, un 2,8% en capacitación, un 41,7% en desarrollo, un 16,4% en documentación, un 10,76% en gestión, un 4,96% en gestión de calidad, un 14,74% en investigación y un 3,7% en reuniones.

Gráfica:

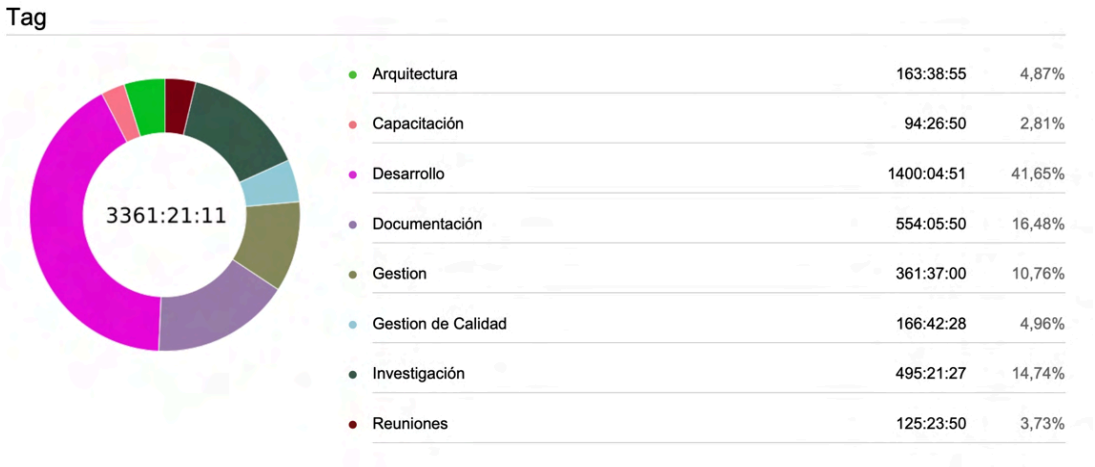


Ilustración 9.15.1: Gráfica distribución del esfuerzo

Análisis gráfica: Para poder realizar el análisis comparativo de la distribución realizamos un análisis de las prácticas recomendadas por el PMBOK Guide del Project Management Institute y las metodologías ágiles y las, adaptamos a las particularidades de nuestro proyecto para poder ver si nuestra distribución estaba dentro de lo correcto. El desarrollo está dentro del 40-50%, con un 42% lo que creemos que se alinea con lo esperado. Por otro lado, la documentación está un poco por fuera del 10-15%, con un 16%, esto se debe al tiempo invertido en la documentación final del proyecto. La gestión de proyecto nos dio alrededor de 11% lo que se encuentra dentro del 10-15%, por otro lado calidad 5% lo cual está dentro de lo esperado y el diseño de arquitectura con 5% también se encuentra alineado con lo analizado que es un porcentaje entre 2-8%. Por otro lado, creemos que la investigación está un poco por arriba de lo establecido en general, que es en un rango de 5-10%, esto se debe a que en la parte de investigación se incluyen muchos de los pasos realizados en *Design Thinking* y por eso esta mas aumentado de lo normal, esto refleja nuestro compromiso con la aplicación de *Design Thinking* y la exploración del *marketing multinivel*, creemos que esto fue lo que nos llevó a pasarnos un poco de la media. Por último, las reuniones y capacitaciones también se encuentran dentro de lo razonable que es entre 0% - 4%.

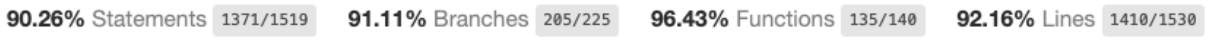
9.16. Cobertura de pruebas unitarias

Descripción: esta métrica otorga la cantidad de código que es testable de forma automática (mediante pruebas unitarias). La misma se realiza con Jest, la cual es una biblioteca que facilita la creación de pruebas unitarias y cuenta con una herramienta que otorga el porcentaje de cobertura de prueba. La herramienta analiza el código en cuatro áreas distintas y otorga un porcentaje de cobertura para cada una de ellas. Function coverage (cantidad de funciones probadas), statement coverage (cantidad de declaraciones probadas), branch coverage (cantidad de ramificaciones lógicas dentro de los métodos que se probaron) y line coverage (cantidad de líneas brutas de código que se probaron). [74]

Meta: mayor a 85% para cada una de las áreas.

Cálculo: para cada área el porcentaje representa el código ejecutado contra el código no ejecutado.

Uso / Decisiones en base a esta métrica: esta métrica nos permite realizar cambios con seguridad en el código sabiendo que la probabilidad de generar un error en el sistema es reducida. Además nos permitió corroborar que las user story estén completadas según el criterio del *definition of done* planteado.
Resultado: Statement coverage: 90.26%, Branch coverage: 91.11%, Funcion coverage: 96.43% y Line coverage: 92.16%.



El nivel de cobertura a la cual llegamos con el equipo está por encima de nuestro objetivo para cada uno de los sectores testeados. Estos resultados nos indican sobre la solidez y fiabilidad de nuestro código base, el cual fue probado exhaustivamente. Sin embargo, es importante mencionar que la cobertura de test no garantiza la ausencia total de errores. Seguiremos mejorando nuestros tests para garantizar la calidad del *software*. Podemos ver en detalle el resultado de la cobertura de pruebas unitarias en 9.16. Cobertura pruebas unitarias.

9.17. Retrabajo

Descripción: esta métrica mide el porcentaje de retrabajo, definido como el esfuerzo invertido en corregir defectos respecto al esfuerzo total dedicado al desarrollo. El cálculo de esta métrica permite evaluar la eficiencia del proceso de desarrollo y la calidad del trabajo inicial.
Meta: menor o igual a 20% del esfuerzo total de desarrollo.
Cálculo: el porcentaje de retrabajo lo calculamos de la siguiente manera:

$$\text{Retrabajo} = (\text{Esfuerzo Total de Desarrollo}/\text{Esfuerzo Total en Correcciones de Defectos}) \times 100$$

Se pueden observar de manera ordenada los resultados obtenidos sobre este cálculo de esta métrica en la sección A.38. Cálculo de retrabajo, así como también de donde obtuvimos los datos en A.19. Historias de usuario estimadas - hoja de cálculo.
Uso / Decisiones en base a esta métrica: en base a esta métrica pudimos ver el esfuerzo invertido por el equipo en el retrabajo, lo que nos permitió identificar áreas

del proceso de desarrollo que requerían mejoras para reducir la cantidad de defectos.

Resultado: el retrabajo fue del 19.5%, lo cual indica que hemos conseguido mantenernos dentro del objetivo de mantener el retrabajo por debajo del 20% del esfuerzo total de desarrollo.

Gráfica:

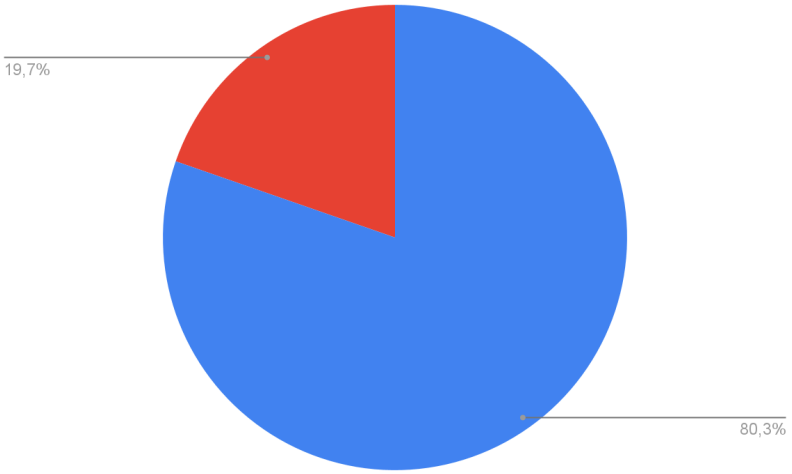


Ilustración 9.17.1: Gráfica retrabajo

Análisis gráfica: La gráfica muestra cómo el equipo logró mantener el retrabajo dentro del límite establecido, lo cual muestra la efectividad de las prácticas de desarrollo y control de calidad implementadas. Igualmente, creemos que estamos en el borde de la meta, es decir que, si bien la cumplimos estamos muy cerca del 20%, esto se debe a que desde un inicio sabíamos que íbamos a tener requerimientos cambiantes y al realizar pruebas iterativas fuimos encontrando varios y el PO fue cambiando de opinión en muchos casos en base a los insights de las pruebas. Con esto podemos asegurar que seguir una metodología ágil fue muy beneficioso para el proceso de nuestro proyecto ya que al tener retroalimentación rápida, pudimos prever varios cambios antes de que llegue a etapas tardías.

9.18. Velocidad

Descripción: la velocidad del equipo es la cantidad de story points de todas las user stories realizadas al momento de terminar el *Sprint*.

Cálculo: Para poder realizar este cálculo, sumamos el total de story points completados en un *Sprint* (los cuales se ven detallados en A.19. Historias de usuario estimadas - hoja de cálculo), y luego promediamos este total a lo largo de varios *Sprints* para obtener una medida consistente de la velocidad del equipo. Se pueden observar más detalles sobre el cálculo de esta métrica en A.39. Cálculo de la velocidad.

Uso / Decisiones en base a esta métrica: gracias a esta métrica pudimos identificar la productividad del equipo y ajustar las historias de usuario siguientes de mejor manera. Además, al poder saber la velocidad pudimos estimar de manera más eficiente la cantidad de historias de usuario que podíamos incluir en un *Sprint*.

Resultado: a lo largo de los *Sprints*, hasta el décimo *Sprint*, observamos una tendencia positiva en nuestra velocidad, lo que indica una mejora en la productividad y en la habilidad del equipo para manejar eficientemente el trabajo asignado. Luego los últimos *Sprints*, los dedicamos a realizar mejoras y agregar últimas mejoras que el PO nos iba solicitando por eso se ve un decremento en la velocidad.

Gráfica:

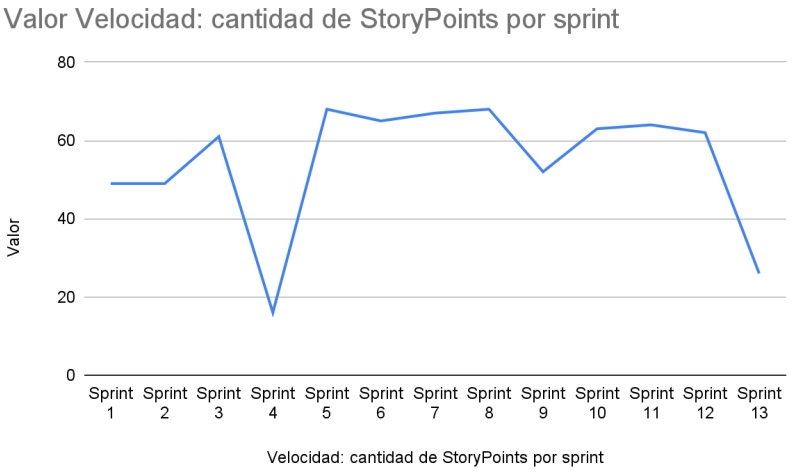


Ilustración 9.18.1: Gráfica velocidad

Análisis gráfica: en esta gráfica podemos observar como la velocidad del equipo fue aumentando a medida que iba pasando el tiempo. Podemos observar también algunos altos y bajos, particularmente en el *Sprint* 4, esto se debe a que, como mencionamos en la sección de 5.9.Delivery, debido a los cambios en el proveedor de infraestructura, tuvimos que agregar una nueva *Sprint* más corta para poder

realizar el deployment y terminar todas las pruebas de aceptación con los usuarios, además de encuestas y pruebas con el PO. Además, podemos ver una leve disminución en el *Sprint* 9, esto es debido a una mala estimación en el caso de una historia de usuario que luego deriva a una lección aprendida sobre la cantidad de esfuerzo máxima que debería llevar una historia de usuario (link a lección aprendida). Por otro lado, podemos observar cómo a partir del *Sprint* 12 la velocidad arranca a disminuir, esto se debe a que las últimas *Sprints* estaban enfocados en mejoras de la aplicación y corrección de errores por lo además de que ya no quedaban suficientes tareas en el backlog.

9.19. Productividad

Descripción: esta métrica mide el rendimiento del equipo en términos de cantidad de esfuerzo dedicado por cada Story Points entregado.
Meta: tendencia no ascendente.
Cálculo: Para poder realizar este cálculo, dividimos el esfuerzo real sobre el tamaño estimado. $\text{Productividad} = \text{Esfuerzo Real}/\text{Tamaño Estimado}$

Se pueden observar más detalles sobre el cálculo de esta métrica en la tabla con los valores A.40. Cálculo de la productividad, y para entender con más detalle el esfuerzo real y tamaño estimado por release se debe mirar A.19. Historias de usuario estimadas - hoja de cálculo.
Uso / Decisiones en base a esta métrica: en conjunto con la velocidad, gracias a esta métrica pudimos identificar la productividad del equipo y ajustar las historias de usuario siguientes de mejor manera. Al analizar la productividad, pudimos ajustar las asignaciones de user stories y como las estimamos. Además, esta métrica nos ayudó a establecer expectativas realistas para la planificación futura y a fijar objetivos de rendimiento basados en datos históricos.
Resultado: logramos un aumento significativo en la productividad del equipo. Ya que por cada Story Point logramos que se requiera menos esfuerzo.
Gráfica:

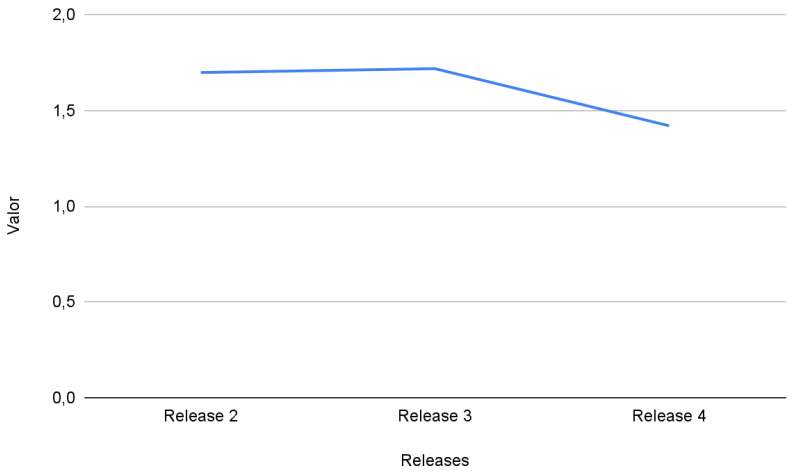


Ilustración 9.19.1: Gráfica productividad

Análisis gráfica: La gráfica de productividad muestra una tendencia descendente , lo cual indica que las medidas adoptadas para mejorar los procesos y la eficiencia del equipo han tenido un impacto positivo, ya que una tendencia descendente significa que cada vez se requiere menos esfuerzo para realizar story points.

10. Conclusiones

El capítulo de conclusiones tiene como finalidad principal realizar una evaluación de los objetivos establecidos al comienzo del proyecto y especificados en la introducción y analizar los resultados que obtuvimos una vez finalizado el del desarrollo del proyecto. Como se mencionó previamente en la 1. Introducción y en el apartado de 9. Métricas, los objetivos fueron definidos y gestionados utilizando la metodología de los *OKRs* (*Objectives and Key Results*). Al adoptar esta metodología, nuestro propósito fue garantizar que los objetivos fueran *SMART (Specific, Measurable, Attainable, Relevant and Timely)*.

De esta forma tuvimos la posibilidad de realizar un seguimiento continuo y efectivo de los objetivos, los cuales revisamos y actualizamos regularmente durante cada *Retrospective* de los diferentes *Release*, permitiéndonos documentar nuestro progreso de manera formal. Aunque el capítulo dedicado a las métricas profundizó en los resultados específicos de cada uno de los *Key Results* de los objetivos, en esta sección nos centraremos en evaluar el grado de cumplimiento de estos Key Results, y por ende de los objetivos, y las conclusiones que sacamos a raíz de dichos resultados.

10.1. Evaluación de Objetivos de Producto

1. Entregarle un producto de calidad al cliente y que le proporcione un valor añadido

Key Results	Meta	Resultado	Diferencia
a. Cantidad de defectos graves pendientes	0	0	0
b. Cantidad de defectos medios y leves pendientes	< 10	3	-6
c. NPS “¿Cuán probable es que recomiendes nuestro producto a un amigo o colega?”, con **Product Owner**	>= 50%	100%	+0.5
d. NPS “¿Cuán probable es que recomiendes nuestro producto a un amigo o colega?”, con una muestra de **10 Consumidores**	>= 50%	80%	+0.3
e. NPS “¿Cuán probable es que recomiendes nuestro producto a un amigo o colega?”, con una muestra de **5 Dueños de restaurantes**	>=50%	100%	+0.5
f. Índice de satisfacción del cliente con el producto [Encuestas de satisfacción en áreas de enfoque	>= 4	5	+1

del 1 al 5] i. Rendimiento y velocidad ii. Facilidad de uso iii. Cumplimiento de requerimientos iv. Evaluación general			
g. Índice de satisfacción del cliente con el equipo [Encuestas de satisfacción en áreas de enfoque del 1 al 5] i. Trato profesional y respeto ii. Puntualidad iii. Claridad en la comunicación iv. Compromiso del equipo v. Adaptación a los cambios	>= 4	5	+1
h. Cumplimiento de las 10 heurísticas de nielsen	>= 80%	100%	+0.2

Tabla 10.1.1: Key Results objetivo de producto N°1

1.a) Cantidad de defectos graves pendientes

Meta: 0.

Significado de meta: representa la cantidad de defectos graves en el sistema. Un defecto grave en el sistema significa un defecto que impide o limita el uso de alguna funcionalidad importante del sistema lo que hace que se vea afectado críticamente. La meta elegida es 0 ya que es importante que no exista nada que comprometa alguna de las funcionalidades relevantes del sistema. El objetivo es garantizar la total funcionalidad y confiabilidad del sistema.

Análisis de métrica: ver en detalle, sección 9. Métricas.

Evolución: para poder cumplir esta meta, al finalizar cada *Release* realizamos pruebas de aceptación con usuarios, restaurantes y el cliente para poder evaluar la calidad del sistema y detectar defectos graves. Una vez encontrados, buscamos priorizarlos agregándolos siempre en el primer *Sprint* del siguiente *Release* para asegurar la corrección de los mismos lo antes posible.

Resultado: cumplido.

1.b) Cantidad de defectos medios y leves pendientes

Meta: menor que 10.

Significado de meta: representa la cantidad de defectos medios y leves en el sistema. Un defecto medio o leve en el sistema significa un defecto que no cumple con los requerimientos del sistema o que afecta la experiencia del usuario o eficiencia del sistema. La meta elegida es menor a 10 ya que es importante reducir

la cantidad de estos defectos para mejorar la experiencia del usuario, pero también dejamos un margen de defectos para que el equipo pueda trabajar en otras funcionalidades más prioritarias de ser necesario.
Análisis de métrica: ver en detalle, sección 9. Métricas.
Evolución: al igual que con los defectos graves, para poder cumplir esta meta, al finalizar cada *Release* realizamos pruebas de aceptación con usuarios, restaurantes y el cliente para poder evaluar la calidad del sistema y detectar defectos. Una vez encontrados, realizábamos una priorización dependiendo del defecto encontrado para poder corregirlos en el siguiente *Release*.
Resultado: cumplido.

1.c) NPS "¿Cuán probable es que recomiendes nuestro producto a un amigo o colega?" *Product Owner*

Meta: mayor o igual a 50%.
Significado de meta: mide la disposición del cliente a recomendar el producto, reflejando su satisfacción y confianza en el mismo. Un NPS igual o superior al 50% indica una percepción positiva y un alto nivel de recomendación.
Análisis de métrica: ver en detalle, sección 9. Métricas.
Evolución: al finalizar cada *Release* se le realizaba esta pregunta al *product owner* para luego poder discutir la evolución del NPS y así destacar las acciones implementadas para mejorar la experiencia del mismo y aumentar su nivel de satisfacción. Además podemos observar cómo a lo largo de todo el proyecto, la satisfacción del cliente se mantuvo constante en 100%, lo que significa que pudimos lograr desde el inicio una buena satisfacción con el mismo.
Resultado: cumplido.

1.d) NPS "¿Cuán probable es que recomiendes nuestro producto a un amigo o colega?" Consumidores

Meta: mayor o igual a 50%.
Significado de meta: mide la disposición de los usuarios a recomendar el producto, reflejando su satisfacción y confianza en el mismo. Un NPS igual o superior al 50% indica una percepción positiva y un alto nivel de recomendación.
Análisis de métrica: ver en detalle, sección 9. Métricas.

Evolución: al finalizar cada release realizábamos esta pregunta a posibles consumidores de la aplicación para luego poder discutir la evolución del NPS y así destacar las acciones implementadas para mejorar la experiencia de los mismos y aumentar su nivel de satisfacción.
Resultado: cumplido.

1.e) NPS "¿Cuán probable es que recomiendes nuestro producto a un amigo o colega?" Restaurantes

Meta: mayor o igual a 50%.
Significado de meta: mide la disposición de los dueños de restaurantes a recomendar el producto, reflejando su satisfacción y confianza en el mismo. Un NPS igual o superior al 50% indica una percepción positiva y un alto nivel de recomendación.
Análisis de métrica: ver en detalle, sección 9. Métricas.
Evolución: al finalizar cada *Release* realizábamos esta pregunta al dueño de restaurantes de sushi para luego poder discutir la evolución del NPS y así destacar las acciones implementadas para mejorar la experiencia de los mismos y aumentar su nivel de satisfacción.
Resultado: cumplido.

1.f) Índice de satisfacción del cliente con el producto

Meta: mayor o igual a 4.
Significado de meta: este índice refleja la satisfacción del cliente en varias áreas seleccionadas por el equipo relacionadas con el producto, como rendimiento, facilidad de uso, cumplimiento de requerimientos y evaluación general, en una escala de 1 a 5. Una puntuación de 4 o más creemos que indica una percepción positiva sobre la satisfacción del producto.
Análisis de métrica: ver en detalle, sección 9. Métricas.
Evolución: a lo largo de todo el proyecto, la satisfacción del cliente fue variando. Se puede ver como hubo un aumento lineal con la satisfacción del cumplimiento de los requerimientos, lo cual tiene sentido ya que a medida que fuimos avanzando en los *Releases* la aplicación iba aumentando sus funcionalidades. Por otro lado, también podemos observar como la facilidad de uso siempre se mantuvo en 5 ya que

siempre buscamos priorizar la experiencia del usuario más allá de los requerimientos realizados ya que cada *Release* era un MVP y buscábamos que el usuario pudiera hacer las funcionalidades incluidas en el mismo, de la mejor manera. Por último, podemos ver como la evaluación general, a medida que se fue mejorando el rendimiento y aumentando las funcionalidades, fue aumentando hasta llegar finalmente a 5.
Resultado: cumplido.

1.g) Índice de satisfacción del cliente con el equipo

Meta: mayor o igual a 4.
Significado de meta: este índice refleja la satisfacción del cliente en varias áreas seleccionadas por el equipo relacionadas con el producto, como trato profesional, puntualidad, compromiso, comunicación y adaptación a los cambios, en una escala de 1 a 5. Una puntuación de 4 o más creemos que indica una percepción positiva sobre la satisfacción del cliente con el equipo.
Análisis de métrica: ver en detalle, sección 9. Métricas.
Evolución: a lo largo de todo el proyecto, la satisfacción del cliente fue variando pero siempre logramos cumplir la meta. Se puede ver como hubo un aumento con respecto a la capacidad del equipo en adaptarse a los cambios. Esto se debe a que, inicialmente aún estábamos aprendiendo a familiarizarnos con las tecnologías, la manera de estimar y las metodologías, pero rápidamente logramos adaptarnos logrando un 5 de satisfacción. Por lo tanto, podemos ver como la en general, a medida que se fue mejorando el relacionamiento con el cliente y el equipo, fue aumentando hasta llegar finalmente a 5.
Resultado: cumplido.

1.h) Cumplimiento de las 10 Heurísticas de Nielsen

Meta: mayor o igual a 80%.
Significado de meta: esta métrica evalúa la conformidad del producto con las 10 Heurísticas de usabilidad de Nielsen, que son principios fundamentales para una buena experiencia de usuario. Un cumplimiento de al menos el 80% indica que el producto es altamente usable y está bien diseñado desde la perspectiva de la usabilidad.

Análisis de métrica: ver en detalle, sección 9. Métricas.
Evolución: podemos observar cómo esta métrica experimentó una mejora progresiva en su porcentaje. Al inicio, durante el primer *Release*, comenzamos con un 70% de cumplimiento, un resultado que no cumplía con nuestras expectativas. Este valor inicial lo atribuimos a la ausencia de elementos clave como *modals* de confirmación, efectos de hover y otros tipos de retroalimentación visual para los usuarios. Luego, ya en el segundo *Release*, logramos incrementar este porcentaje, lo que refleja cómo las intervenciones y mejoras en el diseño a medida que iban pasando los releases, fueron mejorando la usabilidad del producto.
Resultado: cumplido.

En conclusión, el objetivo de entregar un producto de calidad que agregue valor significativo al cliente fue muy importante para acciones y decisiones a lo largo del proyecto, sobre alcance, equipo, tiempos. Además, fue el que nos guió principalmente a nuestra estrategia de aseguramiento de calidad, lo que nos llevó a crear encuestas para medir la satisfacción tanto a los consumidores, como restaurantes y al cliente. También nos llevó a realizar pruebas de aceptación en colaboración con el cliente después de cada *Release*. Estas iniciativas fueron muy importantes para evaluar la calidad de lo entregado, identificar oportunidades de mejora y corregir errores de manera inteligente.

Logramos alcanzar todas las metas propuestas, logrando una alta satisfacción con el resultado final del producto. Creemos que nuestro equipo se distinguió por su alto profesionalismo y dedicación, aspectos que se pueden ver reflejados en las métricas de satisfacción del equipo. Además, creemos que, más allá de cumplir con las métricas para este objetivo, logramos aportar un valor añadido significativo para el cliente. La colaboración y el enfoque de nuestro equipo fueron clave para ayudar a definir claramente el alcance del proyecto y asegurar que el resultado final estuviera perfectamente alineado con las necesidades y deseos del cliente.

10.2. Evaluación de Objetivos de Proceso

2. Establecer un proceso eficiente y adecuado para el proyecto

Key Results	Meta	Resultado	Diferencia
a. Desviación en la estimación del tamaño	No ascendente	No ascendente	-
b. Desviación en la estimación de esfuerzo	No ascendente	No ascendente	-
c. Costo acumulado	<= 3172	3357	-185
d. Grado de satisfacción (del 1 al 3) del equipo con el proceso de metodología ágil seleccionado	> 2	3	0

Tabla 10.2.1: Key Results objetivo de proceso N°2

2.a) Desviación en la estimación del tamaño

Meta: no ascendente.
Significado de meta: en esta métrica evaluamos la precisión que tuvo el equipo en la estimación del tamaño de las historias de usuario realizadas. El objetivo de mantener una tendencia no ascendente en las desviaciones busca garantizar que, a medida que el proyecto avanza, el equipo gane precisión en sus estimaciones.
Análisis de métrica: ver en detalle, sección 9. Métricas.
Evolución: buscamos una mejora continua en la exactitud de las estimaciones. Después de cada *Release*, tuvimos en cuenta dichas desviaciones para el momento de volver a estimar las históricas de usuario teniendo en cuenta las lecciones aprendidas con respecto a las estimaciones.
Resultado: cumplido.

2.b) Desviación en la estimación de esfuerzo

Meta: no ascendente.
Significado de meta: en esta métrica evaluamos la precisión que tuvo el equipo en la estimación del esfuerzo realizado. El objetivo de mantener una tendencia no ascendente en las desviaciones busca garantizar que, al igual que con la desviación del tamaño, a medida que el proyecto avanza, el equipo gane precisión en sus estimaciones.
Análisis de métrica: ver en detalle, sección 9. Métricas.

Evolución: buscamos una mejora continua en la exactitud de las estimaciones. Después de cada *Release*, tuvimos en cuenta dichas desviaciones, tanto la del tamaño como la del esfuerzo, para el momento de volver a estimar el esfuerzo teniendo en cuenta las lecciones aprendidas con respecto a las estimaciones. Es importante destacar que ambas métricas, la de la desviación de tamaño y esfuerzo, las comenzamos a medir a partir del *Release* 2 ya que, como mencionamos en el capítulo 5. Delivery, en el *Release* 1 no contábamos con suficiente experiencia para estimar y necesitamos generar un historial para basar nuestra estimaciones.
Resultado: cumplido.

2.c) Costo acumulado

Meta: menor o igual a 3172.
Significado de meta: esta métrica controla el costo total acumulado del proyecto, para este establecimos un límite superior de 3172 dólares, en donde establecimos que 1hr = 1 USD. La meta de no exceder este monto busca promover la eficiencia en la gestión de recursos.
Análisis de métrica: ver en detalle, sección 9. Métricas.
Evolución: podemos observar que al comienzo del proyecto, nuestros gastos se mantuvieron consistentemente por debajo del costo objetivo establecido. Ante esta situación, procedimos a realizar un ajuste basado en el promedio de gastos de los últimos seis meses, lo cual nos llevó a establecer un nuevo costo objetivo de 214 dólares por mes. Sin embargo, luego de observar un aumento significativo en los costos durante los dos meses siguientes, realizamos otro ajuste, esta vez tomando como referencia el promedio de los últimos seis meses, lo que resultó en un costo objetivo de 266 dólares por mes. Por otro lado, si se observa la gráfica de evolución de costos, podemos ver que luego de noviembre que nuestros gastos se alinearon bastante con el costo objetivo, con la excepción del último mes, febrero. Este desvío se debe a un cambio de las actividades realizadas; mientras que en los meses anteriores el foco estaba en el desarrollo, en febrero arrancamos a realizar la documentación para el proyecto final. Este cambio de actividad hizo que se genere un aumento en los costos, lo que resultó en que superáramos el límite del costo objetivo establecido por 185 horas. En el anexo podemos ver el A.36. Reporte de

clockify extraído que muestra mensualmente la cantidad de horas invertidas por todos los integrantes del equipo.
Resultado: no cumplido.

2.d) Grado de satisfacción (del 1 al 3) del equipo con el proceso de metodología ágil seleccionado

Meta: mayor a 2.
Significado de meta: evalúa la satisfacción del equipo con el proceso de metodología ágil implementado, en una escala de 1 a 3. Una puntuación superior a 2 indica un alto nivel de satisfacción y adaptación del equipo al marco ágil elegido.
Análisis de métrica: ver en detalle, sección 9. Métricas.
Evolución: para esto fuimos monitoreando la evolución de la satisfacción del equipo a lo largo del tiempo, poniendo en práctica ajustes basados en el feedback del equipo para asegurar que la metodología ágil continúe cumpliendo los objetivos del proyecto y las necesidades del equipo.
Resultado: cumplido.

En conclusión, pudimos establecer y seguir un proceso eficiente y adecuado para el proyecto, pudimos alcanzar resultados que se alinearon con nuestras metas. Además, pudimos mantener las desviaciones en las estimaciones de tamaño y esfuerzo controladas, demostrando nuestra habilidad para anticipar y ajustar el trabajo de forma precisa a lo largo del proyecto.

A pesar de no cumplir completamente con la meta del costo acumulado, consideramos que este aumento en el costo al final del proyecto no compromete significativamente nuestros logros. Si observamos la gráfica, estuvimos cumpliendo la meta durante la mayor parte del proyecto, manteniéndonos cercanos a la línea objetivo y realizando ajustes, como el tutor nos recomendaba, para establecer metas realistas. El problema fue el último mes donde registramos una desviación ligeramente mayor, esto se debe a las tareas de documentación para la entrega final, lo cual consideramos comprensible dada la naturaleza del trabajo en esa etapa. A pesar de este leve exceso en el costo, el proyecto no se alejó de nuestro objetivo. Por lo que, todo esto refleja el éxito en la creación de un marco de trabajo no sólo eficiente y exitoso, sino también bien valorado por el equipo.

10.3. Evaluación de Objetivos Personales

3. Aprender nuevas tecnologías

Key Results	Meta	Resultado	Diferencia
a. Aprender y aplicar correctamente nuevas metodologías claves de desarrollo durante la ejecución del proyecto.	>= 2	4	+2
b. Adquirir conocimiento y habilidades en las tecnologías clave utilizadas en el desarrollo de la aplicación, como ReactJs, NodeJs, Azure y AWS	>= 4	4	0

Tabla 10.3.1: Key Results objetivo personal N°3

3.a) Aprender y aplicar correctamente nuevas metodologías claves de desarrollo durante la ejecución del proyecto

Meta: mayor o igual a 2.
Significado de meta: esta métrica se centra en medir la capacidad del equipo de incorporar y utilizar eficazmente al menos dos nuevas metodologías de desarrollo clave a lo largo del proyecto. El objetivo es asegurar que el equipo no solo adquiera conocimiento teórico sino que también demuestre la aplicación práctica de estas metodologías para mejorar la eficiencia y efectividad del desarrollo.
Análisis de métrica: a lo largo del proyecto, pusimos en práctica diversas metodologías o por lo menos nos basamos en ellas para definir nuestra forma de trabajo. BDD, Scrum, *Design Thinking* son algunas de ellas. Esta estrategia nos permitió sumergirnos en las necesidades y expectativas de los usuarios finales, facilitando la ideación y definición de los requerimientos del sistema. Además, nuestra adaptación personalizada de Scrum nos llevó a realizar una muy buena gestión del proyecto y una optimización del flujo de trabajo.
Evolución: la aplicación de estas metodologías hizo que tengamos una mejora en la planificación, ejecución y entrega de nuestro proyecto. A través de iteraciones y las adaptaciones, el equipo no solo aprendió estas metodologías, sino que también las aplicó efectivamente, lo que resultó en un proceso de desarrollo más ágil y orientado al usuario.
Resultado: cumplido, pudimos cumplir de manera exitosa más de dos metodologías, esto lo podemos evidenciar en el capítulo de *5.9. Delivery* y *5.2. Discovery*.

3.b) Adquirir conocimiento y habilidades en las tecnologías clave utilizadas en el desarrollo de la aplicación, como ReactJs, NodeJs, Azure y AWS

Meta: mayor o igual a 4.
Significado de meta: esta métrica mide el nivel de conocimiento adquirido por el equipo en las tecnologías fundamentales para el desarrollo de la aplicación, incluyendo ReactJs, NodeJs, Azure y AWS. La meta establecida en "4 o más" involucra la capacidad de aplicar estos conocimientos de manera efectiva en el proyecto.
Análisis de métrica: aca examinamos el progreso del equipo en el aprendizaje y aplicación práctica de las tecnologías, evaluando su impacto en la calidad de la aplicación.
Evolución: para poder evaluar la evolución de la adquisición de conocimiento de tecnologías clave, primero, organizamos reuniones informales periódicas, donde evaluamos el progreso individual de cada integrante a través de preguntas técnicas, permitiéndonos medir de manera concreta el conocimiento adquirido por cada integrante. En segundo lugar, fomentamos sesiones de intercambio de conocimientos entre los miembros del equipo. Estas sesiones no solo ampliaron nuestra comprensión colectiva, sino que también facilitaron intercambiar conocimientos adquiridos en un entorno colaborativo. Además, al realizar code reviews, nos permitía evaluar la capacidad técnica para aplicar estas tecnologías nuevas en el proyecto. Este enfoque nos aseguró una evaluación continua y un desarrollo competente en las tecnologías seleccionadas. Dado que pudimos aplicar las tecnologías en el proyecto de forma exitosa, consideramos que pudimos adquirir conocimiento de nuevas tecnologías de forma exitosa.
Resultado: cumplido.

En conclusión, alcanzamos con éxito nuestro objetivo de aprender y aplicar nuevas tecnologías,desde dominar dos metodologías de desarrollo importantes hasta adquirir conocimiento en tecnologías críticas como ReactJs, NodeJs y AWS. A pesar de enfrentarnos inicialmente a un dominio y un conjunto de tecnologías parcialmente desconocidas, la investigación continúa y las capacitaciones permitieron al equipo mejorar sustancialmente su eficiencia y productividad.

Además, extendimos nuestro aprendizaje al ámbito de la comida y el marketing multinivel, un sector nuevo para todos nosotros.

10.4. Conclusiones generales

A nivel general, el proyecto se caracterizó por un fuerte enfoque en la calidad del producto y la satisfacción del cliente, guiado por la metodología de OKRs que garantizó objetivos claros y medibles, como explicamos anteriormente. A la fecha de finalización del proyecto TuBocado, consideramos que los 4 integrantes nos encontramos satisfechos con el trabajo que realizamos y los resultados que obtuvimos hasta ahora.

Entendemos que el desarrollo de este proyecto fue una experiencia invaluable de aprendizaje y crecimiento personal y académico, guiado por los desafíos que enfrentamos y los diferentes errores que fueron surgiendo. Aprendimos lo que significa el compromiso durante un extenso periodo de tiempo para realizar un proyecto de estas características. Con errores y aciertos, y muchas veces cierta incertidumbre, logramos establecer una comunicación con todos, ya sea el *product owner*, tutor, expertos, etc., que nos permitió solventar esos desafíos y continuar con el rumbo del proyecto.

11. Lecciones aprendidas

A lo largo del proyecto, nos hemos encontrado con diversos desafíos y obstáculos que no solo nos obligaron a aprender nuevas tecnologías, sino que también nos pusieron a prueba en todas las áreas del proyecto. Algunos de estos problemas ya los habíamos previsto como riesgos, mientras que otros surgieron durante el desarrollo del proyecto, desde la comunicación con el cliente hasta el proceso de desarrollo y las prácticas metodológicas que seguimos.

En este capítulo, recopilamos las lecciones aprendidas de estos desafíos, organizándolas según las áreas a las que pertenecen para que sean más fáciles de encontrar y entender. Además, establecimos un formato para estructurar estas lecciones de manera que puedan ser comprendidas fácilmente y utilizadas por otras personas a futuro, para que aprendan de nuestras experiencias.

Cada lección está acompañada de una anécdota que ilustra cómo enfrentamos este problema en particular, con el objetivo de facilitar su aplicación práctica.

11.1. Gestión

1. **Si** deseamos mejorar continuamente nuestro proceso de desarrollo, debemos realizar retrospectivas periódicas para identificar áreas de mejora y tomar acciones correctivas.

   Anécdota: Inicialmente no realizamos retrospectivas y nos olvidábamos las devoluciones entre nosotros, esto nos llevó a cometer errores varias veces. Aprendimos la importancia de la retroalimentación y de tomar acciones en base a estas para seguir mejorando y no repetir errores.
2. **Si** queremos fomentar la colaboración y el trabajo entre nosotros, es importante establecer roles y responsabilidades claras desde el principio del proyecto.

   Anécdota: En los primeros días del proyecto, no definimos adecuadamente los roles y las responsabilidades de cada uno, lo que generó confusiones y duplicaciones de esfuerzos. Aprendimos la importancia de establecer roles

claros para que exista un referente de un área y así asignar tareas de manera eficiente.

3. **Si** queremos garantizar un progreso constante en el desarrollo del proyecto, es crucial planificar adecuadamente las ausencias y vacaciones.

   Anécdota: Nos dimos cuenta de que cuando algunos miembros se fueron de licencia sin un plan de contingencia claro, el progreso fue afectado. No teníamos un plan para redistribuir las tareas o recursos adicionales para cubrir las ausencias, lo que resultó en retrasos en el proyecto. Aprendimos la importancia de planificar las licencias para poder evitar retrasos inesperados.

4. **Si** queremos realizar estimaciones más precisas en nuestros proyectos, es crucial implementar y seguir metodologías de estimación basadas en la experiencia y los datos anteriores, en lugar de confiar en la intuición.

   Anécdota: Durante las primeras etapas del proyecto, subestimamos la complejidad de ciertas funcionalidades, pensando que podríamos completarlas rápidamente. Esto nos llevó a planificar más tareas de las que podíamos manejar, resultando en estrés para nosotros y retrasos en el cronograma. Aprendimos a generar un historial de esfuerzo para poder luego estimar de manera correcta. En la sección de riesgos materializados se puede ver cómo esto se manifestó.

5. **Si** deseamos evitar problemas de planificación, entonces, es importante estimar el esfuerzo (tiempo por persona) para luego poder estimar la velocidad (story points por *Sprint*).

   Anécdota: Al inicio del proyecto intentamos arrancar asignando story points y luego midiendo el esfuerzo lo cual parecía redundante, pero gracias a la ayuda del tutor nos dimos cuenta que no teníamos la experiencia necesaria para hacerlo y la manera de realizar esta estimación era incorrecta. Gracias a esto, arrancamos registrando cuando nos llevó cada historia de usuario en horas para poder generar un historial y así al siguiente release estimar de manera adecuada, estimando primero las horas por persona y luego los story points.

6. **Si** deseamos evitar problemas de planificación, entonces, al principio, es importante no asumir que todas las historias se pueden completar en un

Sprint, ya que puede ser difícil estimar adecuadamente la duración de las *User Stories*.

Anécdota: En el segundo *Sprint*, nos dimos cuenta de que nos quedaron algunas historias sin terminar. Esto nos llevó a ajustar nuestras expectativas y tener que reajustar el release plan así como aprender la importancia de la expectativa realista. Presentamos más información sobre esta experiencia en la parte de 5. Selección de metodología.

7. **Si** deseamos una gestión efectiva del tablero de tareas, entonces es fundamental entender cómo organizar las historias y otros elementos, como los riesgos del *Sprint*, ya que el armado puede ser complejo y depende de varios factores.

   Anécdota: En el *Sprint* 1, nos encontramos con dificultades para determinar dónde colocar las historias que aún no tenían criterios de aceptación y cómo visualizar los riesgos del *Sprint*. Esto nos llevó a revisar y ajustar la estructura de nuestro tablero de para que fuera más efectivo.

8. **Si** deseamos garantizar la entrega exitosa de un producto, entonces es esencial no considerar una release como "terminada" hasta completar todas las actividades pendientes.

   Anécdota: En una ocasión, marcamos una release como "completada" debido a que habíamos alcanzado todos los hitos principales de desarrollo. Sin embargo, nos dimos cuenta de que aún no habíamos realizado algunas encuestas necesarias de retroalimentación de usuarios, no habíamos efectuado el despliegue de esa versión en el entorno de producción y no habíamos documentado el *Definition of Done* para esa entrega. Un release no debe considerarse terminado hasta que todas las actividades planificadas se hayan realizado con éxito, evitando así sorpresas en el proceso de entrega. Aprendimos que es necesario crear el *Definition of Done* para basarnos en este para decidir cuando una historia está completada.

9. **Si** queremos tener un manejo adecuado y ordenado de user stories, para poder lograr el alcance definido y ser eficientes al desarrollar el código, es crucial mantener controlado el máximo de horas que una historia de usuario nos puede llevar.

Anécdota: Durante las primeras etapas del proyecto, no dividimos correctamente ciertas historias, de modo que al finalizar el primer release nos quedaron historias de usuario de más de 20 horas de desarrollo, lo cual no es ideal y enlentece el desarrollo del proyecto.

11.2. Ingeniería

1. **Si** queremos comprender realmente las necesidades y deseos de los usuarios, es fundamental realizar investigaciones y entrevistas en profundidad para obtener una visión completa de sus experiencias y perspectivas a todos los tipos de usuarios.

   Anécdota: Al principio, creíamos conocer las necesidades del cliente, pero al realizar entrevistas más detalladas, descubrimos nuevas que ni el cliente conocía. Aprendimos a no dar por sentado nuestro conocimiento inicial y a dedicar tiempo suficiente a la investigación empática, es decir, descubrir insights.
2. **Si** queremos maximizar el valor de los pretotipos durante la etapa de prototipación en *Design Thinking*, debemos realizarlos lo antes posible y obtener comentarios tempranos de los usuarios y *product owner* para realizar mejoras continuas.

   Anécdota: En la primera iteración, dedicamos mucho tiempo a desarrollar una parte del prototipo detalladamente sin obtener comentarios de los usuarios y sin realizar pruebas rápidas lo que nos llevó a gastar mucho tiempo en realizar cosas que al final estaban mal. Aprendimos a realizar la validación temprana de prototipos para ahorrar tiempo y esfuerzo.
3. **Si** vamos a utilizar ejemplos de otras empresas o servicios como inspiración en *Design Thinking*, es fundamental adaptar esas ideas a nuestras propias necesidades en lugar de intentar utilizarlas.

   Anécdota: Durante nuestra investigación, nos enfocamos demasiado en cómo funcionaban algunas aplicaciones de comida rápida y en empresas que aplican el *marketing* multinivel que no enfatizamos en nuestras propias características y requisitos. Aprendimos que debemos ver y aprender sobre aplicaciones de empresas similares pero ajustarlas y personalizarlas para que sean adecuadas para nuestra aplicación.

4. **Si** deseamos validar y mejorar la usabilidad de un producto, entonces la retroalimentación temprana de los usuarios es esencial. No esperar hasta el *release* final, permite realizar ajustes en etapas tempranas según las necesidades reales de los usuarios.

   Anécdota: Al tener una retroalimentación temprana del proyecto con diversos usuarios, pudimos validar gran parte del proyecto en la cual teníamos dudas si estábamos bien encaminados.

5. **Si** deseamos evitar la dedicación excesiva de tiempo en la escritura de criterios de aceptación, entonces puede ser más efectivo validar las *user stories* con un prototipo o enfoque más ágil.

   Anécdota: En el *Sprint* 2, nos encontramos con una situación en la que estábamos tratando de definir criterios de aceptación extremadamente detallados para todas las user stories. Esto resultó en que dedicamos una cantidad significativa de tiempo en la planificación del *Sprint*, lo que finalmente generó presión para cumplir con el cronograma del *Sprint*. Aprendimos que somos más visuales y funciona mejor desarrollando al seguir un prototipo.

6. **Si** queremos mantener una comunicación efectiva entre nosotros y el *product owner*, debemos establecer canales de comunicación claros y periódicos, fomentando la retroalimentación constante.

   Anécdota: Al principio, no establecimos una comunicación fluida con el *product owner* y, como resultado, hubo malentendidos de cómo funcionaba la aplicación que recién saltó a la hora de validar la primera versión del pretotipo.

7. **Si** queremos utilizar alguna herramienta en particular, debemos averiguar sus planes gratuitos y pagos para ver qué funcionalidades tiene.

   Anécdota: Al principio, arrancamos utilizando Notion porque nos parecía una herramienta excelente, pero más adelante nos dimos cuenta que debíamos pagar por ella si queríamos seguir guardando documentos, lo que nos hizo realizar un cambio de herramienta de gestión de la documentación.

11.3. Tecnología

1. **Si** deseamos implementar tecnologías nuevas en nuestro proyecto, es esencial dedicar tiempo suficiente para capacitarnos y para hacer pruebas antes de incluirlos en el flujo de desarrollo principal.

   Anécdota: Iniciamos el desarrollo usando una pila tecnológica que algunos no conocíamos bien. Arrancamos una capacitación pero no realizamos suficiente investigación previa. Esto nos hizo perder tiempo en correcciones de errores que podríamos haber evitado si hubiéramos dedicado aún más tiempo al aprendizaje y la capacitación.
2. **Si** debemos elegir y comparar diversas tecnologías para utilizar en aplicaciones, utilizar Matrices de Pugh ayuda mucho a tomar decisiones.

   Anécdota: para poder tomar decisiones y tener en cuenta y comparar diferentes áreas de cada tecnología, la Matriz de Pugh nos ayudó mucho. En esta, listamos todas las áreas a considerar, la importancia y el peso que tenía cada una. Aprendimos a ver globalmente todos los aspectos antes de tomar decisiones.

12. Pasos a seguir

Al finalizar la entrega creemos que existen algunos pasos a seguir pendientes en el proyecto, que junto con el cliente, definimos. A continuación se detallan.

12.1. Validar modelo de negocio con Poc

El punto más importante a seguir es el de la validación del modelo de negocio del POC. Para asegurar la viabilidad y eficacia de nuestro modelo de negocio, tenemos que llevar a cabo una validación con el Proof of Concept (POC) el cual creamos. Esto implica utilizar las aplicaciones realizadas para probar sus características clave y evaluar su aceptación por parte de los usuarios potenciales. Es decir, utilizar la aplicación en un casos reales, en el cual un restaurante reciba pedidos los cuales serán creados y luego enviados al consumidor, además de probar todo el manejo y creación de Bocados y ver si es aceptado en el mercado.

12.2. Expandir base de clientes

Relacionado al paso anterior, expandir clientes es necesario para poder terminar de validar el modelo de negocio. Hoy en día el cliente cuenta con 3 dueños de restaurantes medianos o pequeños por lo que un paso a seguir es la búsqueda activa de nuevos restaurantes. Esto incluye campañas de publicidad, colaboraciones y la mejora continua de la aplicación basada en *feedback* para atraer y retener a un público más amplio.

Por otro lado, la búsqueda de consumidores de la aplicación no es necesaria ya que una de las ventajas del *marketing* multinivel es la viralidad. El *marketing* multinivel busca venderse por sí solo y no se necesita gastar recursos en *marketing* ni realizar una búsqueda activa de consumidores.

12.3. Sistema de facturación

Actualmente la facturación realizada por los pedidos generados en la aplicación van por parte del restaurante. Es por esto que un paso a seguir es la incorporación de un sistema de facturación interno en nuestra aplicación. Este sistema debería encargarse de la generación de facturas electrónicas, tanto para los

usuarios finales como para los restaurantes. Por otro lado, al momento de dar este paso también planeamos modificar la estrategia de facturación; ya que una vez establecida la presencia de la aplicación en el mercado, buscaremos revisar el modelo de cobro junto con la implementación de la facturación como pudimos ver que hicieron otras empresas de comida a lo largo de su crecimiento.

En la actualidad, el esquema de cobro que tiene la aplicación se basa en permitir que los restaurantes retengan los ingresos generados por ventas no asociadas a Bocados (es decir todas las ventas mediante tarjeta o efectivo), mientras que al final del período, el manager de TuBocado es el encargado de emitir una factura global a cada restaurante, basada en la cantidad de Bocados canjeados. Nuestro objetivo futuro es centralizar la recaudación, de modo que sea la aplicación la que acumule todos los ingresos, encargándose luego de pagar a los restaurantes mensualmente según sus ventas. Esta estrategia de cambio surge como un punto identificado durante nuestra etapa de empatía en el proceso de *Design Thinking.*

En algunas conversaciones con propietarios de restaurantes que llevan años en plataformas de venta de comida desde que iniciaron, ellos nos dijeron que inicialmente estas plataformas delegaban toda la gestión de facturación y cobranza a los restaurantes, los cuales debían pagar a la aplicación al final del mes. Sin embargo, debido a retrasos o incumplimientos en los pagos, hicimos necesario que las aplicaciones asumieran directamente la gestión de facturación y cobranza, efectuando los pagos a los restaurantes al cierre de cada mes. Creemos que este ajuste será necesario una vez que la aplicación arranque a agrandarse ya que no solo optimiza la eficiencia operativa sino que también mejora la relación con los restaurantes al facilitar un montón la gestión de la facturación final entre el restaurante y la aplicación.

12.4. Implementar sistema de delivery

Otro de los pasos a seguir es incorporar a la aplicación un sistema de *delivery* propio, el cual se encargue de manejar la logística de las entregas de los pedidos que actualmente, el sistema no cuenta con un seguimiento en tiempo real de los repartidores ni una gestión automática de los mismos.

Desarrollar un sistema de *delivery* propio implicará la creación de una nueva aplicación que permita la gestión de la logística de entrega. El mismo contaría con una aplicación web la cual comunicará a los repartidores con el restaurante y consumidores. Esto incluirá desde el seguimiento en tiempo real de los pedidos hasta la obtención del estado de un pedido en tiempo real. Implementar esta solución mejorará la experiencia del cliente al ofrecer mayor transparencia y eficiencia en el proceso de entrega.

12.5. Agregar promociones con Bocados

En una siguiente etapa, planeamos implementar la funcionalidad de poder crear promociones de Bocados en la aplicación. Lo importante en este paso es enfocarse en proporcionar a los restaurantes la capacidad de configurar y crear promociones personalizadas para fomentar los pedidos de comida en diferentes ocasiones. Esto permitirá a los restaurantes diseñar ofertas especiales, descuentos exclusivos o paquetes promocionales que se adapten a sus necesidades y estrategias comerciales.

12.6. Manejar más de un restaurante con una única cuenta

Otro paso planeado, involucra permitir que los propietarios gestionen múltiples restaurantes con una sola cuenta. Este paso permitirá mejorar la eficiencia operativa y la experiencia del usuario. La decisión de implementar esta funcionalidad responde a la necesidad de flexibilidad y escalabilidad en la gestión de restaurantes, permitiendo a los propietarios expandir sus operaciones sin complicaciones adicionales.

Es importante remarcar que esto se debe a una decisión de diseño premeditada por nosotros y pactada con el cliente, debido a que facilitó la creación de la POC y el cliente creía que era innecesario y agrega complejidad al usuario a la hora de probar la aplicación. Sin embargo durante el desarrollo siempre tuvimos en cuenta que íbamos a realizar este cambio, por lo que la arquitectura ya está preparada para poder recibir más de un restaurante por administrador y esto se debe a las decisiones de diseño que nos van a facilitar realizarlo con el menor impacto posible.

Referencias bibliográficas

[1] Apigee. (2021). *What is an API?* Disponible en: https://apigee.com/about/cp/what-api Accedido el: Mar. 10, 2024.

[2] Bass, L., Clements, P., & Kazman, R. (2012). *Software Architecture in Practice* (3rd ed.). Addison-Wesley.

[3] Stallings, W. (2018). *Computer Security: Principles and Practice* (4th ed.). Pearson.

[4] Fowler, M. (2021). *Patterns of Enterprise Application Architecture*. Addison-Wesley.

[5] Wynne, M., & Hellesoy, A. (2012). *The Cucumber Book: Behaviour-Driven Development for Testers and Developers*. Pragmatic Bookshelf.

[6] Gómez Baray, K. (2020, 2 de febrero). ¿Qué es un bug y cómo afecta a tu computadora? Disponible en: https://www.eleconomista.com.mx/tecnologia/que-es-un-bug-20200131-0079.html. Accedido el: Mar. 17, 2024.

[7] Admin_Donetonic. (2023, 6 julio). Los tableros en la gestión de proyectos agile. DoneTonic. Disponible en: https://donetonic.com/es/los-tableros-en-la-gestion-de-proyectos-agile/ Accedido el: Mar. 17, 2024.

[8] Economipedia. (s.f.). Marketing Multinivel. Economipedia. [En línea]. Disponible en: https://economipedia.com/definiciones/marketing-multinivel.html. Accedido en: 4 marzo de 2024.

[9] Martin, R. C. (2017). *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall.

[10] Clockify. (2020). *About Clockify*. Disponible en: https://clockify.me/about Accedido el: Mar. 10, 2024.

[11] Amway. (s.f.). Título de la página. Disponible en: https://www.amway.com.uy/content/page_Content.aspx?Dest=/Main/Oportunidad/Multinivel. Accedido el: Feb. 27, 2024.

[12] Beck, K., Beedle, M., Bennekum, A. V., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., ... & Thomas, D. (2001). Manifiesto para el desarrollo ágil de software. Recuperado de https://agilemanifesto.org/. Accedido el: Feb. 10, 2024.

[13] Atlassian. (s.f.). ¿Qué es Gitflow? Atlassian. [En línea]. Disponible en: https://www.atlassian.com/es/git/tutorials/comparing-workflows/gitflow-workflow#:~:text=%C2%BFQu%C3%A9%20es%20Gitflow%3F,vez%20y%20quien%20lo%20popularizo%C3%B3. Accedido en: 4 de marzo de 2024.
[14] Schwaber, K., & Sutherland, J. (2020). La Guía de Scrum: La Guía Definitiva de Scrum: Las Reglas del Juego.
[15] ¿Qué es el despliegue continuo? | IBM. (s. f.). Disponible en: https://www.ibm.com/es-es/topics/continuous-deployment Accedido el: Mar. 10, 2024.
[16] Laoyan, S. (2024, 11 febrero). Qué es Design Thinking y cómo aplicarlo [2024] • Asana. Asana. Disponible en: https://asana.com/es/resources/design-thinking-process Accedido el: Mar. 10, 2024.
[17] Duch, A. (2006, marzo). Esquema de Dividir y Vencer. Barcelona.
[18] Contenedores de Docker | ¿Qué es Docker? | AWS. (s. f.). Amazon Web Services, Inc. Disponible en: https://aws.amazon.com/es/docker/ Accedido el: Mar. 10, 2024.
[19] Coppola, M. (s.f.). Qué es Domain Driven Design (DDD), ventajas y proceso básico. HubSpot. Disponible en: https://blog.hubspot.es/website/que-es-ddd Accedido el: Mar. 10, 2024.
[20] Zendesk. (2024, 28 febrero). ¿Qué significa feedback? Tipos de feedback y cómo aplicarlos. Disponible en: https://www.zendesk.com.mx/blog/feedback-que-es/ Accedido el: Mar. 10, 2024.
[21] Urrutia, D. (2023, 17 octubre). Qué es Frontend - Definición, significado y ejemplos. Arimetrics. Disponible en: https://www.arimetrics.com/glosario-digital/frontend Accedido el: Mar. 10, 2024.
[22] German. (2022, 21 diciembre). ¿Qué es Figma y para qué sirve? CEI: Escuela de Diseño y Marketing. Disponible en: https://cei.es/que-es-figma/ Accedido el: Mar. 10, 2024.
[23] Alegsa, L. (2023, 10 junio). Definición de Fix (informática). Alegsa.com.ar. Disponible en: https://www.alegsa.com.ar/Dic/fix.php Accedido el: Mar. 10, 2024.
[24] Gaite, M. A. C. (2024b, enero 26). BDD y Gherkin para comunicar mundos diferentes. Paradigma Digital. Disponible en: https://www.paradigmadigital.com/dev/bdd-gherkin-comunicar-mundos-diferentes/ Accedido el: Mar. 10, 2024.

[25] GitHub.com help documentation. (s. f.). GitHub Docs. Disponible en: https://docs.github.com/ Accedido el: Mar. 10, 2024.
[26] Qué es Google Drive. (s. f.). Arimetrics. Disponible en: https://www.arimetrics.com/glosario-digital/google-drive Accedido el: Mar. 10, 2024.
[27] Atlassian. (s. f.). ¿En qué consiste la integración continua? | Atlassian. Disponible en: https://www.atlassian.com/es/continuous-delivery/continuous-integration Accedido el: Mar. 17, 2024.
[28] Fernández Pérez, M., Barbón Manzano, K., Sánchez Calvo, R., & Rodríguez García, P. (s.f.). JEST: JavaScript framework.
[29] Atlassian. (s.f.). Jira | Issue & Project Tracking Software. Disponible en: https://www.atlassian.com/software/jira Accedido el: Mar. 17, 2024.
[30] JWT.io. (s.f.). Introduction to JSON Web Tokens. jwt.io. Disponible en: https://jwt.io/introduction Accedido el: Mar. 17, 2024.
[31] Atlassian. (s. f.). What is kanban?. Atlassian. Disponible en: https://www.atlassian.com/agile/kanban Accedido el Mar. 17, 2024
[32] DinaHosting. (2023, 24 julio). Loader CSS: Qué son y por qué los necesitas. El Blog de Dinahosting. Disponible en: https://dinahosting.com/blog/loader-css-que-son-y-por-que-los-necesitas/ Accedido el Mar. 17, 2024.
[33] Alegsa, L. (2023, 17 de junio). Definición de Login (informática).
[34] tooltyp. (2020, 7 agosto). ¿Qué es el Look and Feel? - tooltyp. Tooltyp. Disponible en: https://www.tooltyp.com/definiciones-de-marketing/que-es-el-look-and-feel/que-es-el-look-and-feel Accedido el Mar. 17, 2024.
[35] Alonso, A. (2018, 20 junio). Convención de nombres: desde el CamelCase hasta el kebab-case. Medium. Disponible en: https://adrianalonsodev.medium.com/convenci%C3%B3n-de-nombres-desde-el-camelcase-hasta-el-kebab-case-787e56d6d023 Accedido el Mar. 17, 2024.
[36] López, J. M. (2016, 9 mayo). Descubre qué es el Network Marketing o modelo de negocio MLM. Semrush Blog. Disponible en: https://es.semrush.com/blog/que-es-network-marketing/ Accedido el Mar. 17, 2024.
[37] Metro Retro. (s.f.). Free Agile Retrospective Tool. Disponible en: https://metroretro.io/ Accedido el Mar. 17, 2024.

[38] Git - ramificar y fusionar. (s. f.). Disponible en: https://git-scm.com/book/es/v2/Ap%C3%A9ndice-C%3A-Comandos-de-Git-Ramificar-y-Fusionar Accedido el Mar. 17, 2024.
[39] Laoyan, S. (2024, 8 de febrero). Agile Manifesto: qué son las metodologías ágiles [2024] • Asana. Asana. Disponible en: https://asana.com/es/resources/agile-methodology Accedido el Mar. 17, 2024.
[40] Simões, C. (2020, 22 julio). MVP vs MMP. ¿Cuáles son las diferencias? Blog ITDO - Agencia de Desarrollo Web, APPs y Marketing En Barcelona. Disponible en: https://www.itdo.com/blog/mvp-vs-mmp-cuales-son-las-diferencias/ Accedido el Mar. 17, 2024.
[41] Terreros, D. (2023, 20 de enero). ¿Qué es un nicho de mercado? Tipos y ejemplos.
[42] Lucas, J. (2019, 4 de septiembre). Qué es NodeJS y para qué sirve. Disponible en: https://openwebinars.net/blog/que-es-nodejs/ Accedido el Mar. 17, 2024.
[43] Zendesk. (2023, 7 de octubre). ¿Qué es el NPS? Concepto, fórmula y ejemplos. Disponible en: https://www.zendesk.com.mx/blog/nps-que-es/ Accedido el Mar. 17, 2024.
[44] Molina, D. (2022, 10 de febrero). Qué son los OKRs. Disponible en: https://www.iebschool.com/blog/que-es-metodologia-okr-rrhh-2-0/ Accedido el Mar. 17, 2024.
[45] Laoyan, S. (2024, 23 de enero). Planning poker: la estrategia integral para la estimación ágil. Disponible en: https://asana.com/es/resources/planning-poker Accedido el Mar. 17, 2024.
[46] Asana, T. (2024, 18 febrero). Prueba de concepto (POC): qué es y cómo implementarla [2024] • Asana. Asana. Disponible en: https://asana.com/es/resources/proof-of-concept Accedido el Mar. 17, 2024.
[47] ¿Qué es PostgreSQL? | IBM. (s. f.). Disponible en: https://www.ibm.com/mx-es/topics/postgresql Accedido el Mar. 17, 2024.
[48] Deyimar A. (2023, 29 junio). Qué es React: definición, características y funcionamiento. Disponible en: https://www.hostinger.es/tutoriales/que-es-react Accedido el Mar. 17, 2024.
[49] López Mora, S. (2021, 8 de junio). ¿Qué son las Single-page application (SPA)? El desarrollo elegido por Gmail y LinkedIn. Disponible en:

https://digital55.com/blog/que-son-single-page-application-spa-desarrollo-elegido-por-gmail-linkedin/ Accedido el Mar. 17, 2024.
[50] De Agar Tirado, J. M. (2023, 21 julio). ¿Qué es un SPIKE en Scrum? Disponible en: https://mamaqueesscrum.com/2023/07/24/que-es-un-spike-en-scrum/ Accedido el Mar. 17, 2024.
[51] Colas de mensajes completamente administradas - Amazon Simple Queue Service – Amazon Web Services. (s. f.). Amazon Web Services, Inc. Disponible en: https://aws.amazon.com/es/sqs/ Accedido el Mar. 17, 2024.
[52] Radigan, D. D. (s. f.). ¿Qué son los puntos de historia en la metodología ágil y cómo se estiman? Atlassian. Disponible en: https://www.atlassian.com/es/agile/project-management/estimation Accedido el Mar. 17, 2024.
[53] Gómez, P. (2021, 10 diciembre). ¿Qué es UI? - DevCamp. DevCamp. Disponible en: https://devcamp.es/que-es-ui/ Accedido el Mar. 17, 2024.
[54] "User Stories," Mountain Goat Software. [En línea]. Disponible en: https://www.mountaingoatsoftware.com/agile/user-storiess Accedido el: Mar. 17, 2024.
[55] "¿Qué es la velocidad en Scrum?," NimbleWork, 2023. [En línea]. Disponible en: https://www.nimblework.com/es/agile/que-es-la-velocidad/ Accedido el: Mar. 17, 2024
[56] P. Clements, D. Garlan, R. Little, R. Nord, and J. Stafford, *Documenting software architectures: views and beyond*. Portland, OR, USA: IEE, 2003. [en línea]. Disponible en: http://dx.doi.org/10.1109/icse.2003.1201264.. Accedido el: Mar. 10, 2024. f
[57] Bass, L., Clements, P., & Kazman, R. (2012). Software Architecture in Practice (3rd ed.). Boston, MA: Addison-Wesley.
[58] Nielsen, J. (1994). Heuristic evaluation. En J. Nielsen y R. L. Mack (Eds.), Usability Inspection Methods (pp. 25-62). New York: John Wiley & Sons.
[59] "Ingeniería de Calidad. (2023, febrero). Philip B. Crosby: Bibliografía y Aportes. En línea: ttps://www.ingenieriadecalidad.com/2023/02/philip-b-crosby-bibliografia-y-aportes.html Accedido el: Mar. 17, 2024."

[60] "Brave Developer. (2023, mayo 30). La Calidad en el Software: Definiciones. En línea: https://bravedeveloper.com/2023/05/30/la-calidad-en-el-software-definiciones/ Accedido el: Mar. 17, 2024."
[61] Metro Retro. (s.f.). Drop, Add, Keep, Improve Retrospective Template. Disponible en: https://metroretro.io/templates/drop-add-keep-improve-retrospective. Accedido el: Mar. 17, 2024.
[62] Neuman, G. (s.f.). Lo que se mide, se puede controlar. Disponible en: https://gabrielneuman.com/lo-que-se-mide-se-puede-controlar/. Accedido el: Mar. 17, 2024.
[63] Atlassian. (s.f.). Tipos de pruebas de software. Disponible en: https://www.atlassian.com/es/continuous-delivery/software-testing/types-of-software-testing. Accedido el: Mar. 17, 2024.
[64] Dev Mastery. (2019, 22 de abril). Using Clean Architecture for Microservice APIs in Node.js with MongoDB and Express [Video]. YouTube. Disponible en: https://www.youtube.com/watch?v=CnailTcJV_U&t=1713s Accedido el: Mar. 17, 2024.
[65] Google. (s.f.). Google TypeScript Style Guide. Disponible en: https://google.github.io/styleguide/tsguide.html. Accedido el: Mar. 17, 2024.
[66] Jest. (s.f.). Jest - Delightful JavaScript Testing. Disponible en: https://jestjs.io/. Accedido el: Mar. 17, 2024.
[67] Airbnb. (s.f.). Estilo de JavaScript en Airbnb. Disponible en: https://github.com/airbnb/javascript. Accedido el: Mar. 1, 2024.
[68] Microsoft. (s.f.). Guías de API de Microsoft. Disponible en: https://github.com/microsoft/api-guidelines. Accedido el: Mar. 17, 2024.
[69] React. (s.f.). Preguntas frecuentes sobre la estructura de React. Disponible en: https://reactjs.org/docs/faq-structure.html. Accedido el: Mar. 17, 2024.
[70] TypeScript. (s.f.). TypeScript y React. Disponible en: https://www.typescriptlang.org/docs/handbook/react.html. Accedido el: Mar. 17, 2024.
[71] LoginRadius. (s.f.). Mejores prácticas de codificación en React. Disponible en: https://www.loginradius.com/blog/engineering/guest-post/react-best-coding-practices . Accedido el: Mar. 17, 2024.

[72] Tailwind CSS. (s.f.). Documentación de Tailwind CSS. Disponible en: https://tailwindcss.com/docs. Accedido el: Mar. 17, 2024.
[73] Atlassian. (s.f.). Tutorial de Atlassian sobre cómo hacer una solicitud de extracción (pull request) en Git. Disponible en: https://www.atlassian.com/es/git/tutorials/making-a-pull-reques Accedido el: Mar. 17, 2024.
[74] Singhal, K. (s.f.). How to Read Test Coverage Report Generated Using Jest. Medium. [En línea]. Disponible en: https://krishankantsinghal.medium.com/how-to-read-test-coverage-report-generated-using-jest-c2d1cb70da8b Accedido el: 4 de marzo de 2024.
[75] FreeCodeCamp. (n.d.). *Los Principios SOLID explicados en español*. [en línea]. Disponible en: https://www.freecodecamp.org/espanol/news/los-principios-solid-explicados-en-espanol/. Accedido el: Mar. 12, 2024.
[76] Martin, R. C. (2017). *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall.
[77] Fowler, M. (s.f.). *bliki: DomainDrivenDesign*. [en línea]. Disponible en: https://martinfowler.com/bliki/DomainDrivenDesign.html. Accedido el: Mar. 12, 2024.
[78] Dev Mastery. (2019, 10 abril.). *Using Clean Architecture for Microservice APIs in Node.js* [Video]. YouTube. https://www.youtube.com/watch?v=CnailTcJV_U&t=557s Accedido el: Mar. 12, 2024.
[79] Richards, M., & Ford, N. (2020). *Fundamentals of Software Architecture*. O'Reilly Media.

Anexo

A.1. Glosario

API (Application Programming Interface): Conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el *software* de las aplicaciones, permitiendo la comunicación entre dos aplicaciones de *software* de manera independiente. [1]

Atributo de Calidad: Características específicas que determinan la calidad de un sistema, como el rendimiento, la seguridad y la usabilidad. [2]

Autenticación: Proceso de verificar la identidad de un usuario o dispositivo, generalmente antes de otorgar acceso a sistemas o recursos. [3]

Backend: Parte del *software* que no interactúa directamente con el usuario, encargada de ejecutar la lógica de la aplicación y gestionar el almacenamiento de datos. [4]

Behaviour Driven Development (BDD): Metodología de desarrollo de *software* que implica una colaboración estrecha entre desarrolladores, QA y no técnicos o participantes en el negocio, centrada en obtener un entendimiento claro de los comportamientos deseados del *software*. [5]

Bug: Son errores o fallas de un programa o sistema que produce resultados inesperados, es decir, que trabaja de una forma para la que no estaba diseñado originalmente. [6]

Board: Tablero donde se reflejan las tareas que deben desplegar los equipos de trabajo para conseguir el desarrollo del producto en metodologías ágiles. [7]

Bocados: Unidades virtuales de valor que se otorgan al comprar comida con dinero en el sistema TuBocado. Estas unidades se acumulan en la cuenta del consumidor y pueden ser utilizadas para comprar comida en lugar de dinero.

Clean Architecture: Un enfoque de diseño de *software* que separa los elementos del código en capas de responsabilidad, promoviendo la independencia del sistema y la facilidad de mantenimiento. [9]

Clockify: Herramienta de seguimiento de tiempo que permite a los usuarios medir y registrar el tiempo dedicado a tareas y proyectos. [10]

Consumidor: En el contexto de *software*, se refiere a una entidad que utiliza o consume los servicios o datos proporcionados por un sistema o aplicación.

Cucumber: Herramienta que apoya el desarrollo dirigido por comportamiento (BDD), permitiendo la escritura de pruebas en un lenguaje accesible.

Criterio de aceptación: Descripción detallada de los requisitos y condiciones que un producto o función debe cumplir para ser aceptado por el usuario o cliente.

Daily Scrum: Es un evento de 15 minutos para los desarrolladores del equipo de Scrum. Para reducir la complejidad, se lleva a cabo a la misma hora y en el mismo lugar todos los días hábiles del *Sprint*. Si el *product owner* o *scrum master* están trabajando activamente en elementos del *Sprint Backlog*, participan como desarrolladores. [14]

Definition of Done: Descripción formal del estado del Incremento cuando cumple con las medidas de calidad requeridas para el producto. [14]

Despliegue Continuo: Estrategia de desarrollo de *software* en la que los cambios de código de una aplicación se publican automáticamente en el entorno de producción. [15]

Design Thinking: Metodología de diseño de resolución de problemas que te permite abordar problemas complejos mediante un marco centrado en el ser humano. [16]

Divide and Conquer: es una técnica para resolver problemas que consiste en dividir el problema original en subproblemas (de menor tamaño), resolver los subproblemas y finalmente combinar las soluciones de los subproblemas para dar una solución al problema original. [17]

Dueño de Restaurante: Rol perteneciente a TuBocado que corresponde al propietario o gerente responsable de la gestión y operación de un restaurante.

Docker: La ejecución de Docker en AWS les ofrece a desarrolladores y administradores una manera muy confiable y económica de crear, enviar y ejecutar aplicaciones distribuidas en cualquier escala. [18]

Domain Driven Design (DDD): Es una técnica de diseño de *software* que se centra en el análisis y diseño del dominio del problema. Se utiliza para crear *software* de alta calidad que se ajuste a los requisitos del negocio; permite un uso eficiente del código y que el *software* sea fácil de mantener y modificar. [19]

Feedback: El término feedback se utiliza para definir la retroalimentación que se realiza sobre un asunto o proceso. En los equipos de trabajo, su objetivo es ayudar en el análisis del desempeño de los colaboradores, alinear los objetivos, corregir los errores y fomentar el aprendizaje. [20]

Frontend: Es la parte de un sistema de información al que el usuario accede directamente e interactúa para recibir o utilizar las capacidades de back-end del sistema anfitrión. [21]

Figma: Figma es una herramienta de prototipado web y editor de gráficos vectorial, que a diferencia de las otras herramientas, se aloja en la web. [22]

Fix: solución o corrección para un problema en un *software* o sistema. Los desarrolladores de *software* a menudo lanzan fix en forma de actualizaciones para corregir errores, eliminar vulnerabilidades de seguridad y mejorar el rendimiento del *software*. [23]

Gherkin: Gherkin es un lenguaje natural específico del dominio (DSL - Domain Specific Language) que puede ser entendido tanto por perfiles de negocio como técnicos e incluso puede ser traducido en código de manera automática. [24]

Github: Plataforma de control de versiones distribuída, donde los usuarios pueden colaborar en proyectos, o compartir código. [25]

Github Actions: GitHub Actions es una plataforma de integración y despliegue continuos (IC/DC) que te permite automatizar tu mapa de compilación, pruebas y despliegue. Puedes crear flujos de trabajo y crear y probar cada solicitud de

cambios en tu repositorio o desplegar solicitudes de cambios fusionadas a producción. [25]

Google Drive: Es un servicio gratuito de Google de almacenamiento de archivos en la nube. [26]

Integración Continua: La integración continua (CI) es la práctica de automatizar la integración de los cambios de código de varios contribuidores en un único proyecto de *software*. [27]

Jest: Es una biblioteca de JavaScript para crear, ejecutar y estructurar pruebas [28]

Jira: Jira *Software* es una herramienta de gestión de proyectos ágil utilizada por equipos para planificar, rastrear, lanzar y dar soporte a *software*. Permite a los equipos trabajar de manera autónoma proporcionándoles el contexto necesario para avanzar. [29]

JWT: JSON Web Token (JWT) es un estándar abierto que define una forma compacta y autónoma de transmitir información de forma segura entre partes como un objeto JSON. Esta información se puede verificar y confiar porque está firmada digitalmente. [30]

Kanban: Kanban es un marco de trabajo utilizado en el desarrollo de *software* ágil, caracterizado por la comunicación en tiempo real de la capacidad y la transparencia total del trabajo. Los elementos de trabajo se visualizan en un tablero Kanban, permitiendo a los miembros del equipo ver el estado de cada tarea en cualquier momento. [31]

Loader: Un loader CSS es una animación que se muestra mientras se está esperando a que termine algún proceso de la página web. [32]

Login: momento de autentificación al ingresar a un servicio o sistema con el uso de un conjunto de credenciales provistas y validadas por el sistema. [33]

Look and feel: Look and Feel, referido a un sitio web o app, que indica tanto el estilo visual (look) como el modo o sensación en que el Usuario la percibe (feel). [34]

LowerCamelCase: cuando la primera letra de cada una de las palabras es mayúscula, con la excepción de que la primera letra es minúscula. [35]

Manager de TuBocado: Administrador o dueño de la aplicación TuBocado, responsable de gestionar el sistema, incluyendo la administración de Bocados, cotizaciones, y otros aspectos operativos del sistema.

Marketing Multinivel: Es un modelo de negocio que consiste en una empresa que comercializa sus productos a través de una red de distribuidores. [36]

Metro Retro: Herramienta en línea que facilita la realización de retrospectivas ágiles y otras reuniones interactivas mediante un tablero blanco interactivo. [37]

Merge: Se utiliza para fusionar uno o más ramas dentro de la rama que tienes activa. A continuación avanzará la rama actual al resultado de la fusión. [38]

Metodologías ágiles: El esquema ágil es una metodología iterativa, es decir, se realizan entregas cíclicas y en cada entrega se realizan todas las etapas del ciclo: desde toma de requerimientos, diseño, verificación y entrega. Se entrega valor constantemente y se recibe feedback también durante todo el proyecto. [39]

MVP (Minimum Viable Product): Es un producto con las características suficientes para satisfacer a los clientes iniciales. El objetivo del MVP es “obtener el mayor conocimiento de los clientes con el menor de los esfuerzos”. [40]

MMP (Minimum Marketable Product): Aborda todas las necesidades del cliente, siendo la mejor versión del MVP, después de las validaciones, pruebas y errores, y está lista para comercializarse y vender. [40]

Nicho: Un nicho de mercado es una parte muy específica de un grupo de consumidores. Está compuesta por un número reducido de personas o empresas que comparten características y necesidades similares. [41]

NodeJS: Node.js es un entorno de tiempo de ejecución de JavaScript, orientado al desarrollo *backend* en sistemas web. [42]

NPS: es una métrica que se utiliza para medir el nivel de satisfacción del cliente con los productos o servicios adquiridos a una empresa; así como la probabilidad de que promueva la marca a otras personas. [43]

OKR: Proviene de las siglas de Objectives and Key Results (objetivos y resultados clave), y es una metodología de gestión de trabajo que tiene como finalidad facilitar los objetivos marcados por una empresa mediante la medición de manera cuantitativa y cualitativa del progreso de equipos. [44]

One Piece Flow: Flujo de trabajo donde se busca trabajar sobre un único producto o tarea, en lugar de avanzar en varias en paralelo.

Planning Poker: Es un método de estimación que ayuda a tu equipo ágil a calcular la cantidad de esfuerzo que se necesita para completar una historia de usuario en un backlog del producto. [45]

Proof of concept (POC): Es un estudio de viabilidad que realizas antes de involucrarte en un determinado proyecto o idea. Una POC puede ayudarte a demostrar a los clientes o equipos de productos por qué una idea tiene sentido en la práctica. [46]

Postgres: Es una base de datos de código abierto que tiene una sólida reputación por su fiabilidad, flexibilidad y soporte de estándares técnicos abiertos. [47]

Product Backlog: El Product Backlog es una lista emergente y ordenada de lo que se necesita para mejorar el producto. Es la única fuente del trabajo realizado por el equipo de Scrum. [14]

Product Owner: El Product Owner es responsable de maximizar el valor del producto resultante del trabajo del equipo de Scrum. La forma en que esto se hace puede variar ampliamente entre organizaciones, equipo de Scrum e individuos. [14]

ReactJS: Es una de las librerías más populares de JavaScript para el desarrollo de aplicaciones móviles y web. Contiene una colección de fragmentos de código JavaScript reutilizables utilizados para crear interfaces de usuario (UI) llamadas componentes. [48]

Release: Versión de un producto de *software* que cumple con la *Definition of Done* de este.

Referido: Es el usuario que se registra en el sistema utilizando el enlace compartido por otro usuario, formando parte de una red que contribuye a las ganancias en el sistema de *marketing* multinivel.

Referente: Es el usuario que comparte su enlace de referencia con otros, atrayéndolos al sistema de *marketing* multinivel, generando una red de referidos que le genera ganancias.

SCRUM: Scrum es un marco de trabajo liviano que ayuda a las personas, equipos y organizaciones a generar valor a través de soluciones adaptativas para problemas complejos. [14]

Single Page Application: Es un tipo de aplicación web que ejecuta todo su contenido en una sola página. [49]

Spike: Un Spike es un experimento que realiza el equipo de Scrum para profundizar en un determinado tema que les genera incertidumbre. [50]

Sprint: Son eventos de duración fija de un mes o menos para crear consistencia. Un nuevo *Sprint* comienza inmediatamente después de la conclusión del *Sprint* anterior. Todo el trabajo necesario para lograr el Objetivo del Producto, incluido la *Sprint Planning*, *Daily Scrums, Sprint Review y Sprint Retrospective*, ocurre dentro de los *Sprints*. [14]

Sprint Backlog: Se compone del Objetivo del *Sprint* (por qué), el conjunto de elementos del *Product Backlog* seleccionados para el *Sprint* (qué), así como un plan de acción para entregar el Incremento (cómo). [14]

Sprint Planning: Inicia el *Sprint* al establecer el trabajo que se realizará para el *Sprint*. El equipo de Scrum crea este plan resultante mediante trabajo colaborativo. [14]

Sprint Review: Ceremonia de la metodoloǵia Scrum donde el equipo de desarrollo lleva a cabo una reunion con el propósito de inspeccionar el resultado del *Sprint* y determinar futuras adaptaciones [14]

Sprint Retrospective: Ceremonia de la metodoloǵia Scrum donde el equipo de desarrollo lleva a cabo una reunion con el propósito de planificar formas de aumentar la calidad y la efectividad. [14]

SQS: Colas de mensajes completamente administradas para microservicios, sistemas distribuidos y aplicaciones sin servidor. Puede enviar, almacenar y recibir mensajes entre componentes de *software* de cualquier volumen sin perder mensajes ni requerir la disponibilidad de otros servicios. [51]

Story Point: Se trata de unidades de medida que permiten expresar una estimación del esfuerzo total que deberá hacer el equipo para implementar íntegramente un elemento del backlog del producto o cualquier otro trabajo. [52]

UI (User Interface): diseño con el que una persona interactúa dentro de las plataformas digitales. [53]

***UpperCamelCase*:** Es cuando la primera letra de cada palabra se escribe en altas o mayúsculas. [35]

User Story: Es una descripción corta y simple de una funcionalidad desde la perspectiva del persona que desea la nueva función. [54]

Velocidad: Cantidad media de trabajo que refiere a la cantidad que un equipo puede completar en una iteración [55]

***Views and Beyond*:** Mapea los conceptos y modelos de enfoque arquitectónicos conocidos, como el enfoque 4+1 de Rational Unified Process, el enfoque Siemens Four Views y las prácticas recomendadas por ANSI/IEEE-1471-2000 para documentar arquitecturas de sistemas. [56]

A.2. Vista Descomposición TuBocado.TuBocadoBE

En esta sección decidimos mostrar la descomposición de módulos de *TuBocadoBE* en representación a ambos servicios. Esto es ya que, el servicio de puntos se estructura igual que cada módulo del servicio principal.

Representación primaria

Para mejorar la claridad de los diagramas, elaboramos el diagrama de descomposición de los módulos, omitiendo las carpetas internas de cada módulo (administradores, órdenes, usuarios y restaurantes), debido a que la estructura interna de estos es idéntica. Más adelante, detallamos los componentes que se encuentran dentro de cada uno de estos módulos, proporcionando así una vista simplificada que facilita la comprensión del lector.

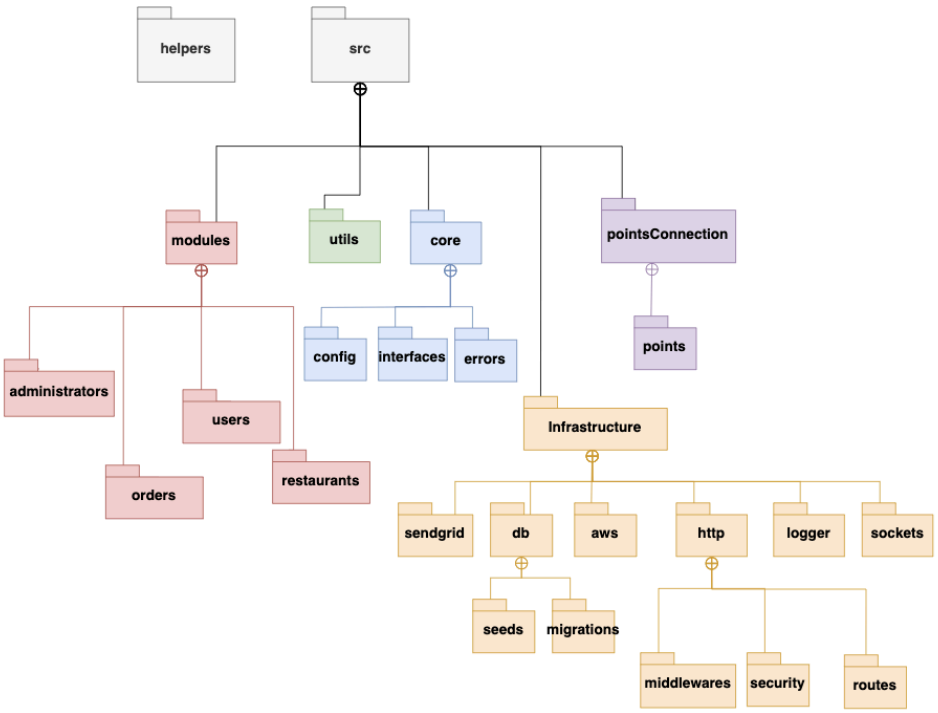


Diagrama de descomposición

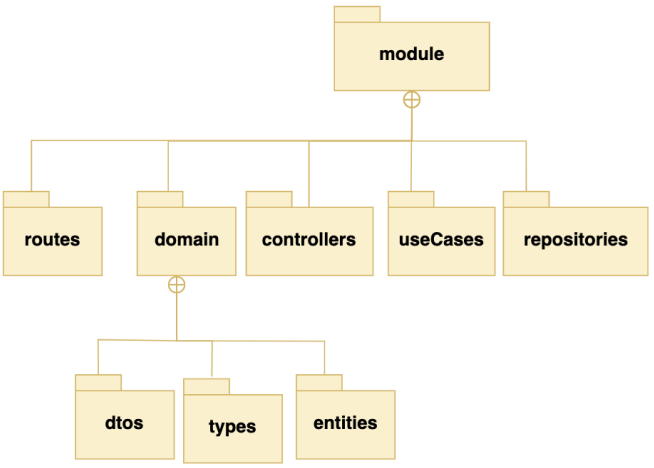


Diagrama de descomposición

Catálogo de elementos vista descomposición TuBocado.TuBocadoBE

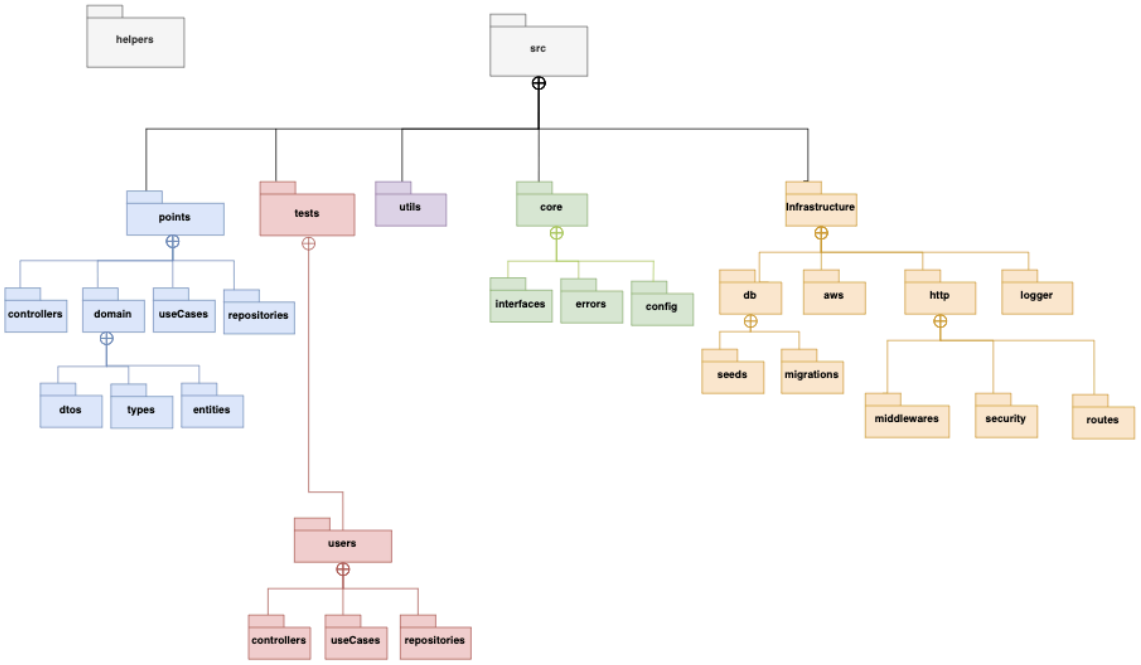
Catálogo de elementos

Elemento	Descripción
src	Directorio raíz que contiene el código fuente del proyecto.
core	Núcleo del sistema, incluye configuraciones globales y lógica esencial compartida, además de las interfaces para cada capa de los módulos.
infrastructure	Capa que maneja la comunicación con recursos externos como bases de datos, sistemas de archivos y APIs externas.
utils	Conjunto de herramientas y funciones de utilidad reutilizables en todo el proyecto.
tests	En este módulo se concentran todas las pruebas unitarias realizadas organizadas por módulo.
modules	Componentes modulares del sistema, cada uno encapsulando una funcionalidad específica. Está el módulo de restaurantes, de administradores, órdenes y de usuarios.
controllers	Controladores que manejan la lógica de entrada y salida de las solicitudes HTTP.
routes	Define todas las rutas del módulo, las mismas direccionan las peticiones a los controllers correspondientes.
domian	Define el modelo de dominio del negocio, incluyendo entidades y lógica de negocio.
useCases	Casos de uso que encapsulan la lógica de negocio específica, sirviendo como puente entre la interfaz de usuario y las operaciones del dominio.

repositories	Abstracciones que manejan la persistencia de datos, permitiendo al dominio interactuar con la base de datos.

Tabla comparativa motor de base de datos

Representación primaria vista descomposición TuBocado.PointsBE



Vista de descomposició de TuBocado.Points.BE

A.3. Vista Layers *TuBocado.PointsBE*

Representación primaria

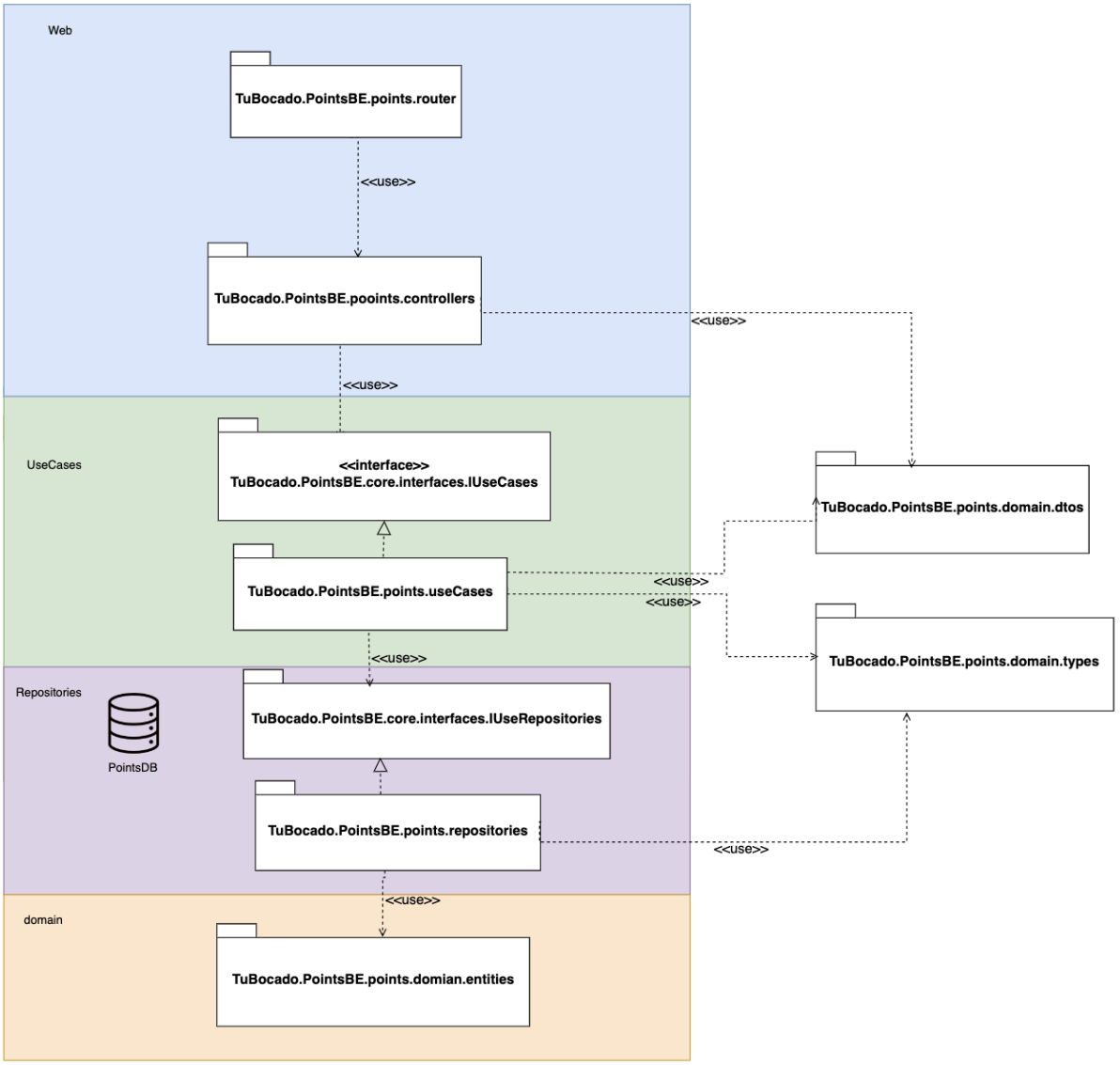


Diagrama de vista layers

Catálogo de elementos

Elemento	Descripción
TuBocado.PointsBE.point.routes.	Define todas las rutas del módulo, las mismas direccionan las peticiones a los *controllers* correspondientes.
TuBocado.PointsBE.points.controllers	Se encargan de recibir las peticiones, utilizados desde el router, para que las procese y responda adecuadamente.
TuBocado.PointsBE.core.interfac	Contiene las interfaces definidas para todos los *useCases*.

es.IUseCases	Estos son utilizados desde los controladores.
TuBocado.PointsBE.points.useCases	Implementa las interfaces definidas para los *useCases*. Aquí podemos encontrar la lógica de negocio del sistema que maneja todo lo relacionado al *marketing* multinivel, desde gestion de puntos a referidos, parámetros de cotización puntos y niveles.
TuBocado.PointsBE.core.interfaces.IRepositories	Contiene las interfaces definidas para todos los *repositories*. Estos son utilizados desde los controladores.
TuBocado.PointsBE.points.repositories	Implementa las interfaces definidas para cada uno de los repositorios del módulo. Son los encargados de la comunicación con los ORM.
TuBocado.PointsBE.points.domain.DTOs	En los DTO (*Data Transfer Object*) se encuentran objetos que se utilizan para transferir datos entre los servicios.
TuBocado.PointsBE.points.doamin.entities	Estos son los tipos o interfaces que definen la estructura de los modelos de la base de datos.
TuBocado.PointsBE.points.doamintypes	Aquí definimos tipos comunes que se utilizan, como tipos de respuesta de API, etc

Catálogo de elementos de vista layers

Decisiones de diseño

Al igual que los módulos de *TuBocado.TuBocadoBE* la arquitectura del servicio de puntos está basada en una adaptación del patrón layers con una combinación de *Clean Architecture* en la cual las dependencias van de afuera hacia dentro en capas concéntricas, siendo las entidades el núcleo de toda la estructura. En este caso está todo formado alrededor de la entidad de puntos. Decidimos utilizar este tipo de arquitectura para el sistema de puntos también justamente porque creemos que, al igual que se indicó arriba, reduce el acoplamiento y aumenta la cohesión. Esto último mencionado es de suma importancia para poder cumplir con la petición del cliente de tener este servicio desacoplado e independiente y que pueda funcionar con otros sistemas de venta y comercialización diferentes a *TuBocado.TuBocadoBE*. Es un tipo de arquitectura óptima para que el servicio para que luego pueda ser consultado por otros servicios de manera independiente.

A.4. Selección de tecnologías

Tipos y motor de bases de datos

Atributos	Base de datos Relacional	Base de datos No Relacional
Tipo	Bases de datos basadas en tablas con una estructura fija.	Bases de datos basadas en documentos, pares clave-valor, etc.
Esquema	Esquema predefinido y estricto.	Esquema dinámico y flexible.
Consultas	Lenguaje de consulta estructurado (SQL).	Consultas variadas basadas en el tipo de base de datos (NoSQL).
Transacciones	Soporte completo de transacciones ACID.	Varía, algunas soportan transacciones ACID, otras eventual consistencia.
Cuando usarlo	Cuando los datos deben ser consistentes y son datos complejos y estructurados.	Cuando los datos no son estructurados o cambian rápidamente.

Tabla comparativa proveedor de infraestructura base de datos

Atributos	SQL Server	PostgreSql
Licenciamiento	Producto comercial de Microsoft	Open source, gratuito.
Rendimiento	Alto rendimiento en transacciones complejas.	Muy eficiente.
Comunidad	Comunidad sólida, enfocada en tecnologías Microsoft.	Comunidad muy activa y diversa.
Seguridad	Seguridad avanzada y cumplimiento.	Excelente seguridad y cumplimiento de estándares.
Extensibilidad	Bueno pero más cerrado.	Muy extensibles
Sistemas operativos compatibles	PostgreSQL parece ser más universal. Está ampliamente disponible en múltiples sistemas operativos.	SQL Server está limitado a Windows y a Linux.
Concurrencia	Eficiente manejo de concurrencia con bloqueos y aislamiento de transacciones.	Avanzado control de concurrencia, con soporte para MVCC (Multi-Version Concurrency Control).
Compatibilidad con AWS/Azure	Alta compatibilidad con Azure, compatible con AWS.	Compatible con ambos, AWS y Azure.

Tabla comparativa motor de base de datos

Al evaluar las opciones de bases de datos para nuestro sistema, decidimos optar por las bases de datos relacionales sobre las no relacionales. Esta decisión, como se refleja en nuestra tabla comparativa, se basa en la necesidad de una estructura de datos definida y un esquema estricto que las bases de datos relacionales ofrecen. Esta estructura es esencial para poder garantizar la integridad y consistencia de los datos críticos en nuestro sistema, como los detalles de las transacciones de pago y el seguimiento de las órdenes. Por lo que, en base a nuestra investigación, podemos afirmar que las bases de datos *SQL* respetan un modelo ACID, que aseguran la consistencia de datos en todo momento mientras que las que son *NoSQL*, realizan copias de datos y no siempre tienen los últimos datos actualizados, sino que pueden presentar datos antiguos lo que hace que no sean consistentes. Además el cliente planteó su preferencia por utilizar bases de datos relacionales ya que sus técnicos ya tienen experiencia con la misma y se sienten cómodos con esa elección.

Por otro lado, a la hora de comparar el tipo de motor de datos, después de una comparación entre *PostgreSQL* y *SQL Server*, optamos por *PostgreSQL* por varias razones que se alinean con los requisitos y limitaciones. Aunque ambos son motores de bases de datos ampliamente utilizados, *PostgreSQ*L presenta ciertas ventajas. En primer lugar, una de las cosas que tuvimos en cuenta es el costo. *PostgreSQL* es un sistema de gestión de base de datos open source, no implica los costos de licencia asociados con *SQL Server*. Esto es especialmente uno de los puntos más importantes porque particularmente la etapa actual en la que se encuentra nuestro proyecto, donde la generación de ingresos aún no es grande, se busca minimizar los costos operativos.

Por último, otro de los puntos importantes es que *PostgreSQL* ofrece una buena compatibilidad con diversas plataformas y servicios en la nube, incluido AWS, que es nuestro proveedor de infraestructura elegido y Azure, que si bien ahora no lo estamos utilizando, fue una de las elecciones iniciales.

Tipos de colas de mensajes

Las colas de mensajes se utilizaron para comunicar el servicio de *TuBocado.TuBocadoBE* con el de *TuBocado.PointsBE* para poder mandar las

ordenes generadas para que el servicio de puntos se encargue de calcular los puntos generados y cargarlos a los usuarios correspondientes. Esto lo hicimos ya que consideramos que no es necesario una respuesta en tiempo real y que puede ejecutarse asincrónicamente sin afectar el flujo de la creación de órdenes.

Para elegir el tipo de cola de mensaje nos basamos inicialmente en el proveedor de infraestructura, dado que, al ser compatibles, la integración entre los mismos iba a ser mucho más simple. No solo eso, sino que generamos un ecosistema y coherencia entre los servicios utilizados. Si bien realizamos una investigación primaria de los tipos de cola de mensaje tanto en AWS (SQS) como en Azure, el cliente decidió cambiar de proveedor de infraestructura antes de que implementáramos las colas de mensaje en Azure. Por esta razón, elegimos colas de mensaje *SQS (Simple Queue Service)* de AWS. Además, como se puede ver en la tabla comparativa, AWS proporciona muchas ventajas, entre ellas es fácil de configurar, tiene una alta disponibilidad “Amazon SQS utiliza una infraestructura redundante para proporcionar acceso con un alto grado de simultaneidad a los mensajes y alta disponibilidad para producir y consumir mensajes.” además de la fiabilidad, escalabilidad y posibilidad de personalización que posee.

Alguna de las desventajas que posee AWS es que la cola no está ordenada por lo que la primera en entrar no es la primera en decir, es decir que no es FIFO, pero para nuestro caso no es relevante el orden del ingreso a la cola por lo que no afectó en la decisión.

Atributos	**Amazon SQS**	**Azure Queue Storage**
Precios	Pago por uso, basado en el número de solicitudes.	Pago por uso, basado en transacciones y almacenamiento.
Seguridad	Ofrece cifrado en tránsito y en reposo, control de acceso vía AWS IAM.	Ofrece cifrado en tránsito y en reposo, control de acceso vía Azure Active Directory y políticas de acceso.
Integración con otros servicios	Muy buena integración con otros servicios de AWS.	Muy buena integración con otros servicios de Azure.
Escalabilidad	Escalabilidad automática.	Escalable, gestionada por Azure, pero puede requerir ajustes basados en el volumen de almacenamiento.

Facilidad de uso	Interfaz de usuario intuitiva y API simple para el desarrollo.	Interfaz de usuario relativamente intuitiva comparado con otros servicios de Azure y acceso mediante Azure SDK.
Periodo de retención de mensajes	De 1 minuto a 14 días.	Hasta 7 días.

Tabla comparativa colas de mensajes

Envío de emails

Para que los managers de TuBocado se puedan comunicar con los dueños de los restaurantes existe la funcionalidad de envío de emails. Aca los managers pueden decir porque aceptan o no aceptan cada restaurante y que requisitos les falta para ser aceptados, como se detalla más arriba. Para esto comparamos dos servicios utilizados en el pasado por los integrantes del equipo; estos son Mailtrap y SendGrid de Twilio. Ambos son servicios que están destinados al envío de correo, SendGrid es un servicio en la nube muy popular y utilizado por empresas conocidas como Uber mientras que Mailtrap está más enfocado al ambiente de desarrollo pero dado que nuestro proyecto es una POC y la idea es validar la viabilidad de la aplicación, lo consideramos como opción.

Al analizar la tabla, observamos que utilizar Mailtrap no ofrecía mayor ventaja sobre Sendgrid ya que no disminuye significativamente los costos y tampoco ofrece una integración más simple, por lo que optamos por utilizar SendGrid. La integración con SendGrid no parecía un desafío ya que algunos de los integrales del equipo ya habían realizado integraciones con el mismo, sumado que el cliente se inclinó más por la opción de utilizar un proveedor que se pueda utilizar, no solo en un ambiente de testing, sino también en producción; mientras no afecte significativamente los costos

Atributos	**Mailtrap**	**SendGrid**
Principal uso	Pruebas de correo electrónico en desarrollo.	Envío de correos electrónicos en producción.
Precios	Planes gratuitos y pagos basados en la cantidad de correos electrónicos y la capacidad de la bandeja de entrada.	Planes gratuitos y pagos basados en el volumen de correos enviados por mes.
Integración	Fácil integración con sistemas de	API para integración con

	desarrollo.	aplicaciones web y *mobile*, relativamente fácil de integrar.
Soporte	Soporte básico en planes gratuitos, soporte prioritario en planes pagos.	Documentación extensa para planes gratuitos, soporte 24/7 en planes pagos.
Seguridad	Seguridad estándar para la protección de datos de correo electrónico de prueba.	Seguridad avanzada.

Tabla comparativa envío de emails

Manejo de imágenes

En nuestro sistema, la funcionalidad de guardado de las imágenes se realiza tanto para la foto de cada uno de los restaurantes como para los platos de los mismos. Para elegir el tipo de manejo de imágenes, al igual que para la cola de mensajes, nos basamos principalmente en el proveedor de infraestructura ya que al ser compatibles, la integración entre los mismos es mucho más simple. Esto hace que se genere un ecosistema entre los servicios utilizados y coherencia entre los mismos.

Al igual que con la cola de mensajes, realizamos una investigación primaria del servicio brindado por AWS el cual es AWS S3 y el brindado por Azure qué es Azure Blob Storage, a diferencia de la cola de mensajes, el cliente decidió cambiar de proveedor de infraestructura luego de investigar más en profundidad la implementación de Azure Blob Storage pero no llegamos a implementarla en el código ya que la primera versión del MVP contenía platos y restaurantes generados por un *seed* y solo realizamos la funcionalidad de la gestión de órdenes. Por esta razón, elegimos AWS S3 de AWS. Además, como se puede ver en la tabla comparativa, AWS S3 ofrece una interfaz de usuario muy intuitiva completa y un soporte con una extensa comunidad. No solo esto sino que es muy utilizada a nivel mundial y ofrece herramientas avanzadas para la gestión de los datos incluyendo mucha seguridad y control de acceso detallado según las configuraciones que se realicen.

Atributos	**AWS S3**	**Azure Blob Storage**
Precios	Pago por uso basado en almacenamiento, solicitudes y transferencia de datos.	Pago por uso basado en almacenamiento, transacciones y transferencia de datos.

Seguridad	Amplias características de seguridad, incluyendo control de acceso detallado, cifrado en tránsito y en reposo, y compatibilidad con políticas de IAM.	Funcionalidades de seguridad con cifrado en tránsito y en reposo, y soporte para Azure Active Directory.
Escalabilidad	Escalabilidad sin límites.	Alta escalabilidad.
Gestión de datos	Ofrece herramientas avanzadas para la gestión del ciclo de vida de los datos, incluyendo políticas automáticas de transición y expiración.	Proporciona capacidades similares para la gestión automática del ciclo de vida, facilitando la transición entre diferentes niveles de acceso.
Interfaz de usuario	Interfaz de usuario intuitiva.	Interfaz de usuario parcialmente intuitiva en comparación con el ecosistema de Azure.
Integración con otros servicios	Integración profunda con otros servicios de AWS.	Excelente integración con otros servicios de Azure.
Soporte	Disponible globalmente con una extensa red de regiones y puntos de presencia.	Amplia disponibilidad global con numerosas regiones para cumplir con requisitos de residencia de datos.

Tabla comparativa envío de manejo de imágenes

Pasarela de pagos

Para la elección de la pasarela de pagos inicialmente realizamos una investigación de cuáles son las más utilizadas en eCommerce en Uruguay. En base a esta investigación le planteamos esas opciones al cliente, pero el cliente se decidió por Mercado Pago, ya que era una pasarela de pagos conocida por él y creía fiable. Si bien le planteamos que Mercado Pago puede llevarse una comisión un poco más alta que otras, el cliente estaba convencido de su fiabilidad y de lo sencillo que resultaría integrarlo a nuestra aplicación.

Una vez decidido esto, realizamos *spikes* de la utilización de MercadoPago para poder validar la utilización del mismo, en el capítulo de metodología detallaremos más sobre esto.

Además, dado que la aplicación inicialmente estará disponible en Uruguay, MercadoPago aplica perfectamente. Es una plataforma conocida en Uruguay, líder en la región, que hace que en términos de visibilidad y confiabilidad sume mucho.

Esto nos dio la confianza tanto a nosotros como al cliente de que los usuarios no tendrán problemas al usarla para sus pagos.

Por lo tanto, siguiendo el consejo del cliente y considerando que algunos de los integrantes del equipo ya habían hecho integraciones con Mercado pago y dado que es más sencilla la integración que otras pasarelas (ya que realizáramos un spike y pudimos verificarlo), y como se mencionó, ofrece una amplia aceptación y confianza entre los consumidores uruguayos, decidimos que esta pasarela era la opción más lógica para nosotros.

Tailwind

Para el manejo de estilos de la aplicación seleccionamos como framework de CSS Tailwind, esto fue propuesto al cliente luego de los primeros diseños del prototipo FIGMA, lo que hizo que genere un retrabajo en alguno de los diseños y pantallas elaboradas.

Nos decidimos por *Tailwind* ya que actualmente el desarrollo web presenta tantas opciones avanzadas para la implementación de estilos que recurrir a CSS tradicional no parecía valer la pena. Con esto, evaluamos otros frameworks como puede ser *Bootstrap* pero a pesar de que *Bootstrap* trae consigo una biblioteca de componentes predefinidos que dan coherencia y agilizan el desarrollo, su rigidez y la tendencia a homogeneizar el aspecto de las aplicaciones web nos llevaron a buscar alternativas que tengan más flexibilidad y personalización, para adecuarse a los diseños realizados.

Entre las opciones evaluadas, nos debatimos entre adoptar una librería como *styled-components*, que permite escribir código CSS dentro de una función Javascript en lugar de escribir CSS en un archivo separado, y optar por un framework de CSS como Tailwind. Ambas opciones prometían una libertad creativa considerable en la construcción de estilos. Pero que, a pesar de la curva de aprendizaje inicial, Tailwind supera a la librería en términos de velocidad de desarrollo una vez superada la etapa inicial. Tailwind facilita la generación inmediata de los estilos directamente en el lugar donde se define el HTML y JavaScript, simplificando el flujo de trabajo al evitar el manejo de múltiples archivos. Además,

permite la reutilización de componentes de React personalizados por el equipo, combinando la flexibilidad deseada con la eficiencia de desarrollo.

Atributos	CSS Puro	Styled Components	Tailwind
Flexibilidad	Máxima, pero sin guías estructuradas.	Alta, con estilos basados en componentes.	Alta, con clases utilitarias y reutilizables.
Curva de aprendizaje	Depende del conocimiento en CSS.	Moderada, requiere entender CSS.	Moderada, necesidad de saber CSS y entender las clases utilitarias.
Personalización	Completa, pero puede ser compleja y larga.	Muy alta, permite estilos dinámicos y tematización.	Alta, con personalización a través de configuración.
Mantenimiento	Puede ser complicado en proyectos grandes.	Simplificado por el enfoque de componentes.	Simplificado por el uso de clases predefinidas.
Popularidad	Universalmente conocido.	Muy popular en proyectos basados en React.	Crecientemente popular por su enfoque de diseño rápido.
Integración	Universal.	Requiere React.	Fácil en la mayoría de los entornos de desarrollo.

Tabla comparativa de manejo de estilos

SocketIO

Para realizar una conexión en tiempo real entre los sistemas de *TuBocado.MarketFE* y *TuBocado.RestaurantFE* utilizamos Socket.IO, una librería de Javascript que permite y facilita la comunicación en tiempo real entre cliente y servidor. Una de las razones por la que elegimos esta librería es que no solo comunica en tiempo real sino que también maneja automáticamente casos complicados como la desconexión y reconexión de los clientes. Otro punto a destacar es que no utilizamos sockets directamente ya que es muy complejo de implementar y se necesita agregarle mucha seguridad, por lo que optamos por una librería.

Un factor clave que hizo que la integración de un sistema en tiempo real sea más sencillo fue la decisión de utilizar *ReactJS* y *Node.js* en el *frontend* y en el *backend* correspondientemente. Esta compatibilidad permitió, no solo que la

integración sea más simple de desarrollar, sino que se favorezca la coherencia en el código y la eficiencia ya que asegura una integración optimizada.

Por otro lado, si uno de los servicios no estuviera basado en JavaScript, no solo aumentaría la curva de aprendizaje ya que habría que entender cómo se integran los sockets en ambos lenguajes, sino que aumentaría mucho la complejidad para integrar los sistemas y generaría una carga de trabajo mayor para los integrantes del equipo.

Por lo tanto, la combinación de Socket.IO, *React* y *Node.js* crea una integración que ofrece ventajas significativas en términos de desarrollo, seguridad, rendimiento y experiencia del usuario, permitiéndonos construir aplicaciones eficientes y fácilmente escalables, que pueden manejar comunicaciones complejas en tiempo real de manera fluida y confiable sin la necesidad de implementar manualmente sockets y tener que conectarlo con los diversos sistemas, tanto de *backend* como los dos *frontends*.

A.5. Descripción de estructura de las APIs

Para diseñar nuestras APIs, nos basamos en los criterios REST. Iremos punto por puto mostrando las parincipales características cumplidas de REST sobretodo para la API de servicio de puntos ya que el cliente ademas, nos solicitó una especificación de la API de servicio de puntos. Esto se debe a que dicho servicio es fundamental para él y quiere reutilizarlo en otro tipo de servicios, por esto pidió que sea completamente desacoplado del resto. Por eso, solicitó la creación de una documentación de especificación de la api por si quiere integrarlo con otro tipo de aplicaciones.

Endpoint / Recurso

En lugar de tener servicios que exponen diversos comportamientos, exponemos endpoints asociados a los diferentes servicios y comportamientos, exponiendolos mediante URIs. Seguimos diversos criterios, en primer lugar, utilizamos sustantivos en lugar de verbos, por ejemplo, si queremos obtener el *user* 2, hacemos GET/ {{url}}/users/2/points en lugar de GET/ {{url}}/gerUserPoints/2. También podemos observar que cumplimos con el criterio de que estos sean

sencillos, en minúscula y plural ya que, como se puede observar, dice “points” en lugar de “point” y está todo en minúscula. Esto lo tomamos y adoptamos durante todo el desarrollo y fuimos consistente con todos los endpoints de los sistemas.

Verbos HTTP

Con lo que respecta a los verbos, utilizamos 4 verbos siguiendo los criterios REST: GET, POST, PUT y DELETE; para poder obtener(GET), agregar(POST), modificar(PUT) y eliminar(DELETE). Esto hace que cumpla y se entienda por sí solo lo que se quiere hacer con cada recurso.

Manejo de errores

Como se sabe que el que consume nuestras Web APIs ve todo desconocido, buscamos tener un buen manejo de errores siguiendo estos criterios para visualizar cómo usar la API. Por lo que como recomienda, inicialmente comenzamos con tres códigos de error básicos:

200 - Ok

400 - Bad Request

500 - Internal Server Error

Luego al encontrar errores más específicos fuimos ampliando, agregando el 401 - Cuando no tenía autorización al recurso, y 404 cuando algo de lo que busca el usuario no se encuentra.

Todas las *requests* que se le realizan a las APIs son "stateless", al no conservar un estado activo del lado del servidor, para la correcta utilización de cualquiera de los recursos.

Estructura de respuesta

El formato por defecto de las APIs es JSON. Pero tuvimos en cuenta además los criterios como usar JSON por defecto, siguiendo la convención de JS para el nombramiento de atributos y usamos camelCase.

Descripción general de códigos de error

Como indicamos al principio, para el desarrollo de las APIs manejamos los errores siguiendo REST, utilizando diversos tipos de códigos para cada ocasión. Para esto, utilizamos los códigos de error de las familias 200, 400 y 500. Las de 2xx para indicar cuando una solicitud se realiza de forma correcta, 4xx para indicar que existió algún tipo de error por parte del usuario y por último 5xx para indicar cuando existe algún error pero del lado del servidor.

Para poder manejar estas excepciones, inicialmente habíamos colocado en la capa de *controller* los *try* y *catches* para obtener todas las excepciones y mandar el código de error para cada una de estas. Al ver que el código era repetitivo, y cómo creamos una excepción para cada tipo de error que se quería, lo que hicimos fue capturarlas en un manejador de excepciones más arriba. Este filtro nos permitió obtener la excepción que se ejecutó y anexar el mensaje a la respuesta. En muchos casos simplemente mostramos el que venía con la excepción. Esto fue de gran utilidad para no repetir muchos *try & catch* en los métodos de los *Controllers*.

A continuación se puede ver en detalle la tabla de utilización de código de errores:

Código de error	Como se utiliza
200	Código retornado cuando una petición fue exitosa.
201	Código retornado cuando una petición fue exitosa y se crea algo.
400	Se utiliza cuando alguno de los datos ingresados por los usuarios no son correctos. Tanto porque se pasó un campo vacío, o falto alguno o se ingresó algo fuera de rango
404	Este código de error se utiliza para indicar cuando uno de los datos ingresado por los usuarios es incorrecto y hace referencia a un objeto que no existe. Sobre todo cuando se realizan operaciones por Id como GET o DELETE by id.
401	Este código es utilizado cuando el usuario no tiene un token o tiene un token invalido y está intentando acceder a recursos privados.
500	Código utilizado para indicarle al cliente cuando ocurrió un error por parte del servidor. Son errores no manejados por nosotros.

Tabla de utilización de código de errores

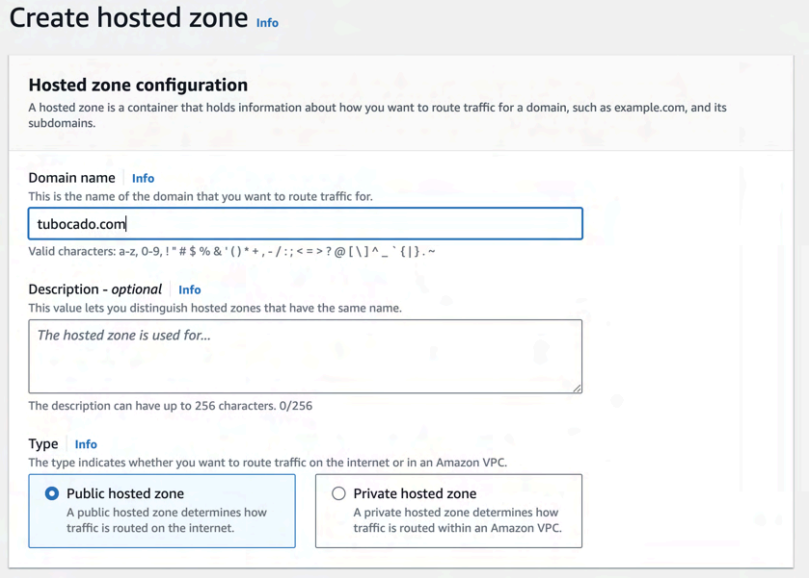
Descripción de los *resources* de la API puntos

Con lo que respecta a la descripción de los *resources*, en el anexo, se proporcionan detalles específicos para los recursos de la API del servicio de puntos, incluyendo los “resources” asociados a cada uno. Para compilar esta información, utilizamos POSTMAN, aprovechando que ya contábamos con la documentación de cada endpoint. Así, para cada recurso, se describirá el endpoint correspondiente, su propósito, los encabezados (*headers*) requeridos, y los parámetros que deben incluirse en el cuerpo del mensaje (*body*).

A.6. Guía de deployment

Para poder realizar el deploy de todo los sistemas, si bien se hacían automáticos tuvimos que seguir varios pasos para configurar todo.

1. **Asegurarse de que el cliente tenga un dominio comprado**. En este caso el cliente ya tenía el dominio TuBocado.com por lo que no fue un problema.
2. **Ir a Route 53 y crear una hosted zone**, allí podemos definir luego los alias para cada uno de nuestros sistemas.

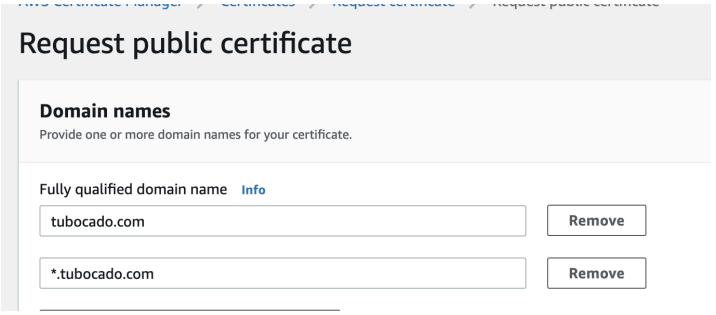


Crear hosted zone en AWS

a. Una vez creada hay que entrar al record y agarrar los valores de los dns para dárselos al cliente y que este los configure dónde está el dominio comprado

3. **Ir a Certificate Manager AWS:**

a. Apretar en “request Certificate”, por default la opción va a ser public certificate y hay que dejarla así.
b. Definir el dominio que va a ser tubcoado.com y aclarar que cualquier dominio antes tambien, *.tubocoado.com

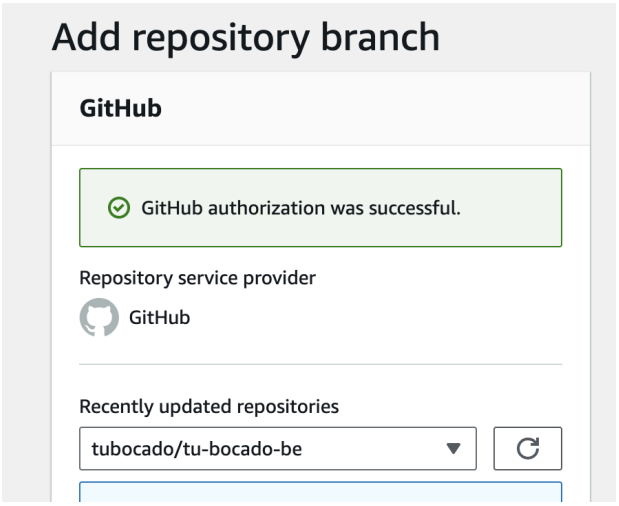


Solicitar certificado en AWS

c. Validar el certificado, para esto hay que crear el certificado y luego apretar el botón que dice “Create records in Route 35” para que se sincronice con el dominio.
d. Una vez que esté validado ya estará disponible el certificado
e. Volver a Route 53 y solicitar los dominios para todas las aplicaciones necesarias.

4. **Ir a Amplify AWS:**

a. Seleccionar “Host your webapp” y seleccionar que se quiere conectar con Github.
b. Seleccionar continuar y hay que dar los permisos necesarios para que AWS se pueda conectar con Github.
c. Allí se debe seleccionar el repositorio y posteriormente la rama que se quiere sincronizar

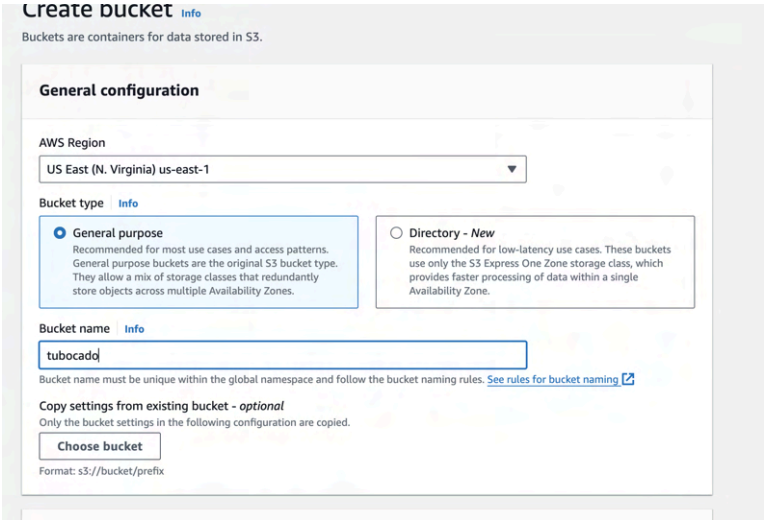


Conectar repositorio a Amplify en AWS

d. Agregar variables de entorno correspondientes en “Environment variables”
e. Ir a “Domain management” y seleccionar el dominio
f. Repetir esto para cada una de las web apps alojadas

5. Crear bucket s3 para imágenes:

a. Ir a S3 y crear un bucket

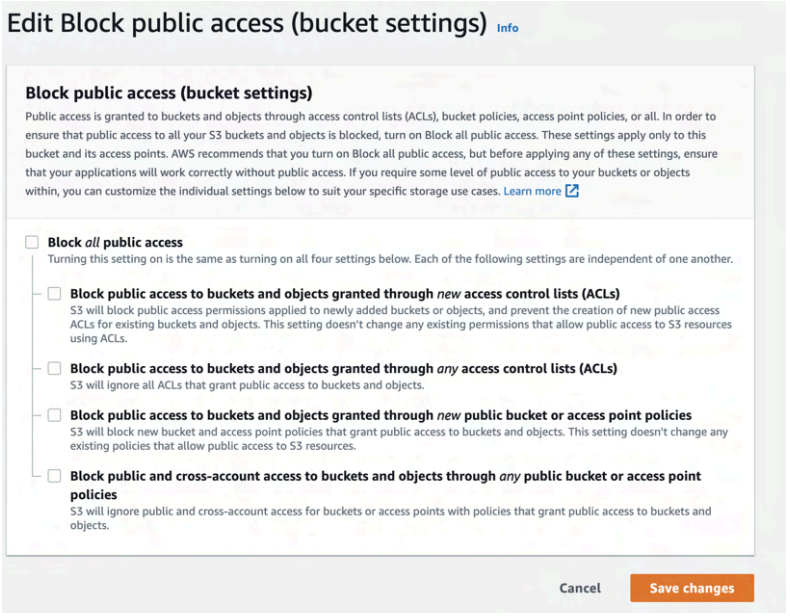


Crear bucket en S3 en AWS

b. IMPORTANTE: es importante habilitar el acceso público de las cosas guardadas en los buckets de S3 para que se puedan ver sin

autenticación. Para esto hay que ir al bucket creado y seleccionar la ventana de permisos:

Luego se debe deschequear los checkboxes por defecto para que quede de la siguiente manera:



Dar acceso público al bucket de S3 en AWS

c. Por último agregar una policy:

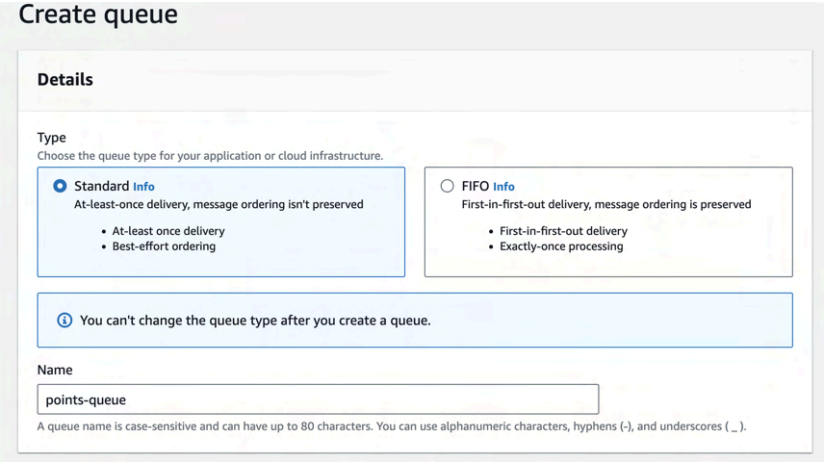
```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Principal": "*",
         "Action": "s3:GetObject",
         "Resource": "arn:aws:s3:::arq-dev-bucket/*"
      }
   ]
}
```

Agregar policy de S3 en AWS

Esto permite que se publicó.

6. **Crear SQS cola para la cola de los puntos:**

a. Ir a SQS y apretar en “Create queue”
b. Crear una “Standard queue”



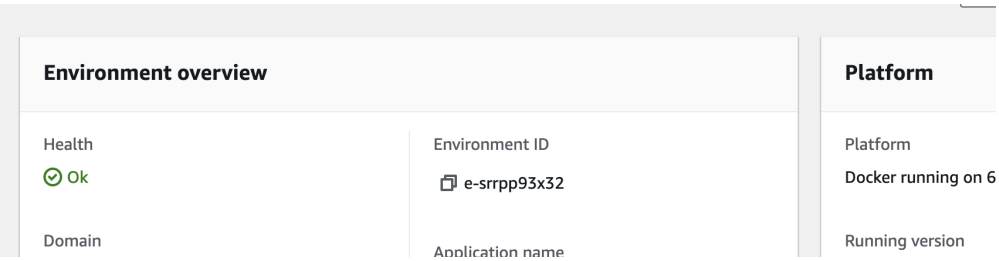
Crear cola de mensajes de SQS en AWS

7. Crear RDS base de datos para ambos servicios:

a. Crear base de datos, easy create con la db más pequeña.
b. Luego ir a los security groups y configurar las inbound rules para que acep el tráfico de la ip de la computadora que se quiere conectar y posteriormente repetirlo para conectar la instancia de *elastic beanstalk*
c. Repetir para ambas bases de datos

8. Crear instancias en elastic beanstalk para ambos servicios:

a. Apretar “Create new environment”
b. Rellenar los datos de application name, versión, etc.
c. En la plataforma seleccionar Docker y adjuntar un zip que incluye el Dockerfile con el código necesario. Esto posteriormente será conectado con AWS Pipeline.
   agregar load balancer y configurar https linkeado al certificado que solicitamos
d. Seleccionar los roles correspondientes
e. Saltear las partes opcionales y finalmente llenar las variables de entorno
f. Deberá quedar así:

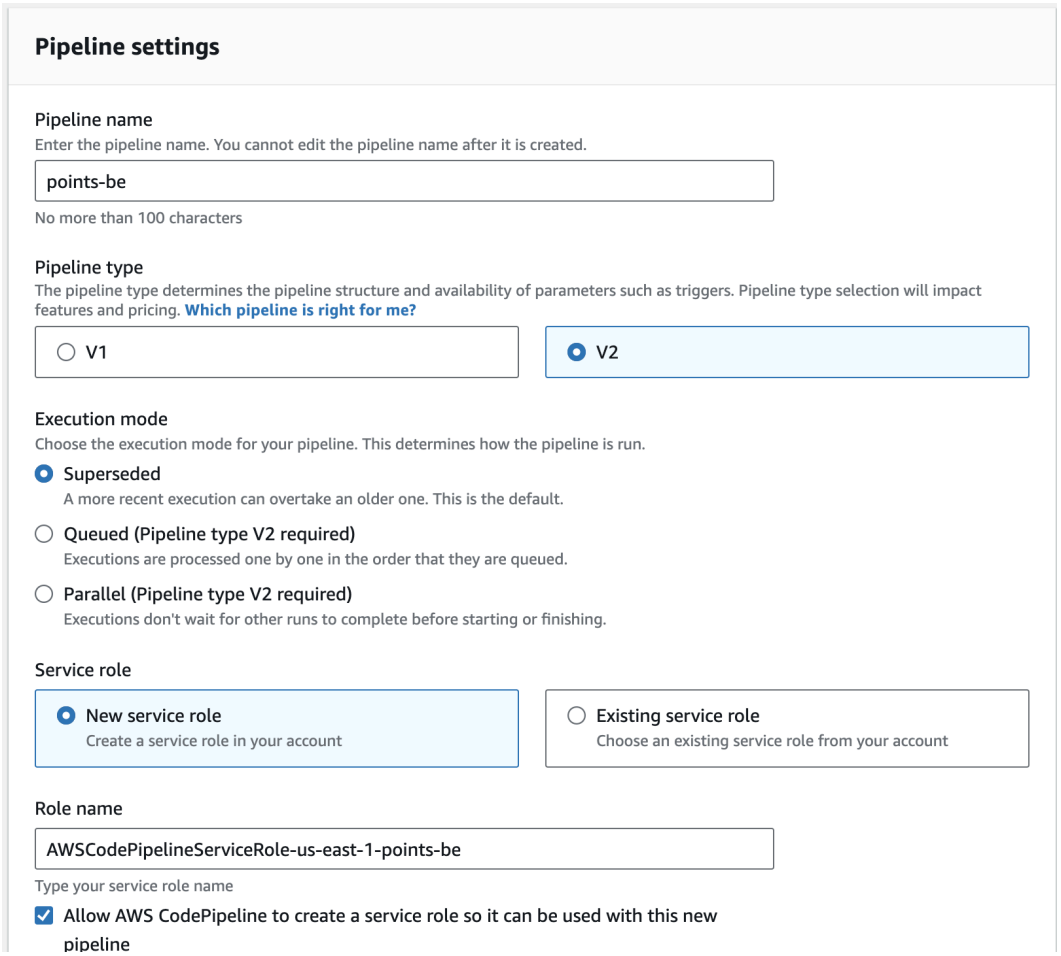


Environment deploy en elasticBeanstalk

g. Una vez terminado, volver a la configuración y setear el load balancer y en los Listeners agregar el puerto 443 con el certificado creado anteriormente asociado a una ruta creada en Route 53.

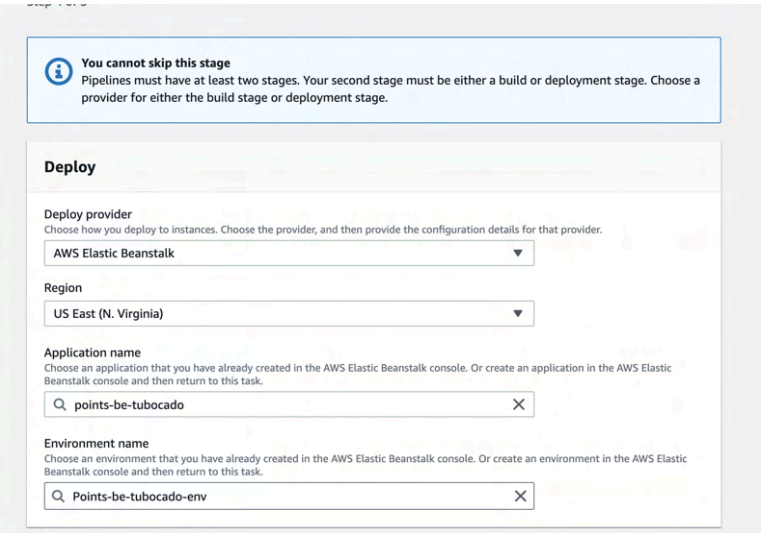
9. Conectar con aws Code Pipeline:

a. Crear pipeline



Crear pipline en AWS

b. Seleccionar como source Github
c. Seleccionamos el repo y la branch
d. Conectar a la instancia deseada



Elegir instancia de elastic Beanstalk

e. Repetir para ambas instancias

A.7. Especificación de API servicio de puntos

Users

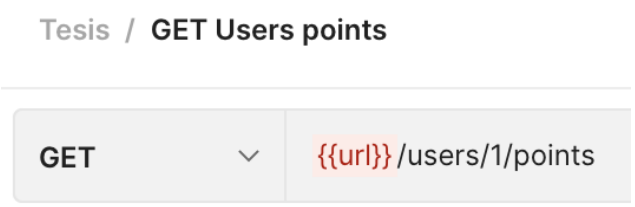
base URL {{url}}/v1/users

Resources

1. **Titulo**: GET Users points

   Descripción: devuelve todos los puntos de un usuario en particular

   Formación de la ruta:



Endpoint GET usuarios

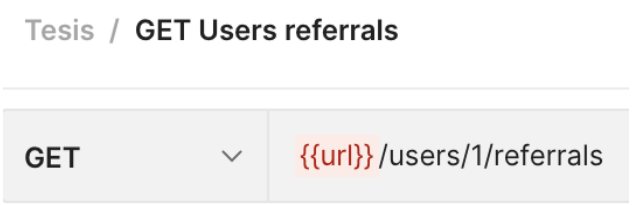
Necesario: `x-points-token` para la autorización

Devolución: json con el atributo “points” y la cantidad de puntos del usuario

2. **Título**: GET Users referrals

   Descripción: devuelve todos los referidos de un usuario en particular

   Formación de la ruta:



Endpoint GET referrals de usuarios

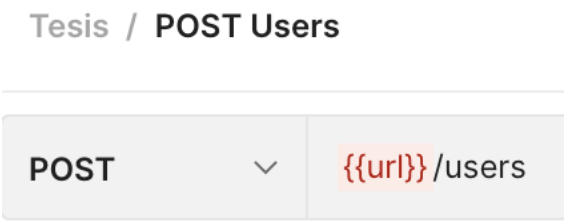
Necesario: `x-points-token` para la autorización

Devolución: Json con los atributos “tree”, “amount” y “levels” en donde indica el árbol de referidos que tiene el usuario, la cantidad de niveles que tiene formados y la cantidad de referidos del usuario.

3. **Título**: POST Users

   Descripción: crea un nuevo usuario con el referido asociado

   Formación de la ruta:



Endpoint POST usuarios

Necesario: `x-points-token` para la autorización

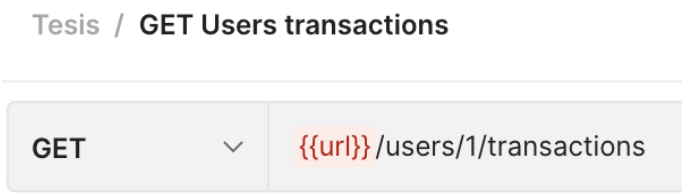
Body: json con los atributos “user_id”, “referred_by” en donde indica el usuario que se registró y el id del usuario que lo referencio.

Devolución: OK

4. **Título**: GET Users transactions

   Descripción: devuelve todas las transacciones de un usuario

   Formación de la ruta:



Endpoint GET transacciones del usuario

Necesario: `x-points-token` para la autorización

Devolución: json con los atributos “output”, “input” y “transactions” en donde indica las transacciones que le generaron Bocados y las que le sacaron y una lista entera de ellas.

Parameters

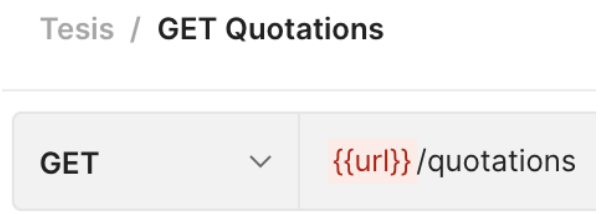
base URL {{url}}/v1/quotations

Resources

5. **Título**: GET quotations

   Descripción: devuelve todas las cotizaciones del sistema, es decir las cotizaciones de acumulacion y gasto de puntos.

   Formación de la ruta:



: Endpoint GET cotizaciones

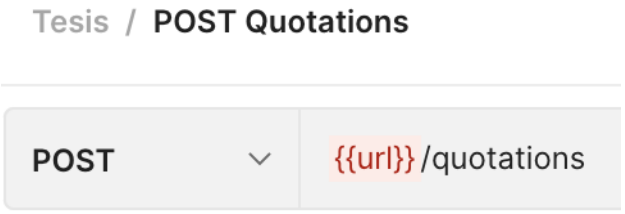
   Necesario: `x-points-token` para la autorización

   Devolución: json con el atributo “points_accumulation_rate” y “points_redemption_rate” en donde indica la cotización de cada uno

6. **Título**: POST quotations

   Descripción: crea cotizaciones nuevas en el sistema, es decir las cotizaciones de acumulacion y gasto de puntos.

   Formación de la ruta:



Endpoint POST cotizaciones

   Query Params: opcional, si se da una fecha devuelve las cotizaciones para esas fecha, de lo contrario devuelve los activos

   Necesario: `x-points-token` para la autorización

Body: json con el atributo “points_accumulation_rate” y “points_redemption_rate” en donde indica la cotización nueva creada de cada uno

Devolución: OK

Levels

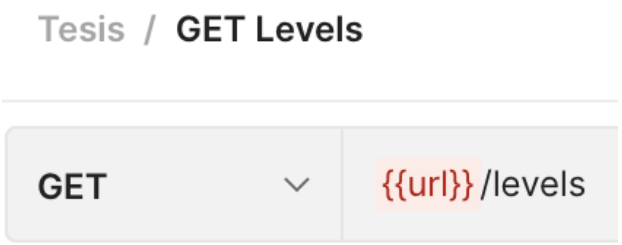
base URL {{url}}/v1/levels

Resources

7. **Título**: GET levels

   Descripción: devuelve todas los niveles del sistema, es decir la cantidad de niveles hacia arriba que generan puntos los referidos y el porcentaje que gana cada nivel

   Formación de la ruta:



Endpoint GET niveles

   Necesario: `x-points-token` para la autorización

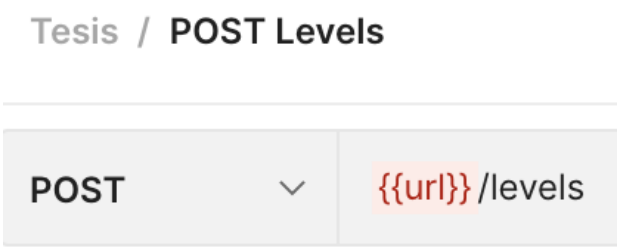
   Query Params: opcional, si se da una fecha devuelve los niveles y porcentajes para esa fecha, de lo contrario devuelve los activos

   Devolución: json con un array con “level_number” y“ percentage” de cada uno de los niveles

8. **Título**: POST levels

   Descripción: crea una configuración de niveles nueva con los porcentajes de ganancia para cada uno de los niveles

Formación de la ruta:



Endpoint POST niveles

Necesario: `x-points-token` para la autorización

Body: json con un array con level_total con la cantidad de niveles que se crearán y un array con “level_number” y“ percentage” de cada uno de los niveles

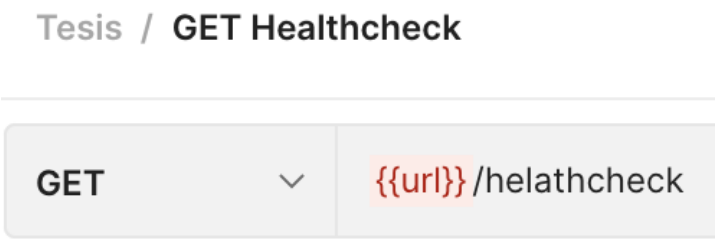
Devolución: OK

Healthcheck

9. *Título*: GET healthcheck

Descripción: se fija si el servicio está bien

Formación de la ruta:



Endpoint de health check

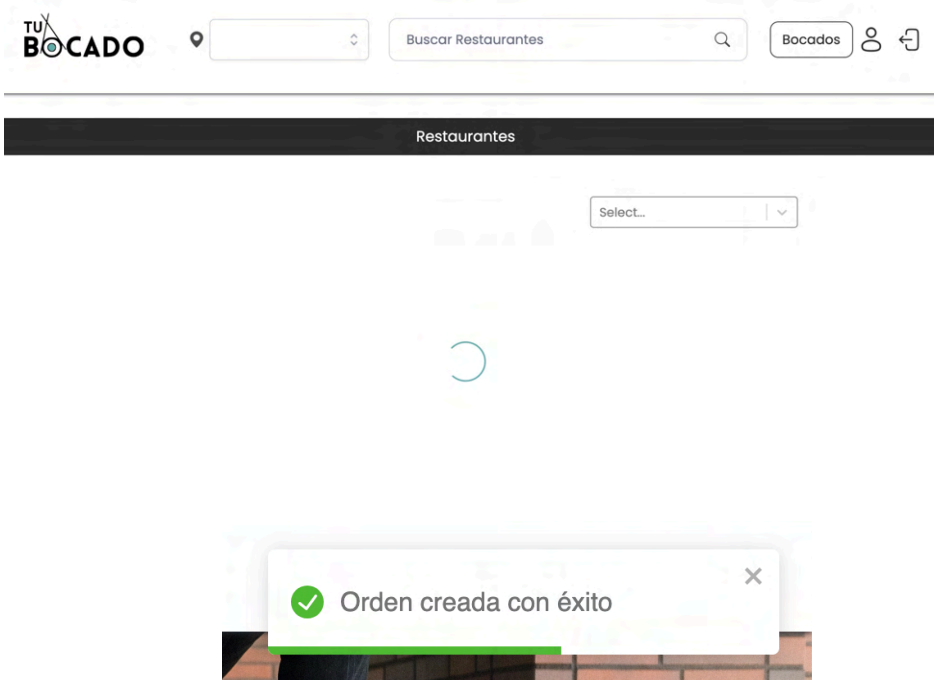
Devolución: OK o error

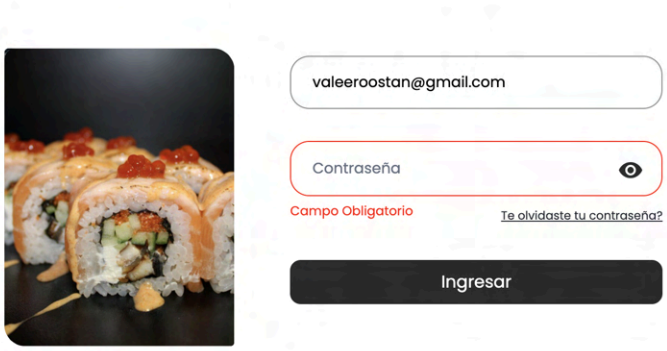
Es importante señalar que algunas operaciones dentro de nuestro sistema no se ejecutan a través de HTTP, sino que se manejan mediante una infraestructura de colas de mensajes, tal como detallamos en secciones de arquitectura. Este enfoque nos permite aprovechar la eficiencia que ofrecen las comunicaciones asincrónicas para procesos críticos. Las operaciones que utilizan este mecanismo son:

- **Creación de Puntos**: Este proceso asincrónico permite agregar puntos a la cuenta de un usuario, garantizando una gestión eficaz y escalable del sistema de recompensas.
- **Reembolso de Puntos**: Facilita la devolución de puntos a los usuarios de manera eficiente, asegurando que las operaciones de reembolso se manejen de forma robusta y confiable.
- **Deducción de Puntos**: Permite restar puntos de la cuenta de un usuario, optimizando el manejo de deducciones y ajustes en el balance de puntos de manera asincrónica.

A.8. Especificación de cumplimiento de heurísticas

1. **Visibilidad de sistema:** Evidencia de incorporación de *loaders,* feedback para usuarios y mensajes de errores

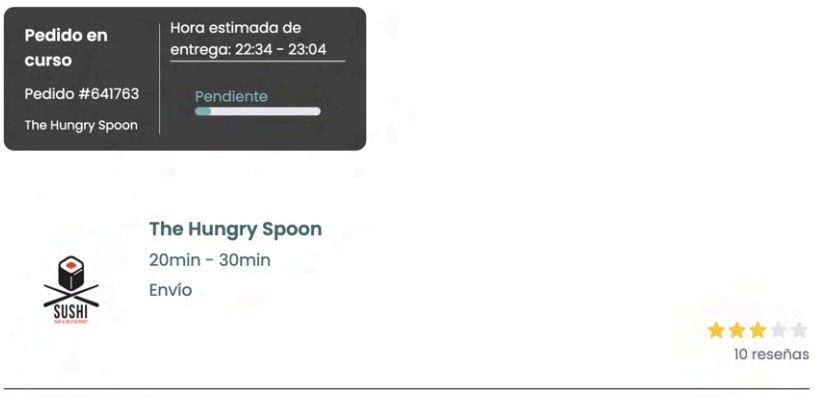




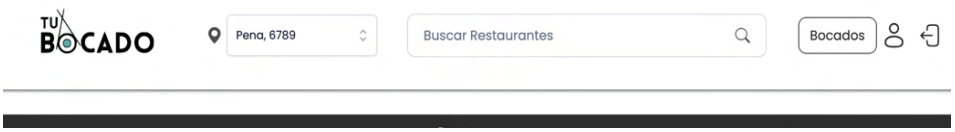
2. **Coincidencia entre el sistema y el mundo real:** Evidencia de *sidebars* que muestran cómo se une el sistema con el mundo real.



3. **Dale al usuario el control y la libertad:** Evidencia de agregado de *shortcuts, en el* inicio, *sidebars y headers*







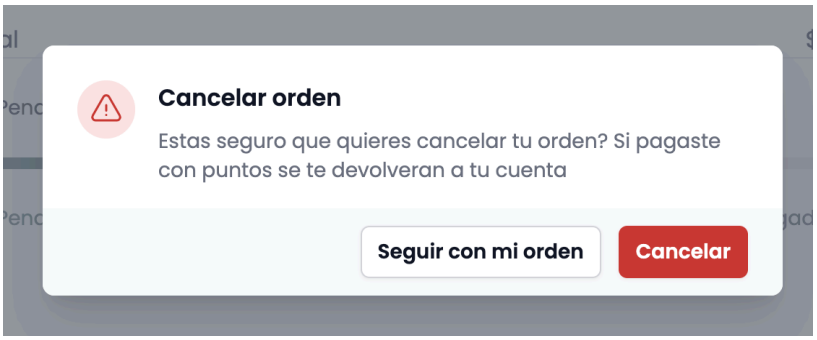
4. **Consistencia y estándares:** Evidencia terminología en todos los sistemas. Por ejemplo, los Bocados.





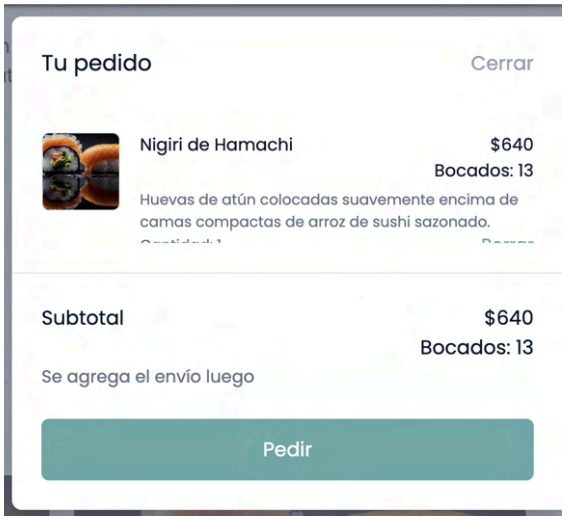


5. **Prevención de errores:** Evidencia de prevención de errores



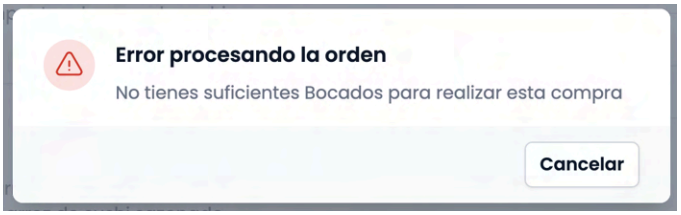


7. **Reconocer en lugar de recordar:** Evidencia de recordatorios y resúmenes para los usuarios antes de realizar acciones.





9. Ayuda al usuario a reconocer, diagnosticar y recuperarse: Evidencia de diagnostico de problemas



A.9. Matrices de Pugh - Metodología

Atributo	Peso	Kanban	SCRUM
Entregas en tiempo fijo	1	0	1
Fecha de finalizacion inmovil	1	1	1
Baja burocracia	2	1	0
Nivel de control	2	0	1
Experiencia del equipo	3	0	1
Facil de implementar	2	1	0
Suma		4	6
Ranking		2	1

A.10. Entrevistas e inmersiones y los resultados obtenidos

En esta sección encontramos fragmentos transcritos de las entrevistas llevadas a cabo durante el proceso de empatía en *Design Thinking*, los cuales se consideran destacados, manteniendo el anonimato y la privacidad de los datos de los entrevistados:

Entrevista Restaurante 1

Entrevista al dueño del restaurante 1, restaurante de sushi que hoy en día hace uso de una plataforma de envío de comida rápida conocida.

> **Pregunta**: ¿Cuáles son las principales problemáticas o desafíos que ha experimentado el restaurante al utilizar la aplicación?
>
> **Respuesta**: *El local contaba con cadeteria propia, al comienzo (2019) se encontraron problemas con cancelamiento de clientes, la aplicación no les avisaba de la cancelación y ellos mandaban el pedido igual (había falta de comunicación). Supuestamente te reembolsa el 50%, pero no era en todos los casos.*
>
> *[...]*
>
> *Para cobrar utilizaban notas de créditos, era complicado al comienzo, tenganlo en cuenta cuando se lancen al mercado. Después mejoró, se mandaba mail con el baucher y comprobante de venta. El local cancelaba al menos dos pedidos*

por mes, para comunicarlo con la aplicación tenían que mandar un mail y ellos informan al usuario.

Pregunta: ¿Ha habido problemas con la precisión de los pedidos, la entrega o la comunicación con los clientes a través de la plataforma?

Respuesta: *Era complicado transmitir los problemas, cuando la aplicación emite la orden los datos del cliente no aparecían por protección de datos, el local requería la dirección porque era delivery propio, como solución, generaron su base de datos con clientes nuevos para saber su direcciones. En este momento se podía integrar la facturación del restaurante a la app.*

Pregunta: ¿Qué llevó al restaurante a unirse a la aplicación? ¿Qué beneficios o ventajas ofrece la plataforma?

Respuesta: *Uno de los socios ya tenía una sucursal, fue unos de los primeros 50 restaurantes en entrar a la app, y consiguieron un descuento por usar delivery propio, les cobraban el 9%. El entrevistado afirma que la app sirve para no tener cadetes inactivos, ellos tenían cadetes con sueldo fijo, más el cobro por cada envío.*

Los cadetes que hacían más horario sacaban 30.000, 40.000 al mes.

El local era únicamente de delivery, 70% de las ventas provenían de la aplicación y el resto de llamadas directas al local.

Pregunta: ¿Cómo ha simplificado el proceso de pedidos y entregas para el restaurante?

Respuesta: *Al comienzo no simplificaba tanto, en 2019 se caía bastante la app, y whatsapp no te limitaba las listas de difusión y con la base de datos de los clientes que tenían (3000) cuando se caía la plataforma mandaba para todos lados para ver si alguien quería delivery, Y haciendo eso no perdía las ventas. La app se cae para todos, no solo para ellos.*

Pregunta: ¿Hay alguna característica o herramienta específica que le gustaría que se implementara en la plataforma?

Respuesta: *En este periodo la parte de integración y editar precios, descripción de productos era por mensaje, Lo tenias que hacer vos.*

Ahora es un toque cambiarlo todo, por sesión, precio descripción. Antes se complicaba tener la carta al día. Antes tenías platos principales que figuraban como entrada o al revés.

[...]

Además, de alguna forma lograr que no nos sea tan caro a nosotros vender comida por ahí, lograr que todos ganemos.

Pregunta: Los picos de demanda, como los días con mayor cantidad de pedidos, ¿representan un problema para el restaurante? ¿Cómo se manejan estos momentos de alta demanda?

Respuesta: *Picos viernes y sábados, el viernes 17:30 a 21:00 pico máximo del día, 80 % de los pedidos ese día. Los sábados igual pero una hora corrido.*

Viernes sábado domingo lunes jueves miércoles martes, días de más venta a menos.

Días de demanda tenía un cocinero más y dos motos más. Viernes y sabado tenia entre 8 y 10 motos. El domingo 5, 6 motos, lunes y jueves 4,5. Demás días 3 o 4, capas si había un partido o algo se manejan mas motos.

Demoraba 30 o 40 min, en algunas zonas 1 hora.

Pregunta: ¿Cómo varían los niveles de ventas por fecha? Ej: Por semana, por día de la semana, por mes, por feriado, por temporada / estaciones.

Respuesta: *Está arriba, por días y festivos, pero por mes es más o menos lo mismo. Enero podría bajar un poco porque la gente se va para afuera. "Se facturaba linealmente".*

Pregunta: En los días de menos demanda, ¿utilizan algún tipo de promoción para fomentar las ventas?

Respuesta: *Depende mucho del humor, la semana pasada estuvo media muerta, quiero más ventas. Esto lo hacía por la lista de difusión de whatsapp, la app dejaba pero por la forma de mail que no era muy eficiente.*

Pregunta: ¿De qué formas ves viable atacar la problemática de tener la cocina con poco trabajo, algunos días de la semana?

Respuesta: *Fomentando la venta todos los días, no llegar a por igual porque creo que nunca lo lograríamos, pero intentando emparejar.*

Pregunta: ¿Existen desafíos o consideraciones adicionales al manejar el *delivery* por cuenta propia?

Respuesta: *Se complica, con el tema de conseguir motos confiables. Tener sus motos era mejor para manejar mejor el tiempo de entrega, complicado que no rompan las motos, que te cumplan. La responsabilidad del cedente era hacer el servicio de la moto etc, la mitad no lo hacían. Ejemplo si llueve algunos no te van. Con este aspecto la aplicación tiene un poco más de seguridad.*

Pregunta: ¿Cómo se maneja la facturación y el pago por parte de las plataformas? ¿Qué métodos de pago utilizan? ¿Costos y comisiones?

Respuesta: *Te integra con visa master, etc. El 80% de los pedidos es online y hay muy poco contado, lo que era crédito a los 21 días lo tenían. En algunos casos, contenían la base de datos de whatsapp lo pedían por la app y le giraban directo mandando comprobante por whatsapp.*

Pregunta: ¿Ha tenido algún problema relacionado con la facturación o retrasos en los pagos?

Respuesta: *Si pero no es por parte de la app, de parte de bancos, partes administrativas o por la pasarle de pago.*

Entrevista Restaurante 2

Entrevista al dueño del restaurante 2, restaurante de sushi que hoy en día hace uso de una plataforma de envío de comida rápida conocida.

Pregunta: ¿Cuáles son las principales problemáticas o desafíos que ha experimentado el restaurante al utilizar la aplicación?

Respuesta: *Al principio la aplicación tenía monopolio y habían dos modalidades que era delivery propio y otra que era delivery del punto de venta (6% y 8%), lo manda con cadeteria con tu cadete eso y si usas la de la app entre un (15% y un 20%). Cuando arranca a haber competencia con otras plataformas va mutando a nivel internacional porque arrancan a cobrar más comisión de 16% a 17, 20%, hasta que hoy en día que se supone que es un buen cliente paga 23%. Las que tienen delivery propio no dejan más. Empezó a exigir que tenga bromatología. Tenía opciones de los primeros clientes les dejaba un radio de 10km y ahora redujeron a todo que es 3,3km. Antes él era uno y llegaba a carrasco y ahora no más.*

Controlan mucho el tiempo, cuando reciben el pedido, aceptan y ahí corre el tiempo. Hay puntos clave donde arranca. Cuando arranca a llegar al tiempo límite arranca a sonar una alarma y cuando se pasa cierra.

Te penalizan al punto que te cierran la aplicación 15 o 20 y eso hace que se cagan las ventas MUCHO. De 8 a 10:30 los findes y te cierran 10 minutos.

Plataforma de comida 2 entra solo con opción de cadeteria de Plataforma de comida 2. Por eso Plataforma de comida 1 saca la opción de que cada restaurante tenga su propio cadete. Subió la tasa a 22%. Compraba en base a descuentos y siempre pedía promoción continuamente y en base a eso hacía conocer a sus clientes y hacía facturar pero no tenía tanta rentabilidad. Se arranca a achicar después porque cuando llueve por los cadetes y te cierra un poco el radio y llegas más lejos porque se reducen a la mitad.

Sin tener estructura y un proyecto nuevo solo tenían que tener internet y le sacaban un 30% y el costo de ventas pasa entre un 33% con mercadería y se quedaban con un 40%. Esto para los micro emprendimientos, ahora tenían que tener empresas más regularizada y eso hizo que muchos dejaran de vendar

Hoy en día tiene un 65% plataforma de comida 1 y un 5% plataforma de comida 2 y un 50% por canales propios. Sacan mucho. Cuando tenían motos propios era más fácil

Compra el sushi en 2019, estaba con Plataforma de comida 1 y había cambiado a la plataforma de comida 2 y aseguro algo que si no vendía lo mismo que en Plataforma de comida 1 te devolvía la plata. Con esto se aseguró.

Pregunta: Porque no estaría en todas las plataformas?

Respuesta: *Las plataformas no cobran por entrar. Pero Plataforma de comida 1 te dice ¿trabajas con otros? 23% sino un 22%.*

Pregunta: ¿Cómo ha simplificado el proceso de pedidos y entregas para el restaurante?

Respuesta: *ventas MUCHO y publicidad.*

Pregunta: ¿Cuáles son los costos de tener un restaurante en la o las plataformas mencionadas?

Respuesta: *alrededor del 23%*

Pregunta: ¿Cree que la venta de sushi en plataformas de *delivery* presenta diferencias significativas en comparación con otros tipos de comida?

Respuesta: *plataforma de comida 1 tiene un rubro por tipo y podes pedir ampliar tu tiempo de hacer e igual te penalizan.*

Pez globo tiene un 20% de plataforma de comida 1 y 80% ventas propias. No limitan la cantidad de usuarios solo te penalizan por tiempo.

Los días que está saturado porque no llegan los cadetes y la gente los está esperando y muchas veces los cadetes ponen que recibió el pedido para que no lo penalicen pero en realidad nunca llegó y eso hace que muchas veces tengas que conectarte con Plataforma de comida 1.

Pregunta: ¿Cómo varían los niveles de ventas ventas por fecha? Por semana, por día de la semana, por mes, por feriado, por temporada / estaciones

Respuesta: *Si el fin de semana*

Pregunta: ¿El restaurante cuenta con su propio servicio de *delivery* o utiliza exclusivamente los repartidores de *Plataforma de comida 1*/*Plataforma de comida 2*/*Plataforma de comida 3*?

Respuesta: *Sí y generalmente tienen más radio.*

Pregunta: ¿Consideraría gestionar la entrega de pedidos por su cuenta en caso de que los costos de las plataformas se reduzcan?

Respuesta: *Si*

Pregunta: ¿Existen desafíos o consideraciones adicionales al manejar el *delivery* por cuenta propia?

Respuesta: *Radios más grandes si es propio del delivery.*

Con los deliverys de plataforma de comida 1 puede pasar que haya saturación y no te agarre ningún delivery o se generan situaciones tipo que un delivery dice que ya tiene el pedido y no lo tiene.

La plataforma de comida 1 arrancó a cobrar envío y antes a los cadetes no pagaban y empezaron a pedir que le paguen. Ellos pagan por zonas y lugares.

Pregunta: Tienen *delivery* por su cuenta? ¿El restaurante podría manejar el reparto de pedidos por su parte?

Respuesta: *En caso de no tener, ¿implicaría un problema pasar a tener un delivery propio en el caso que se reduzcan los costos de las plataformas como Plataforma de comida 1?*

Son necesarios e indispensables por los pedidos por llamada o por el contacto que ya generaste con clientes.

Días de lluvia sobre todo.

Pregunta: ¿Hay mucha demanda de *delivery*? O usualmente la gente va más al restaurante a comer?

Respuesta: *Generalmente casi todas son por pedido, no va casi nadie al restaurante. Mucha gente va a buscar las cosas allí y les cobra siempre lo mismo. No importa si llevan la comida o no.*

Pregunta: ¿Cómo se maneja la facturación y el pago por parte de las plataformas como *Plataforma de comida 1* y *Plataforma de comida 2*? ¿Qué métodos de pago utilizan? Costos y comisiones?

Respuesta: *Los métodos de pago los definen ellos: venden con tarjetas (nro comercio, cuenta visa y cuando visa líquida le paga visa a el).*

Respuesta: *Con Plataforma de comida 1, te hace una cesión de crédito o cobro "cede toda la cobranza de la tarjeta visa". plataforma de comida 1 te exige que le cedas el crédito de visa y él te devuelve la plata (crédito 21 días y debido 48 horas).*

Pregunta: Cuanto demora en pagar la plataforma? 60 días, 1 día?

Respuesta: *Vendes 100 mil pesos con visa entonces en lugar de cobrar con sushi go le paga a plataforma de comida 1 porque cede el crédito de ventas y Plataforma de comida 1. Demora 21 días demora en pagar en credito y debito 48 horas. Antes fiaba y ahora no.*

Entrevista Restaurante 3

Entrevista al dueño del restaurante 3, restaurante de comida con enfoque en sushi que hoy en día hace uso de más de una plataforma de envío de comida rápida conocida.

Pregunta: ¿Cuáles son las principales problemáticas o desafíos que ha experimentado el restaurante al utilizar las plataformas?

Respuesta: *Uno de los principales desafíos es mantener la calidad de nuestros productos durante el transporte. Además, enfrentamos dificultades para fidelizar a los clientes debido a la competencia desigual y los altos precios de las aplicaciones existentes.*

Pregunta: ¿Ha habido problemas con la precisión de los pedidos, la entrega o la comunicación con los clientes a través de la plataforma?

Respuesta: *Sí, tuvimos varios desafíos en varios aspectos. Tuvimos problemas con la precisión de los pedidos, que creo que fue por errores de interpretación o ingreso de datos incorrectos. Además, a veces tenemos retrasos en las entregas, generalmente por la disponibilidad de repartidores. En cuanto a la comunicación, a veces es limitada, lo que nos complica resolver rápidamente cualquier problema o consulta de los clientes.*

Pregunta: ¿Qué llevó al restaurante a unirse a las aplicaciones? ¿Qué beneficios o ventajas ofrecen?

Respuesta: Nos unimos a estas aplicaciones para ampliar nuestra base de clientes y facilitar el proceso de pedidos. Las principales ventajas incluyen una mayor visibilidad, conveniencia para los clientes y una infraestructura establecida para la entrega de alimentos.

Pregunta: ¿Cómo ha simplificado el proceso de pedidos y entregas para el restaurante?

Respuesta: El uso de las plataformas simplifica el proceso de pedidos porque nos da una interfaz bastante fácil de usar tanto para nosotros como para nuestros clientes. Creo que son de fácil acceso y con pocos pasos ya tienen la comida en su casa, no tienen ni que moverse. Además, nos permiten externalizar el proceso de entrega, lo que nos libera de esa responsabilidad. Por más que tiene sus contras no tener el control de la entrega, nos facilita mucho.

Pregunta: ¿Hay alguna característica o herramienta específica que le gustaría que se implementara en la plataforma?

Respuesta: *Nos gustaría ver una mejor integración con nuestro sistema interno de gestión de pedidos y quizás una opción para obtener retroalimentación instantánea de los clientes sobre sus experiencias.*

Pregunta: ¿Cuáles son los costos de tener un restaurante en la o las plataformas mencionadas?

Respuesta: *Los costos varían según la plataforma y el acuerdo específico que tengamos. Por lo general, incluyen una comisión por cada pedido y en algunos casos también hay tarifas mensuales o por servicios adicionales.*

Pregunta: ¿Considera que las plataformas son todos lo mismo? ¿Todos sacan el mismo porcentaje?

Respuesta: Aunque tienen funciones similares, cada plataforma tiene sus propias características y audiencias. En cuanto a las comisiones, pueden variar entre ellas y dependen de varios factores como el tipo de restaurante, la ubicación y los acuerdos comerciales.

Pregunta: Los picos de demanda, como los días con mayor cantidad de pedidos, ¿representan un problema para el restaurante? ¿Cómo se manejan estos momentos de alta demanda?

Respuesta: *Los picos de demanda pueden ser desafiantes, pero también son oportunidades para aumentar nuestras ventas. Para manejarlos, nos aseguramos de tener suficiente personal y suministros en esos momentos y también podríamos implementar restricciones en los pedidos para evitar la saturación. Y por otro lado, intentar emparejar los días de ventas, aunque sería muy difícil.*

Pregunta: ¿Cree que la venta de sushi en plataformas de *delivery* presenta diferencias significativas en comparación con otros tipos de comida?

Respuesta: *Sí, la venta de sushi puede presentar desafíos únicos por su naturaleza perecedera y a la necesidad de mantener la frescura y la presentación durante el transporte. Sin embargo, también tiene la ventaja de ser un producto popular y con alta demanda en el mercado de delivery hoy en día en Montevideo.*

Pregunta: ¿Cómo varían los niveles de ventas por fecha? Ej: Por semana, por día de la semana, por mes, por feriado, por temporada / estaciones.

Respuesta: *Las ventas suelen ser más altas los fines de semana, durante los días festivos y en ciertas temporadas como el verano. Sin embargo, también podemos experimentar fluctuaciones dependiendo de otros factores como eventos locales o condiciones climáticas, la lluvia suele ser un buen incentivo para el pedido de comida.*

Pregunta: ¿Cómo varían los niveles de ventas por lugar? Ej: Por barrio

Respuesta: *Las ventas pueden variar según la ubicación, siendo generalmente más altas en áreas con mayor densidad de población como Pocitos por ejemplo, o donde hay una mayor demanda de comida para llevar.*

Pregunta: En los días de menos demanda, ¿utilizan algún tipo de promoción para fomentar las ventas?

Respuesta: *Sí, en días de baja demanda a veces ofrecemos promociones especiales, descuentos o paquetes de comida para incentivar las ventas. Todo depende de la época del año que estemos y como venimos de ventas últimamente. Si no es necesario no lo hacemos.*

Pregunta: ¿De qué formas ves viable atacar la problemática de tener la cocina con poco trabajo algunos días de la semana?

Respuesta: *Podríamos desarrollar estrategias de marketing específicas para promover nuestro negocio en esos días, así como lanzar ofertas especiales y promociones para atraer a más clientes. También podríamos explorar la posibilidad de ampliar nuestro menú o diversificar nuestros servicios para atraer a diferentes audiencias.*

Pregunta: ¿El restaurante cuenta con su propio servicio de *delivery* o utiliza exclusivamente los repartidores de las aplicaciones?

Respuesta: *Actualmente utilizamos exclusivamente los repartidores de estas plataformas.*

Pregunta: ¿Consideraría gestionar la entrega de pedidos por su cuenta en caso de que los costos de las plataformas aumenten?

Respuesta: *Es una opción que consideraríamos si los costos de las plataformas aumentan significativamente y si vemos que podemos gestionar eficientemente el servicio de entrega por nuestra cuenta.*

Pregunta: ¿De ser el caso, existen desafíos o consideraciones adicionales al manejar el *delivery* por cuenta propia?

Respuesta: *Sí, tendríamos que considerar aspectos como la contratación y capacitación de personal de entrega, la logística de distribución y la inversión en vehículos y equipos de entrega.*

Pregunta: ¿Hay mucha demanda de *delivery* o usualmente la gente va más al restaurante a comer?

Respuesta: *La demanda de delivery ha ido en aumento, especialmente debido a los cambios en los hábitos de consumo y las restricciones relacionadas con la pandemia. Sin embargo, también tenemos clientes que prefieren venir al restaurante a disfrutar de la experiencia completa.*

Pregunta: ¿Cómo se maneja la facturación y el pago por parte de las plataformas? ¿Qué métodos de pago utilizan? ¿Costos y comisiones?

Respuesta: *La facturación y el pago se gestionan a través de las plataformas, que suelen ofrecer diferentes métodos de pago como tarjetas de crédito, débito o efectivo. Los costos y comisiones varían según el acuerdo comercial que tengamos con cada plataforma.*

Pregunta: ¿Ha tenido algún problema relacionado con la facturación o retrasos en los pagos?

Respuesta: *Hasta el momento no.*

Entrevista Ollas Essen

Pregunta: ¿Cómo registran y rastrean las ventas realizadas por los agentes en su sistema de Essen? ¿Utilizan alguna aplicación o herramienta específica para este propósito?

Respuesta: *Existe una aplicación, una vez que te unis al sistema de ollas essen, tu "patrocinadora" te asigna un número de usuario y con eso podes acceder a la página. Allí es donde se registra todo y se ve la información.*

Pregunta: ¿Qué información capturan al registrar una venta realizada por un distribuidor?

Respuesta: *Se registran los datos y pongo la venta allí, tengo que además adjuntar el recibo de pago ya que tengo que transferir la plata*

Pregunta: ¿Utilizan algún sistema de seguimiento o reporte para visualizar las ventas y los beneficios generados por cada agente?

Respuesta: *Si, por cada venta vas acumulando puntos y se puede ver en el sistema, además si sos patrocinador y tenes gente que patrocinas, por cada persona que vende también gana puntos y tienen objetivos de puntos para canjear cosas.*

Pregunta: ¿Cuál es la experiencia general de su empresa con el marketing multinivel? ¿Cuánto tiempo han estado implementando este modelo y qué resultados han obtenido?

Respuesta: *La verdad estoy hace poco y no tengo gente abajo por lo que no tengo tanta experiencia en eso, pero hasta ahora muy bueno*

Pero básicamente funciona así: por cada venta se ganan puntos y comisión. Además cada mes se fijan metas y si se cumple te dan regalos, hay un mínimo de puntos. También te ofrecen hacer equipo si quieres y cada persona que tienes debajo tenes que estar ayudando, respondiendo preguntas además de que ganas por cada persona que vende en tu equipo. No solo esto, sino que luego que tenes determinado número de equipo te arrancan a pagar por tener esa gente, entonces va acumulando más puntos y comienza a ser "ejecutiva" hay muchos rangos. La primera es "emprendedora" luego ejecutiva y sigue subiendo.

Con esto van escalando y van dando más premios, van pagando más comisiones, las invitan a reuniones y a otros países, le ponen límites para poder

subir de escalón y avanzar. Se ganan viajes como Punta cana, los vendedores de ella suman puntos para ella pero para el equipo también.

El primer mes si pasa los 400 puntos de venta (porque cada pieza tiene puntos) le dan un mate por ejemplo, después de 600 puntos más piezas. Te viene a tu domicilio.

Pregunta: ¿Han enfrentado algún desafío o problema específico en la implementación y gestión del sistema de marketing multinivel en Essen? ¿Cómo han abordado estos desafíos?

Respuesta: *no*

Pregunta: ¿Cómo arrancó a vender?

Respuesta: *Seguía gente en instagram y le escribió a una persona y le compró una olla y ella le ofreció para vender. Dijo que podía ser a través de instagram y no tenía que "atomizar".*

Pregunta: ¿Cómo manejan los pagos y las comisiones para los distribuidores en su sistema de Essen? ¿Utilizan alguna aplicación o plataforma para gestionar estos procesos? Es la misma aplicación donde se gestiona el sistema de puntos?

Respuesta: *Para pagar se paga por transferencia, y se adjunta el comprobante y luego esos puntos se suman automáticamente luego de que alguien revisa las ventas.*

Ahí registra los pagos puede ser por mercado pago o transferencia.

Pregunta: ¿Qué tipo de herramientas o recursos proporcionan a los distribuidores para facilitar la venta y promoción de los productos o servicios? ¿Tienen alguna aplicación móvil o plataforma digital que los agentes utilizan para realizar seguimiento de sus actividades y contactar a posibles clientes?

Respuesta: *Para los clientes la principal aplicación que se utiliza son las redes sociales, como mencionaba anteriormente, conoció las ollas y la posibilidad de venderlas mediante instagram y hoy en día es la principal aplicación que utiliza.*

Además también se usa mucho el "boca a boca" o contactar gente mediante wpp.

Pregunta: ¿Existen materiales de capacitación o programas de desarrollo profesional para los distribuidores? ¿Cómo fomentan el crecimiento y el éxito de los agentes en su sistema de multinivel?

Respuesta: *Si, tu patrocinadora o agente organiza reuniones de muchas personas,*

A la coach le puedes escribir por Whatsapp todo el tiempo, puedes tener charlas grupales e individuales y te explica todo. Hay dos grupos de wpp uno para nuevos usuarios y otro para más gente. Es como una pirámide.

Pregunta: ¿Cómo fomentan la comunicación y colaboración entre los agentes en su sistema en Essen? ¿Cómo sabes quién es el agente?

Respuesta: *Por los grupos de wpp que se generan de los agentes en Uruguay.*

Pregunta: ¿Organizan eventos o reuniones periódicas para reunir a los distribuidores y fortalecer la comunidad?

Respuesta: *Si, como menciono anteriormente, se juntan para informarse todos juntos y compartir estrategias y tips.*

Pregunta: ¿Dónde se registran los nuevos agentes interesados en unirse al sistema de Essen?

Respuesta: *Los registra la "agente" que tiene otro tipo de rol en la aplicación en la que puede crear su equipo y allí registrar nuevos vendedores para que puedan loguearse a la aplicación.*

En la página abajo te dice que te sumes como emprendedora ahí se puede subir, el número de emprendedor te lo dan, con la primera venta se registran como vendedora activa.

A.11. Benchmarking aplicación de ventas de comida 1

ANÁLISIS PLATAFORMA DE COMIDA 1

- ➔ Los restaurantes son "outlets"
- ➔ Se loguea un usuario que maneja muchos restaurantes y ahí puede elegir sobre cual quiere estar manejando la plataforma. Hay una opcion que es "all" y ahi los datos serían sobre todos



- ➔ Según qué restaurante esté seleccionado, qué datos se muestran. Por ejemplo en la sección reviews y ratings se muestran sobre ese específico y no sobre todos en general.



- ➔ En la barra de navegación tiene todas las secciones de manejo para la gestión de los restaurantes

➔ **DASHBOARD:**
- Business summary: acá se ven las órdenes y niveles de ventas en gráficas con opciones de filtrado variadas según la fecha. A parte de las gráficas hay datos resaltados como totales, promedios, etc.
- Live Ops Monitor: datos generales de los outlets en tiempo real, por ej: cuales están offline, órdenes canceladas para hoy, ordenes atrasadas para hoy, ratings de hoy.
- Operations health: rejections, avg preparation time, etc
- Customer reviews: Últimos comentarios de las personas
- En todas los recuadros con datos hay opciones de “More in reviews o reports”, es decir ir a ver más detalladamente en sección reviews o reports
- Marketing campaigns:

➔ **ONBOARDING:**
- https://www.youtube.com/watch?v=bR1dUUjOk28

➔ **MENÚ:**
- Según el restaurante seleccionado, se puede editar qué platos se venden en el menú del mismo.
- Hay un video tutorial de como se agregan productos al menu: https://www.youtube.com/watch?v=88zQQW674tE&t=2s
- Existen categorías y dentro de cada categoría van los platos. Se puede activar y desactivar cada categoría y plato
- Para cada artículo se pone una foto, nombre, precio

➔ **OPENING TIMES:**
- Acá se manejan los horarios y días de la semana que se está abierto

➔ **BRANCH STATUS:**
- Estado del local si está abierto o no

➔ **DISCOUNTS:**
- Aca se crean los descuentos y campañas -> No aplica

➔ **MARKETING:**

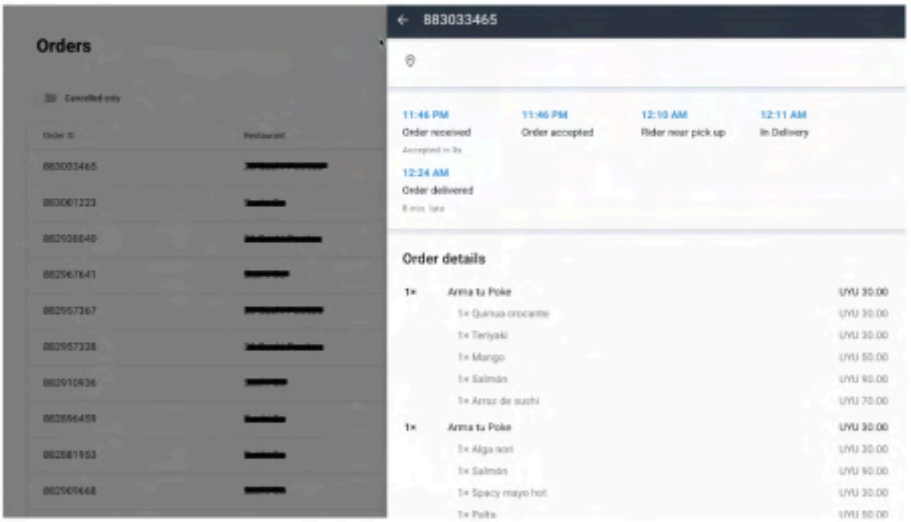
- Campañas para tener más clientes -> No aplica

➔ **REPORTS:**

- Sales & orders by day: Gráfica, rango fechas, cuantas succesfull y cuantas cancelled. Comparación entre cash y online sales.
  - Opción de export a excel o csv
- Sales y orders by hour: misma información pero cada hora del día, en columnas de x hora a x hora. El valor de las ventas en celdas clasificadas en colores más o menos intensos de rojos.
- Sales by month: Misma info pero en un rango de meses
- Sales by menu item:
- Operations: Order delay y preparation time

➔ **ORDERS:**

- Se muestran todas las órdenes con sus estados, fechas id, y se filtra por fecha y rangos
- Si se toca uno, se ven los detalles con todo lo que pidió y las fechas



➔ **FINANCE:**

- Estado de cuenta y contrato
- Muestra los saldos en rangos de fecha con las facturas
- Gestión de los cobros y gastos en la relación con Plataforma de comida 1

➔ **UNIVERSITY:**

- Aprender a mejorar el rendimiento de Plataforma de comida 1

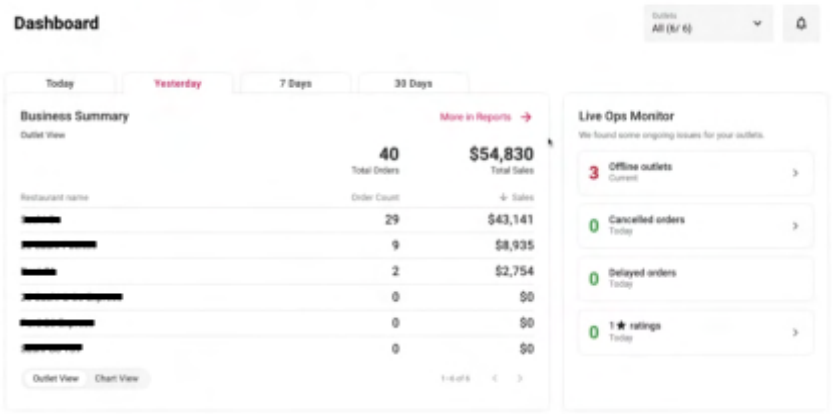
➔ **USER MANAGEMENT:**

- Manejo de usuarios, que roles tienen, que branches manejan, jerarquías

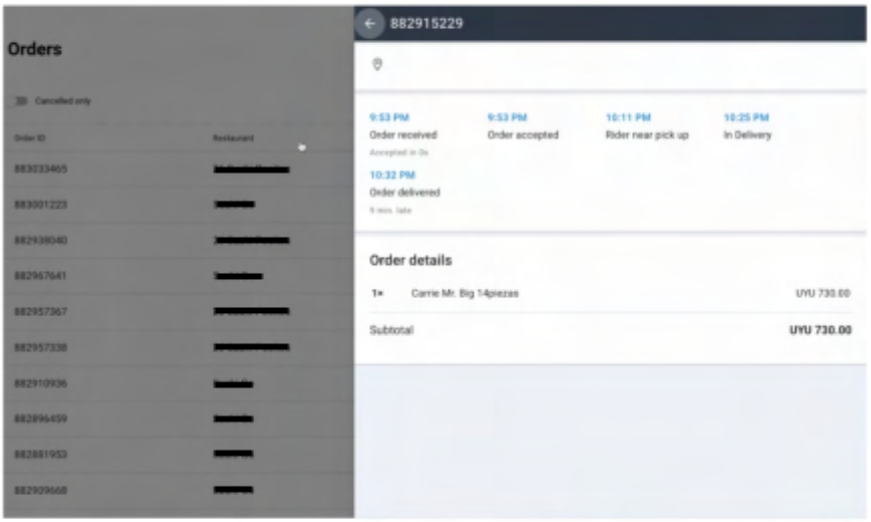
➔ **RATINGS AND REVIEWS:**

- Ver las reviews por restaurante y las calificaciones de cada uno y cada plato según las órdenes.

- Ej Dashboard:



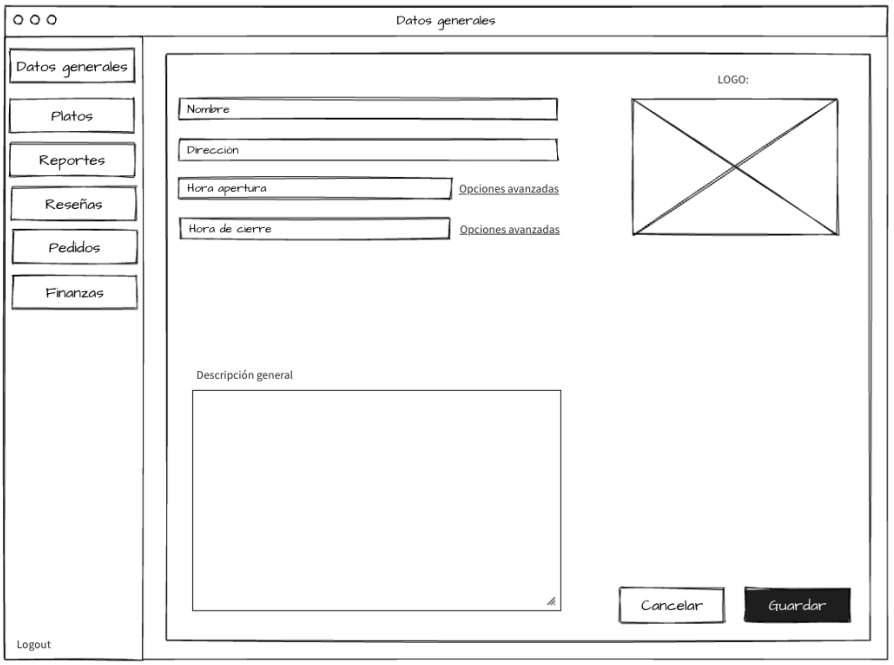
- Orders:



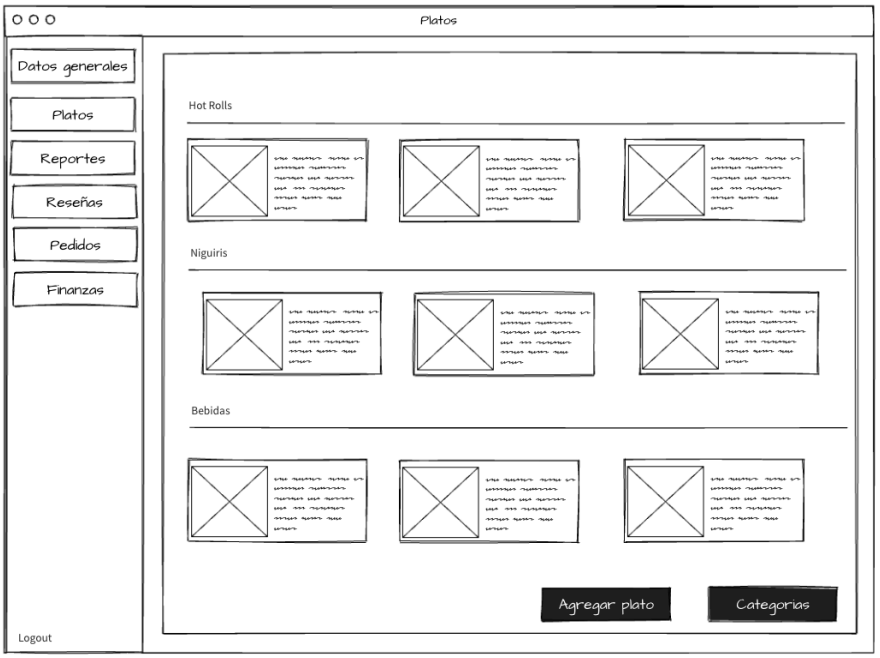
- Link explicacion subida productos: https://www.youtube.com/watch?v=88zQQW674tE&t=2s
- Link: https://www.youtube.com/watch?v=bR1dUUjOk28&ab_channel=

A.12. Pretotipo

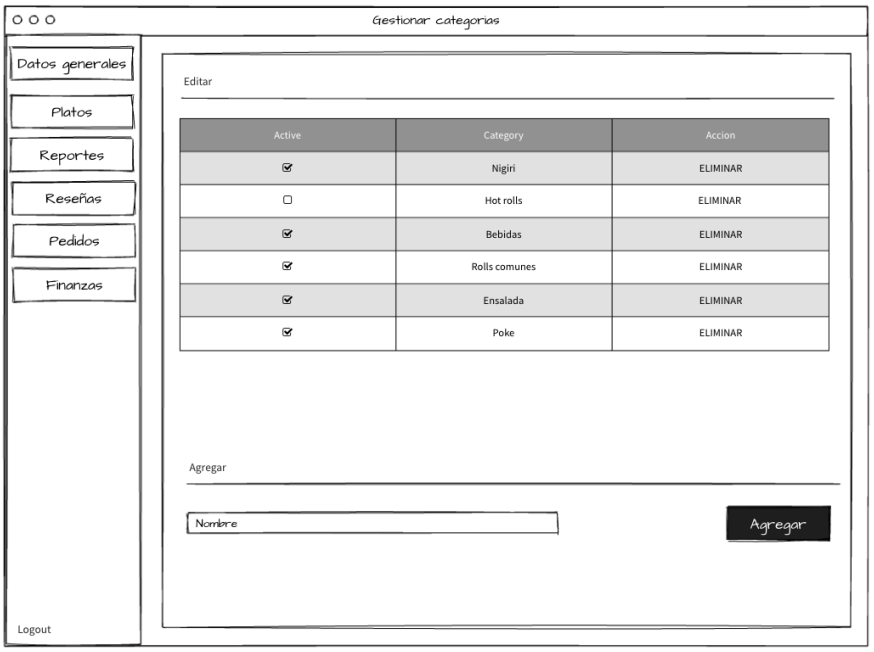
Sistema de Restaurantes



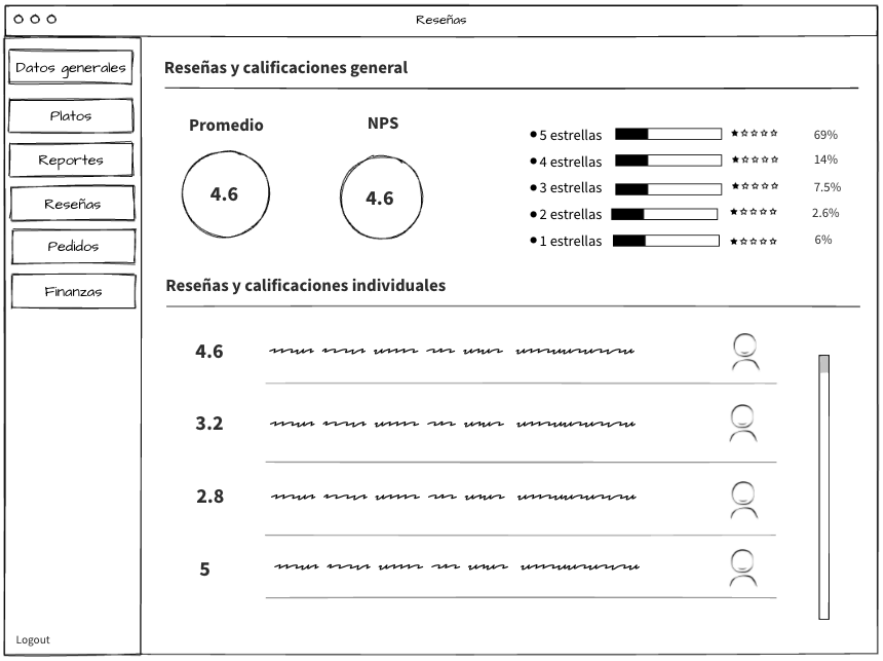
Sección Datos generales



Sección Manejo de platos



Sección Gestión de categorías

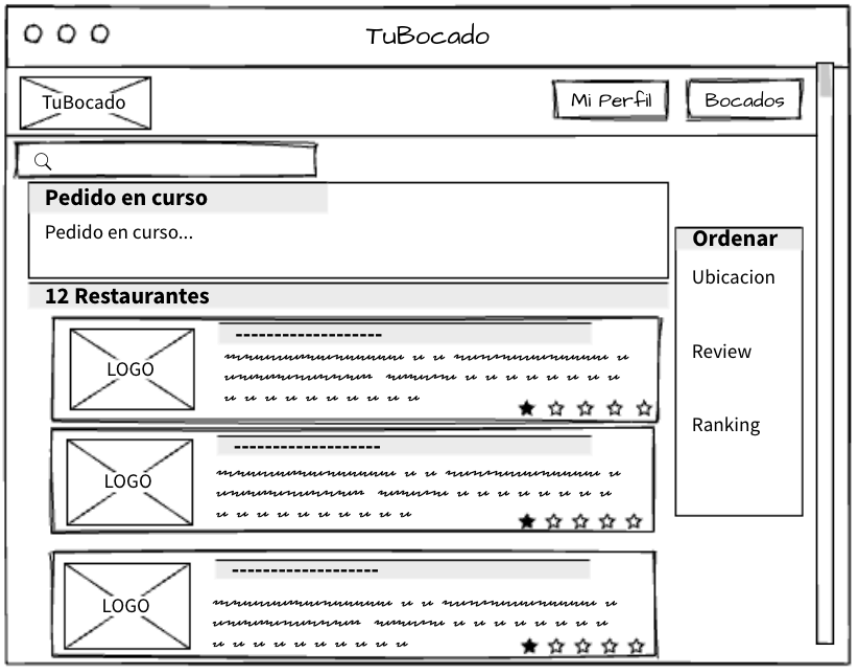


Sección Reseñas de restaurantes

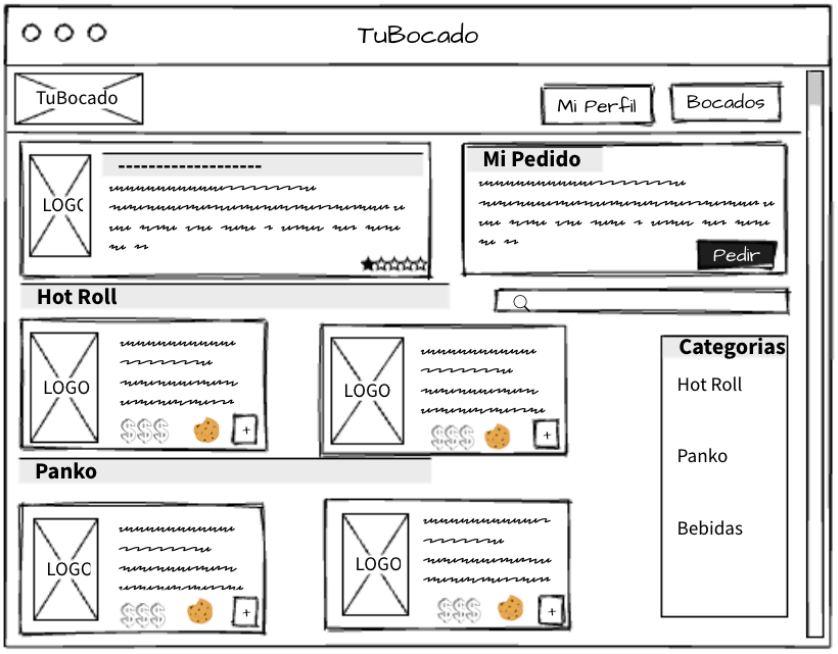


Sección Ver Pedidos

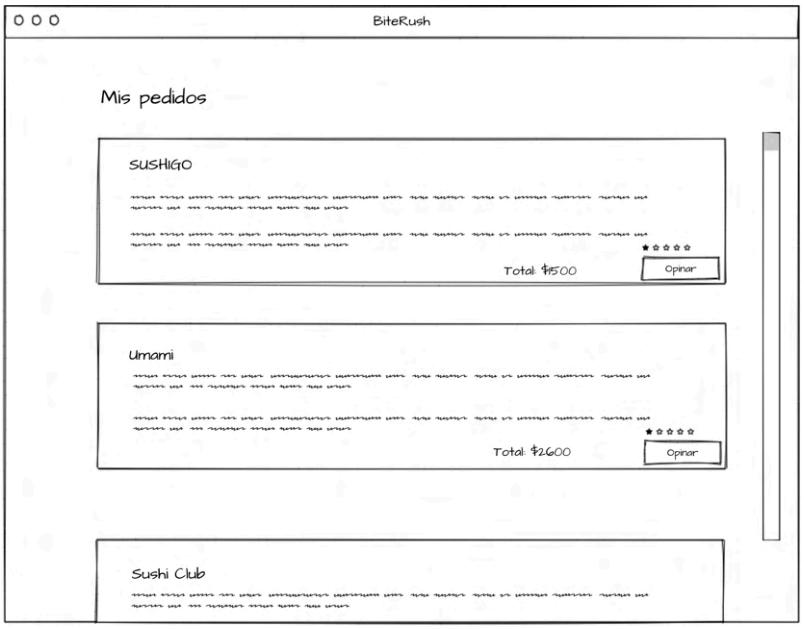
Sistema de Consumidores



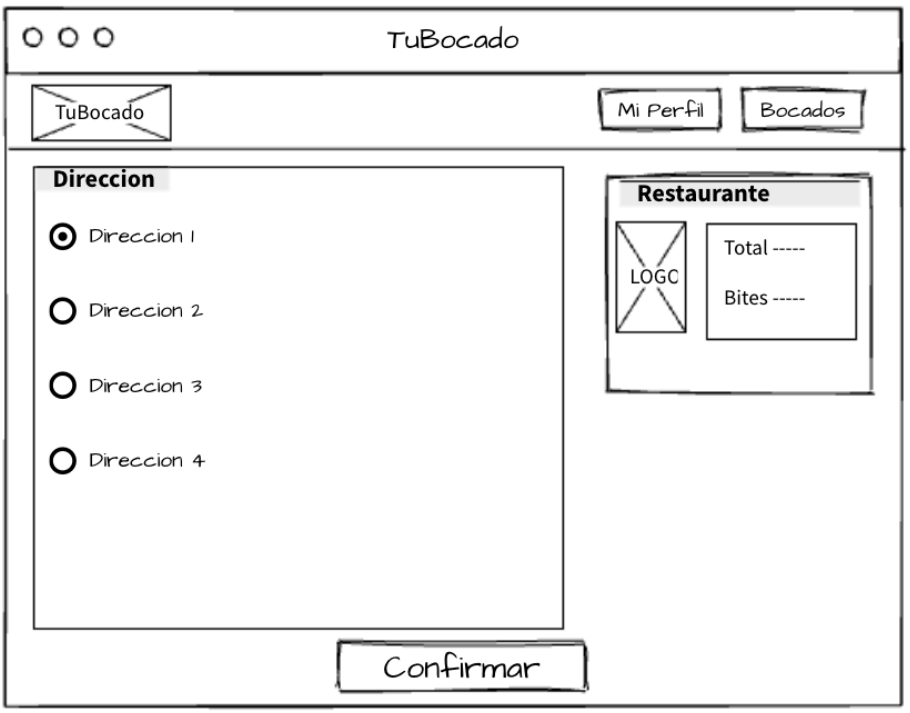
Sección Home



Sección Ver menú



Sección Ver mis pedidos

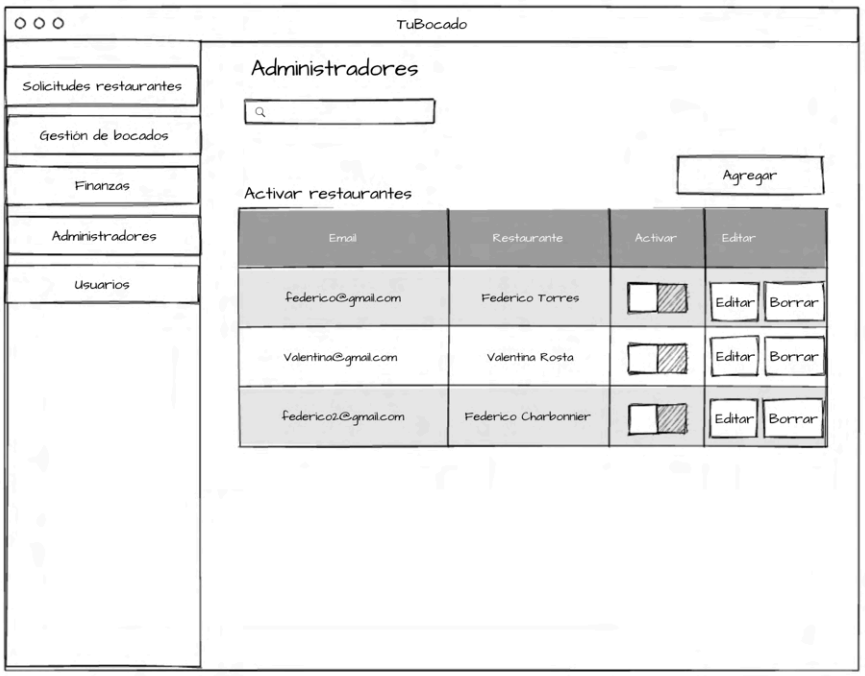


Sección Editar dirección de envío

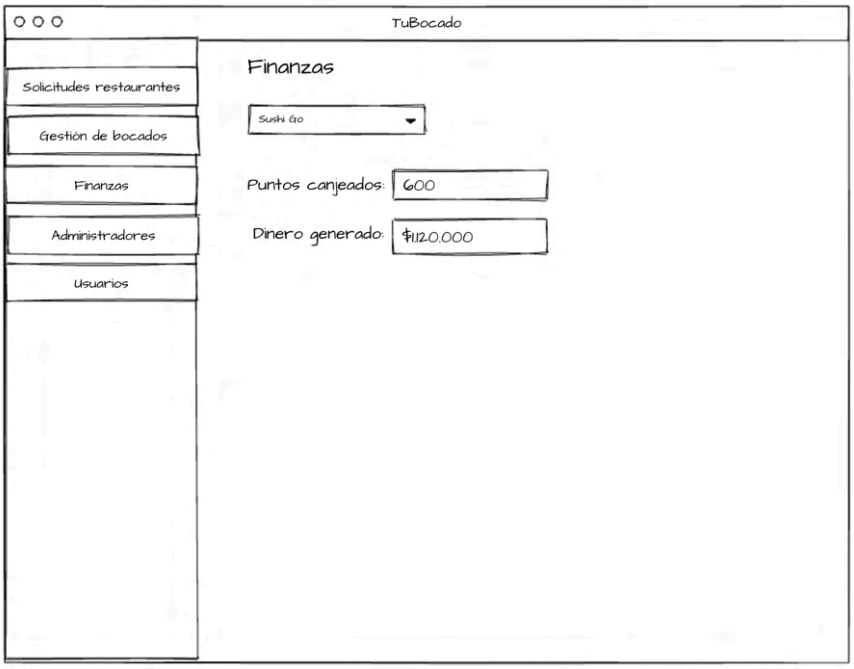


Sección Mi Cuenta

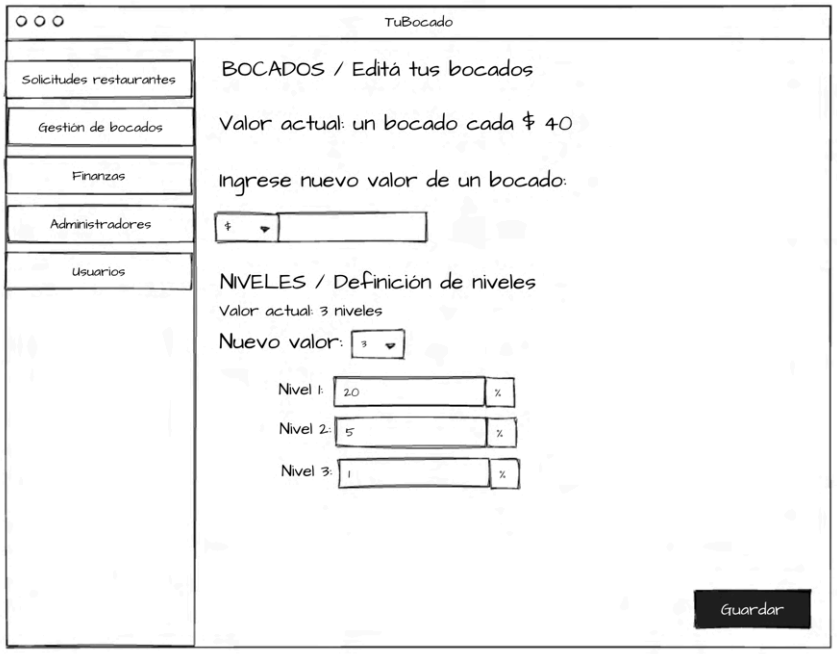
Sistema de Managers de TuBocado



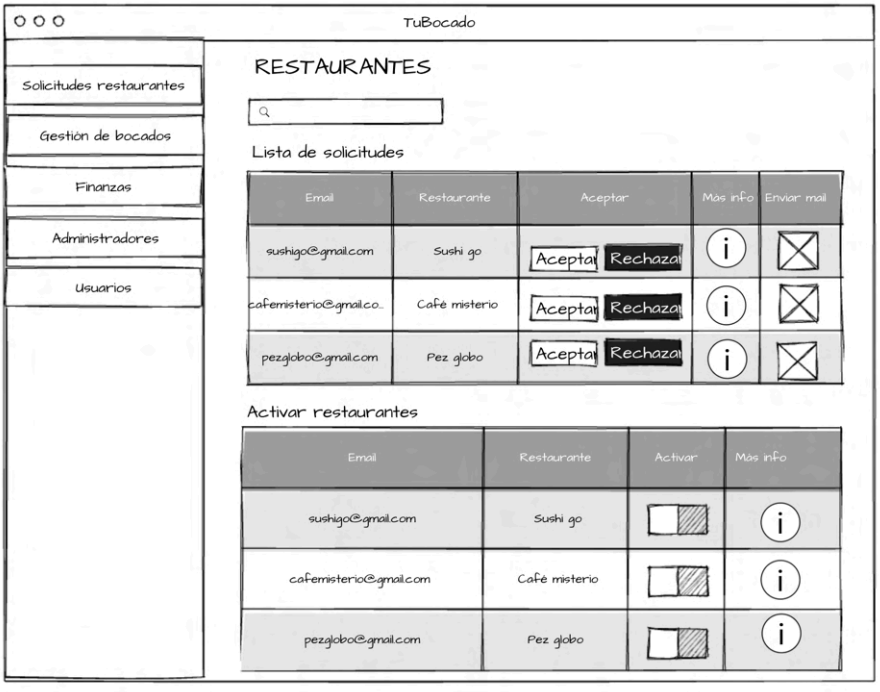
Sección Administradores



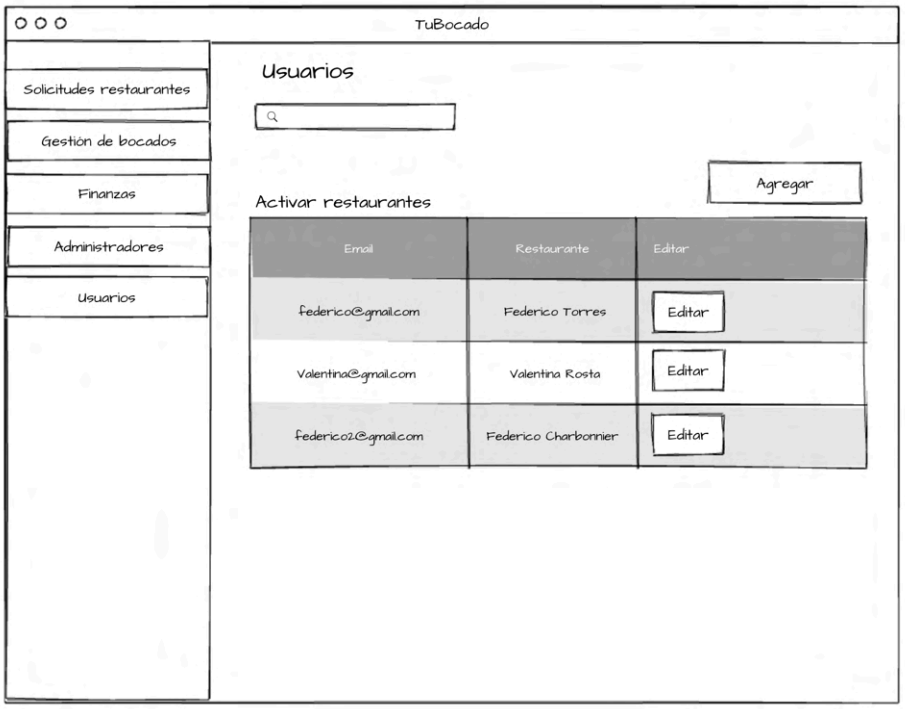
Sección Finanzas



Sección Gestión de Bocados



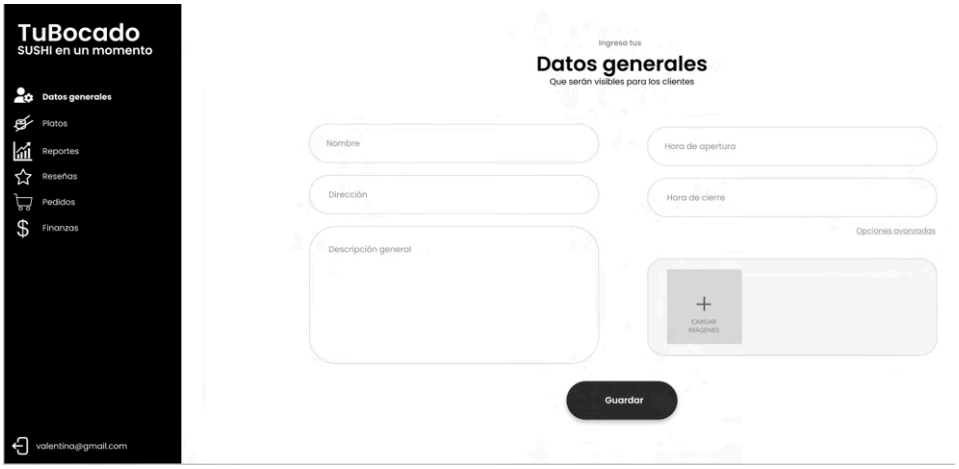
Sección Restaurantes



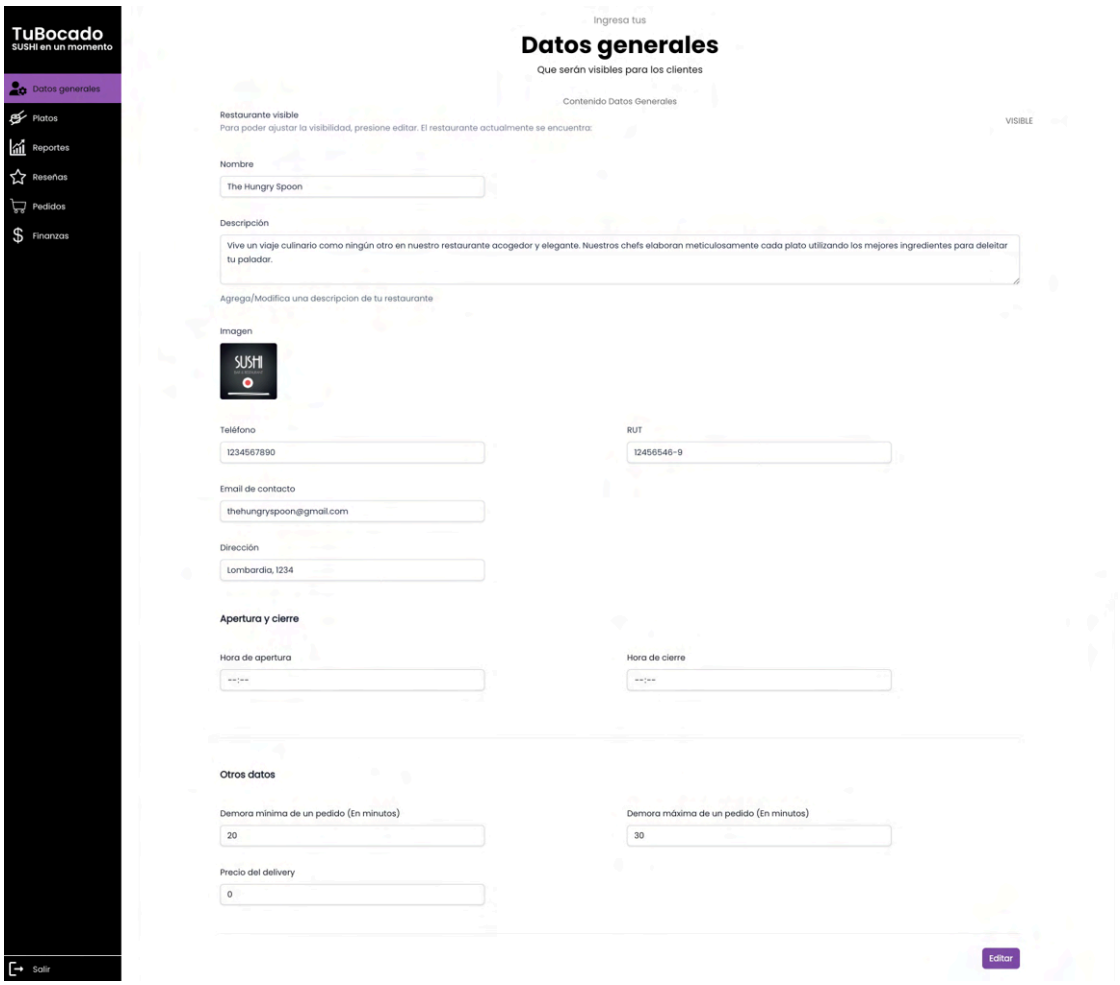
Sección Usuarios

A.13. Prototipo

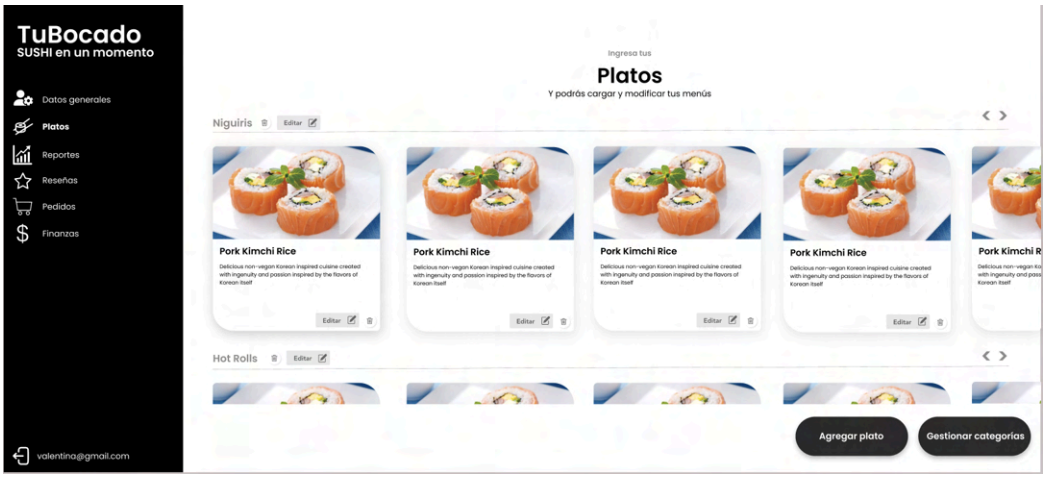
Sistema de Restaurantes



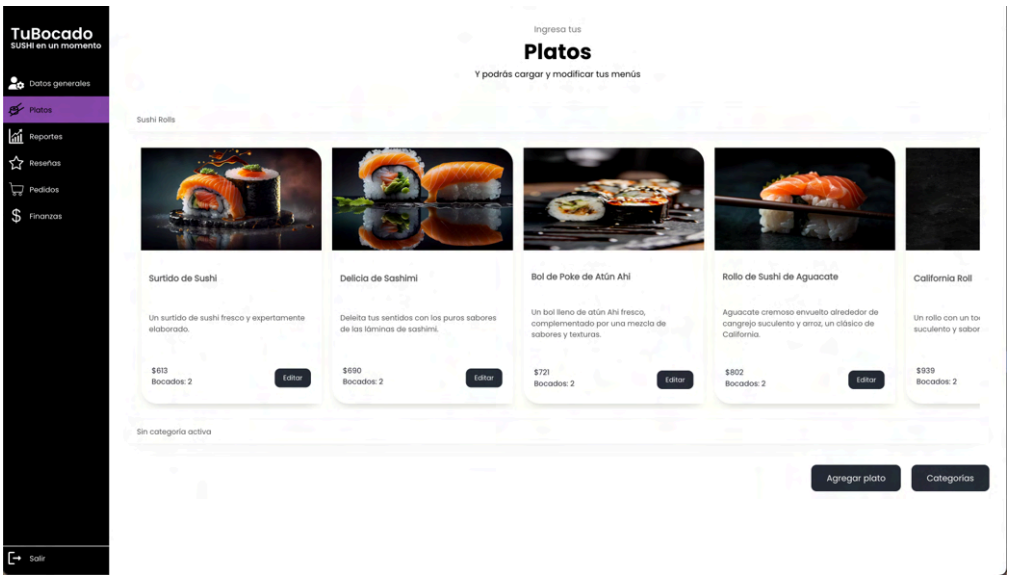
Sección Datos generales V1



Sección Datos generales V4



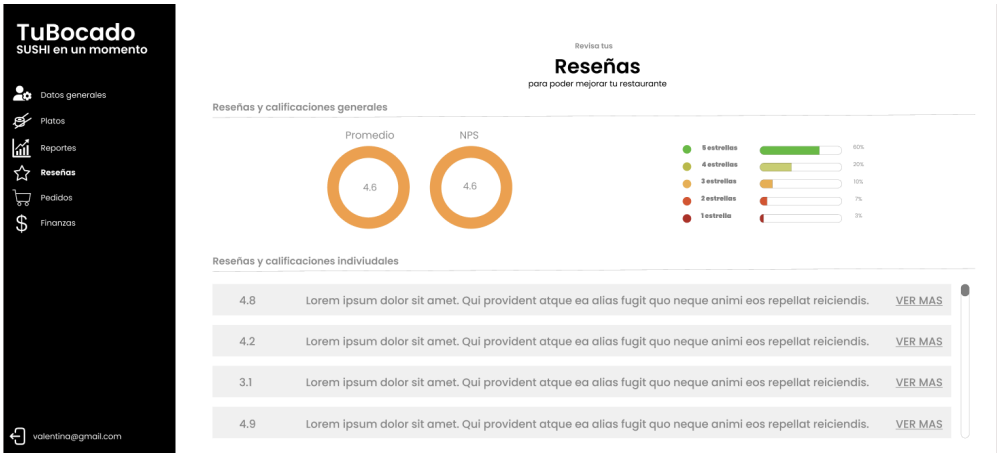
Sección Manejo de platos V1



Sección Manejo de platos V4



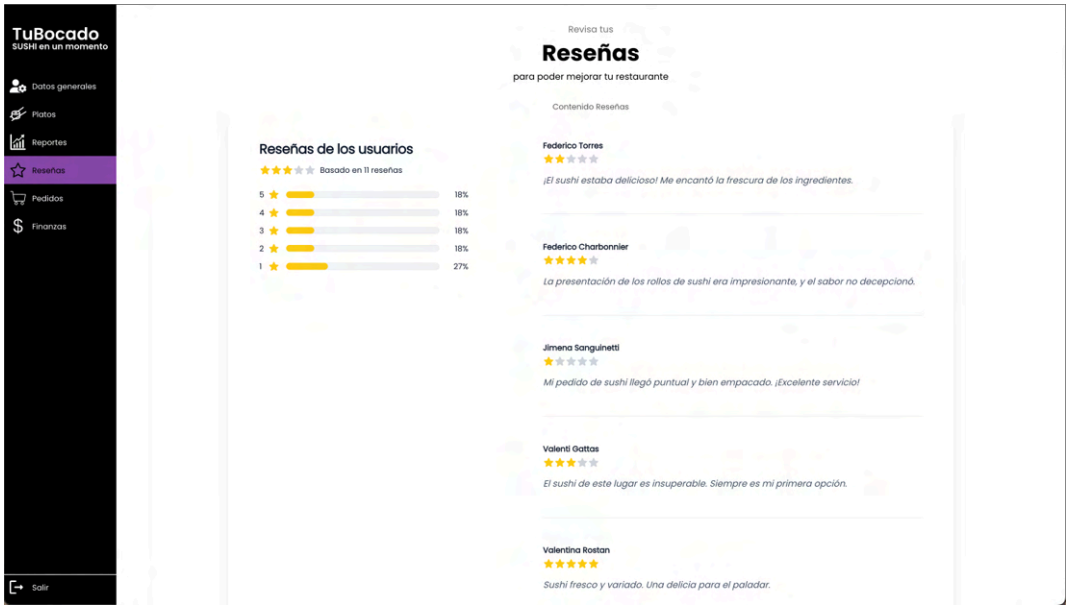
Sección Reseñas de restaurantes V1



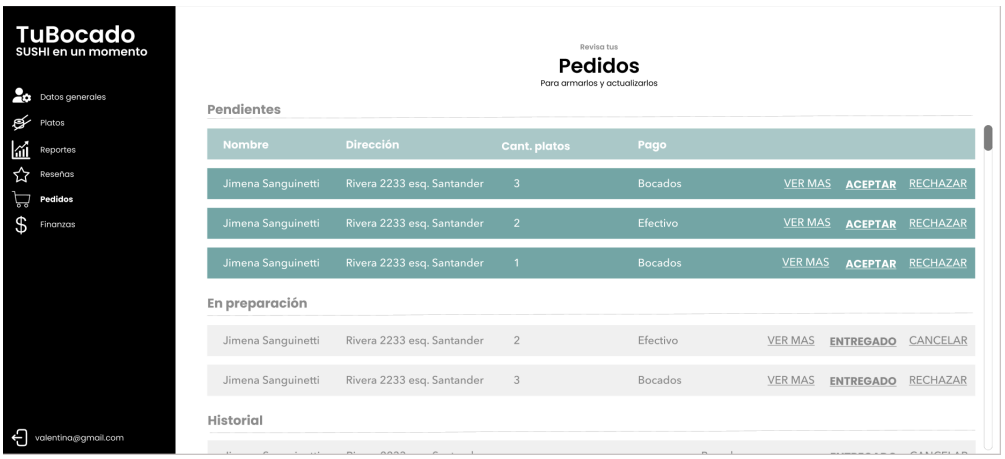
Sección Reseñas de restaurantes V2



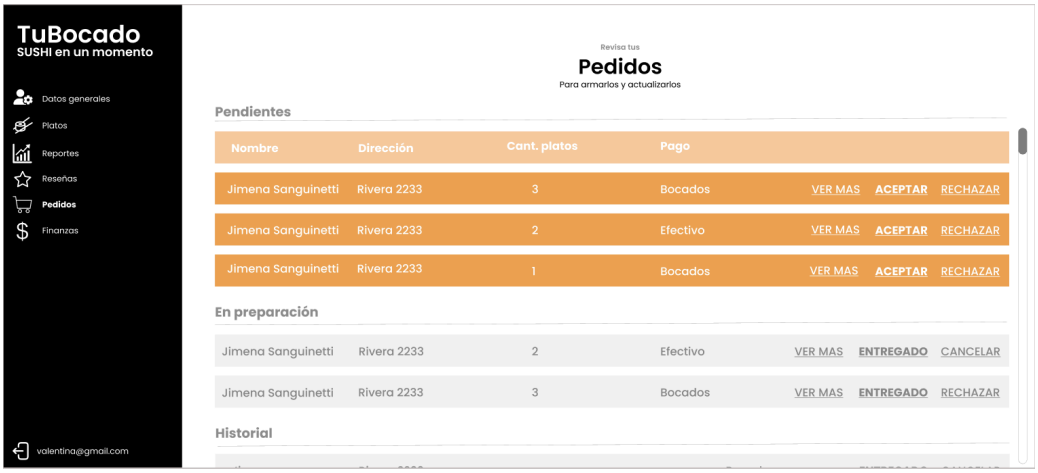
Sección Reseñas de restaurantes V3



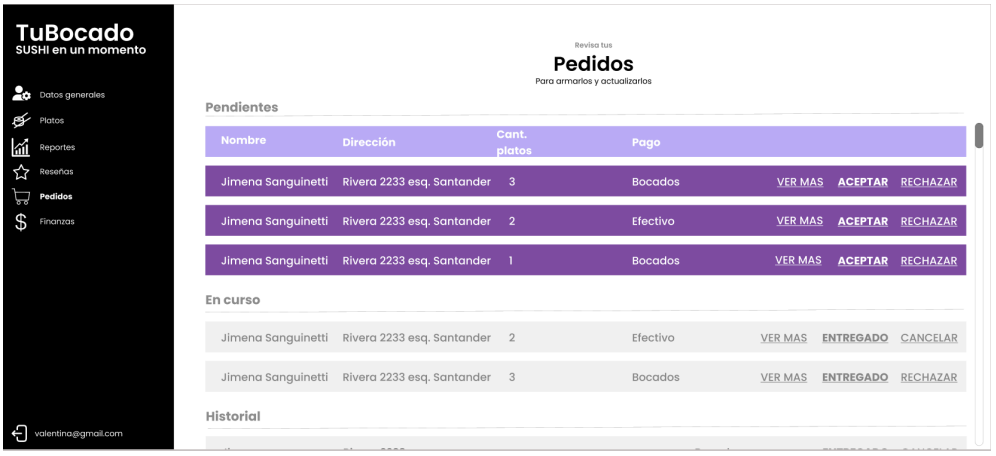
Sección Reseñas de restaurantes V4



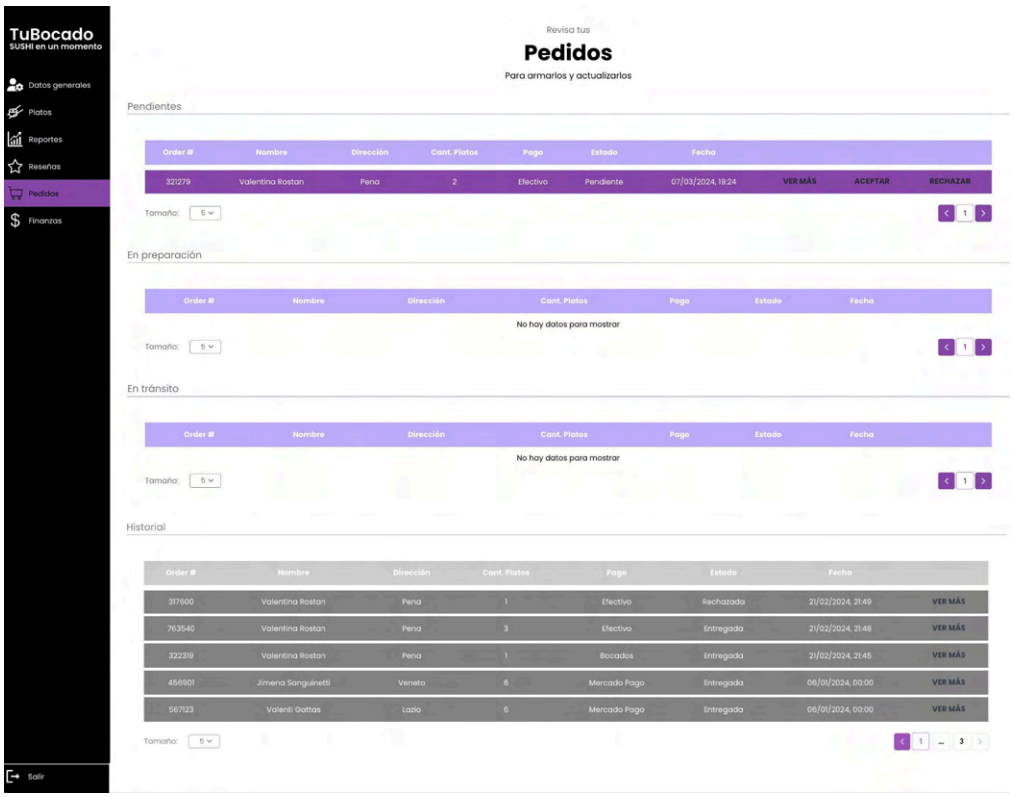
Sección Ver Pedidos V1



Sección Ver Pedidos V2

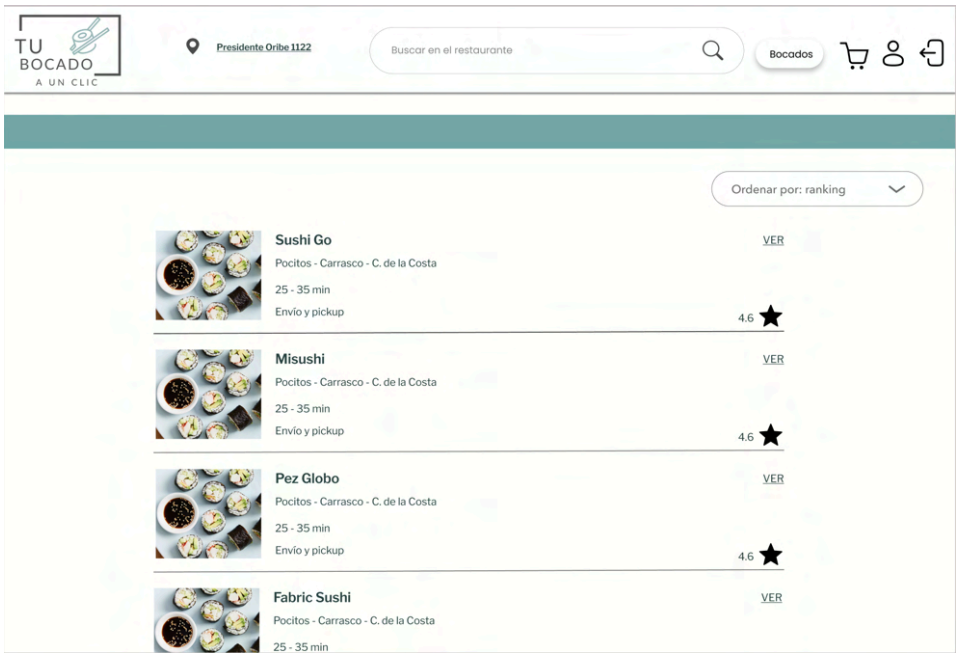


Sección Ver Pedidos V3

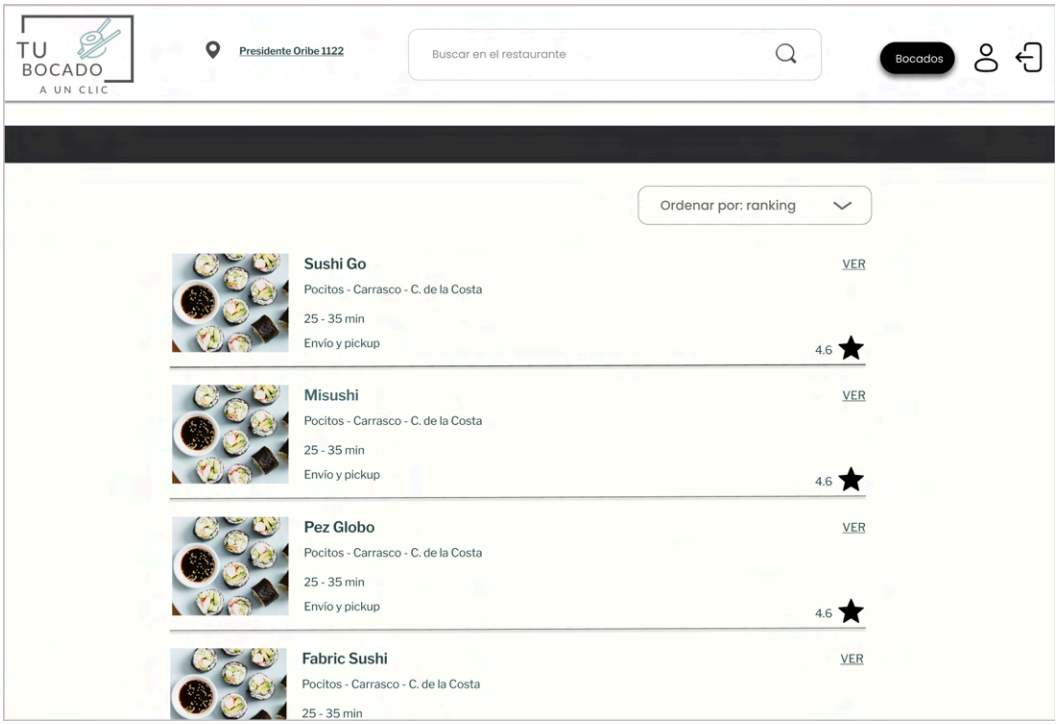


Sección Ver Pedidos V4

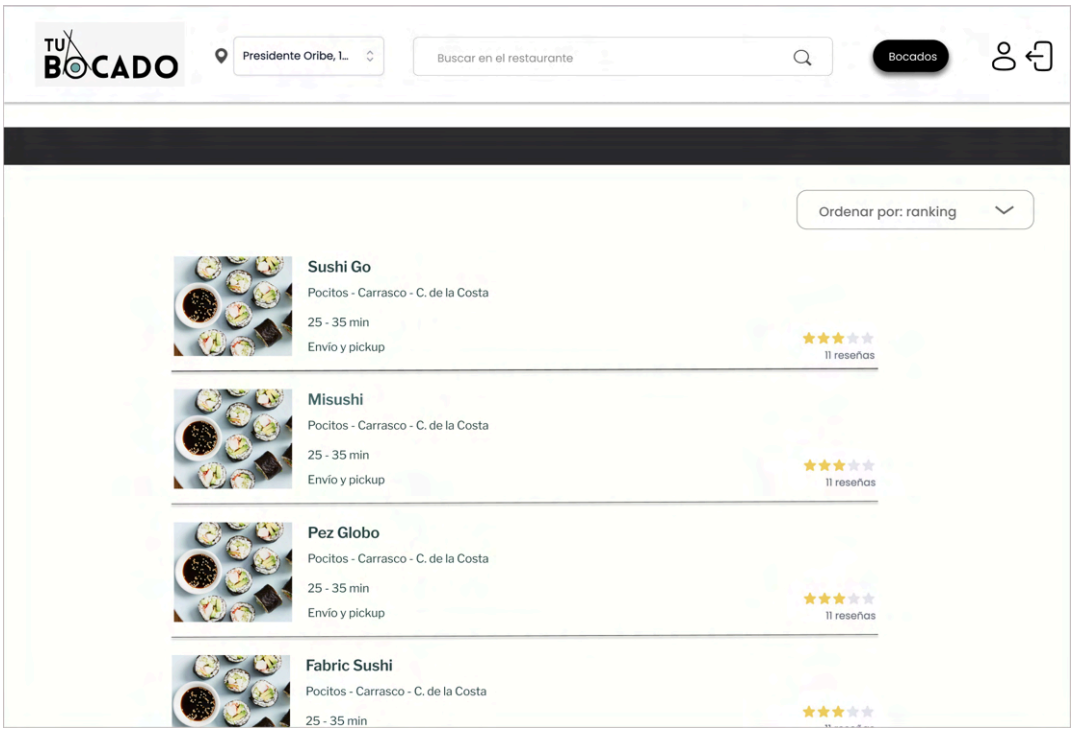
Sistema de Consumidores



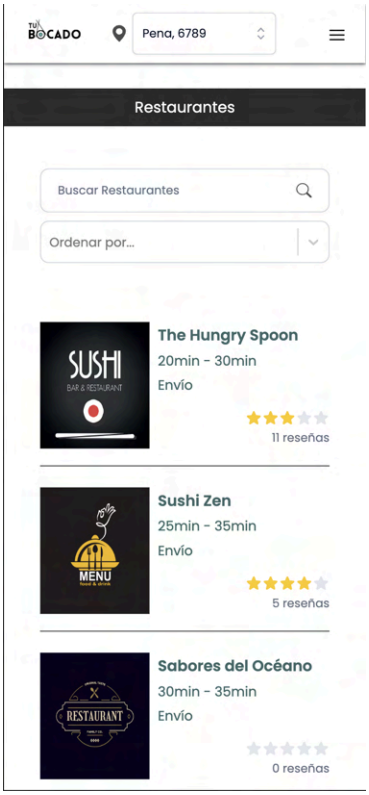
Sección Home V1



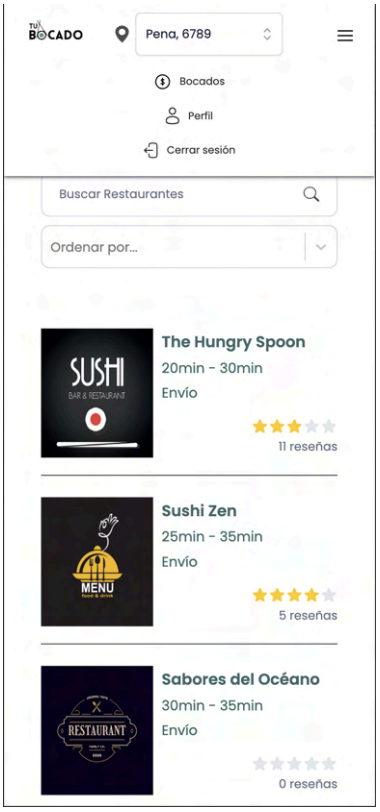
Sección Home V2



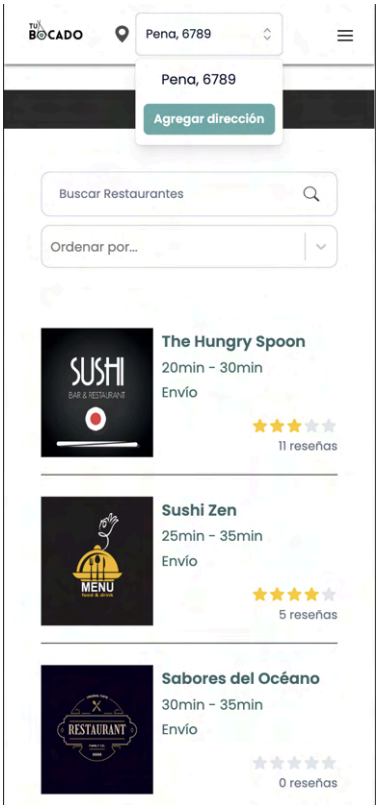
Sección Home V3



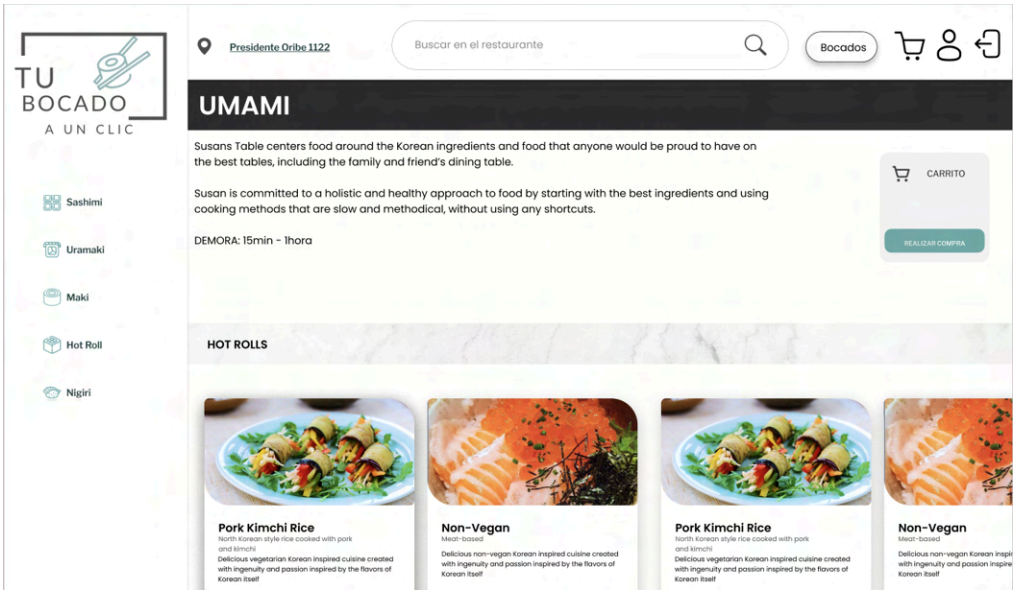
Sección Home Mobile V4



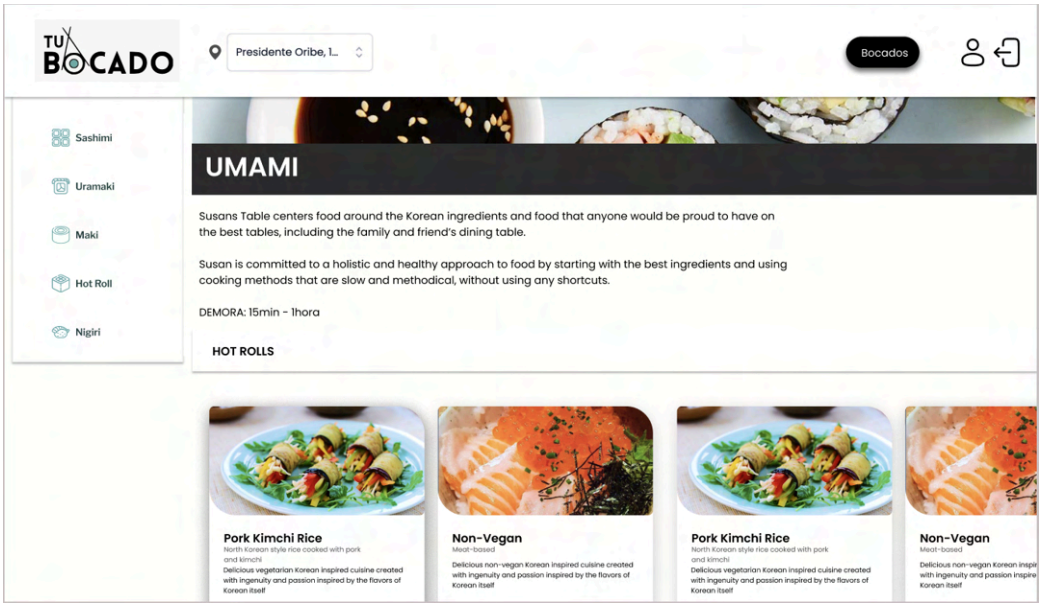
Header desplegado V4



Direcciones Mobile V4



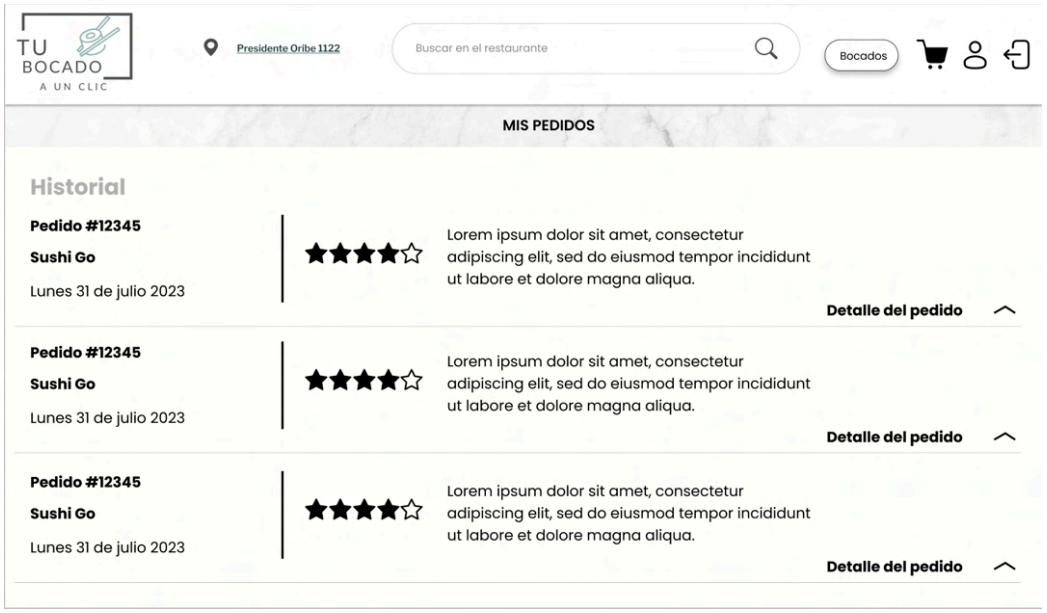
Sección Ver menú V1



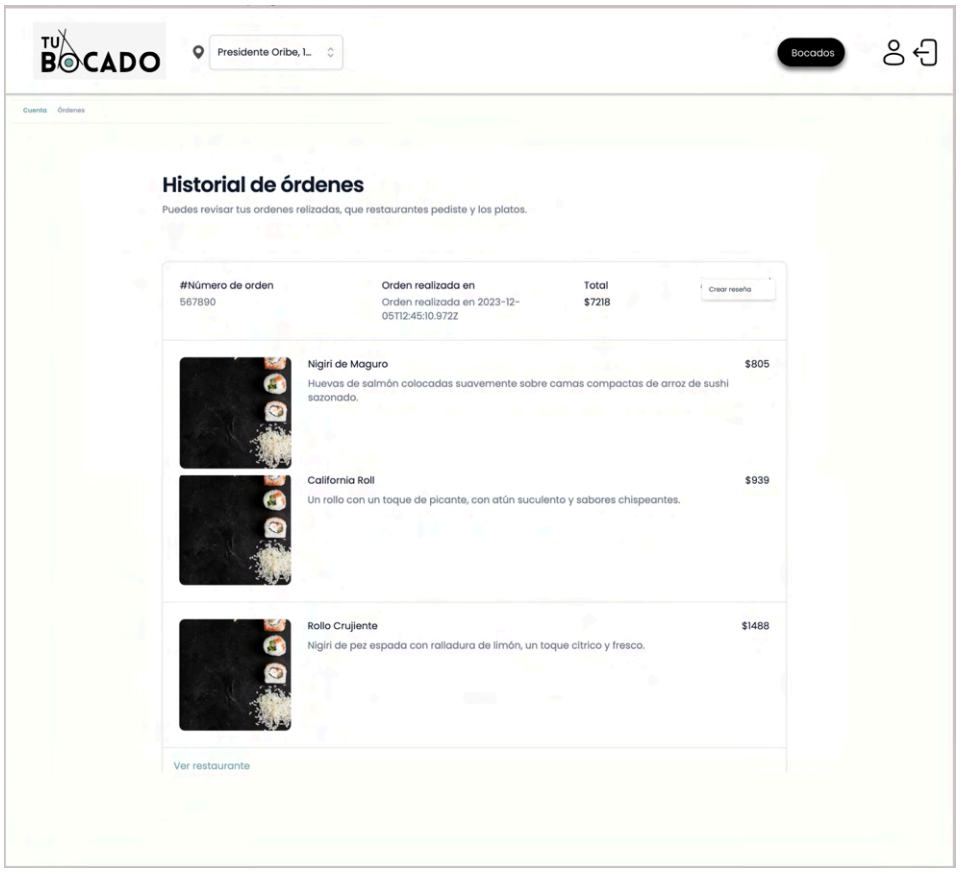
Sección Ver menú V3



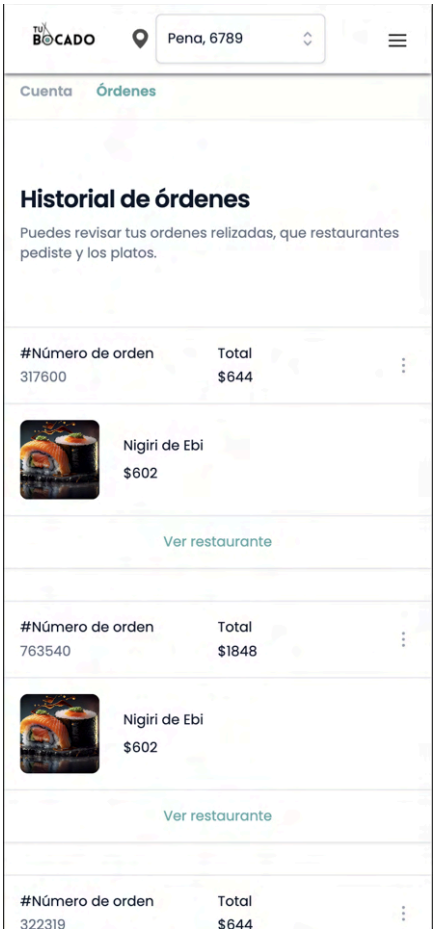
Sección Ver menú mobile V4



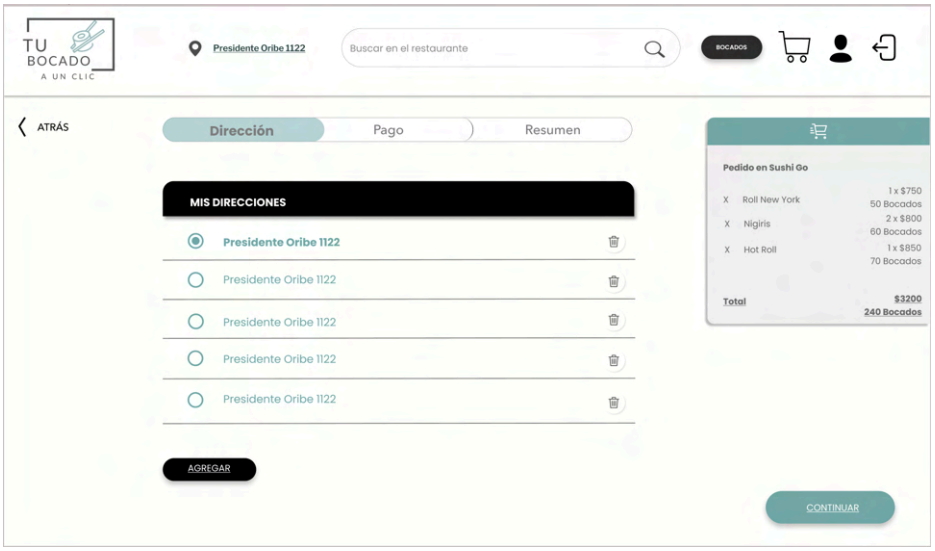
Sección Ver mis pedidos V1



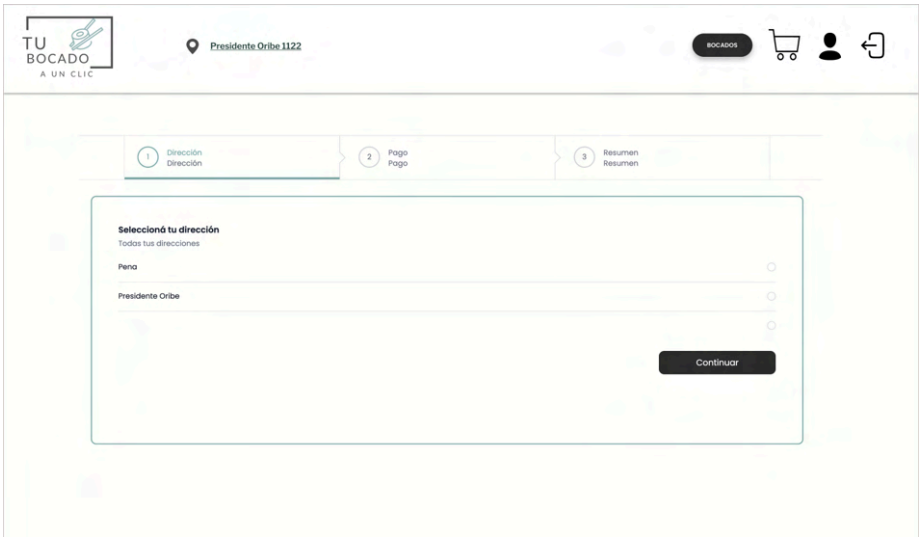
Sección Ver mis pedidos V3



Sección Ver mis pedidos mobile V4



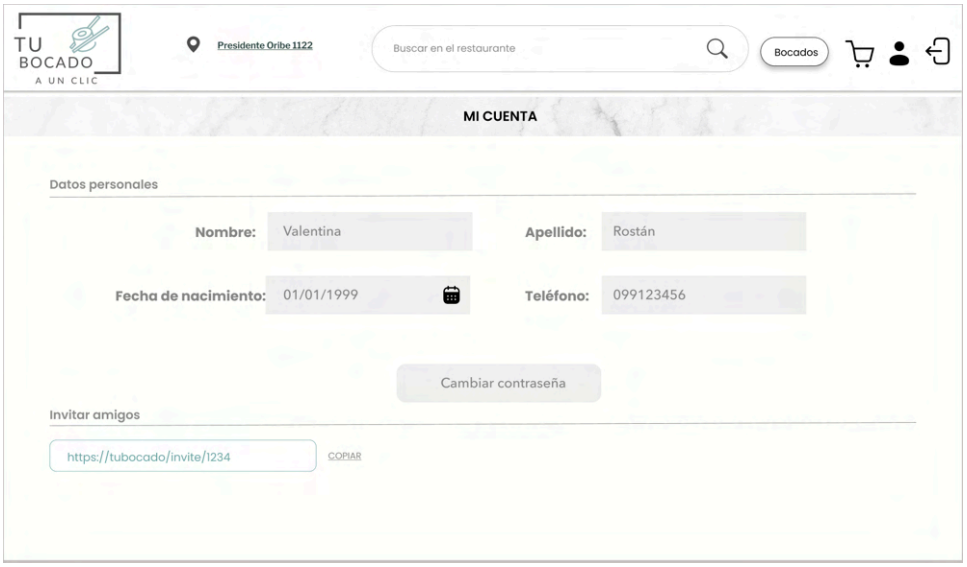
Sección Editar dirección de envío V1



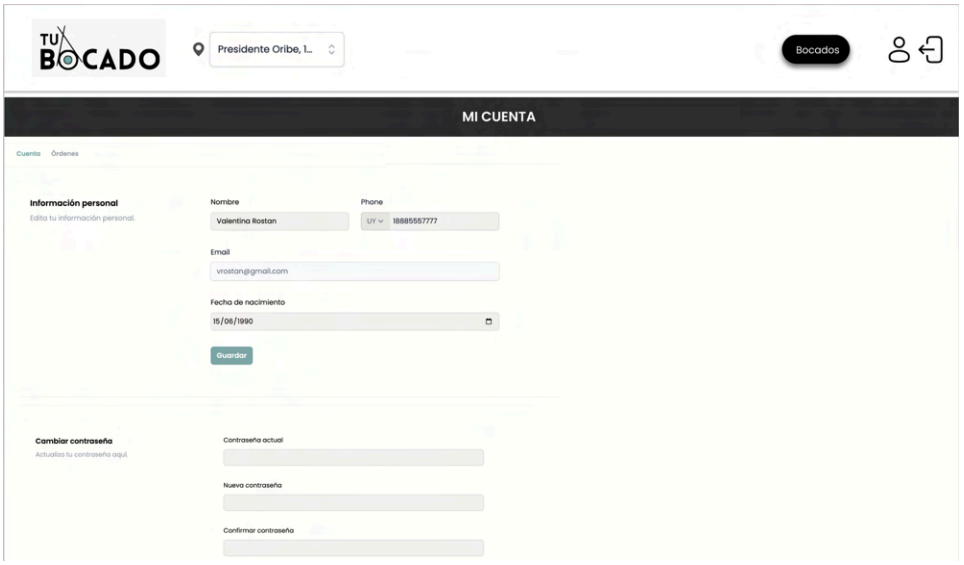
Sección Editar dirección de envío V2



Sección Editar dirección de envío mobile V4



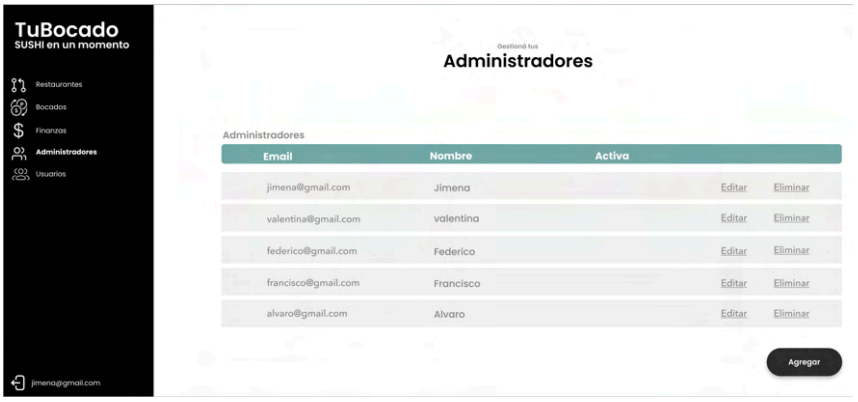
Sección Mi Cuenta V1



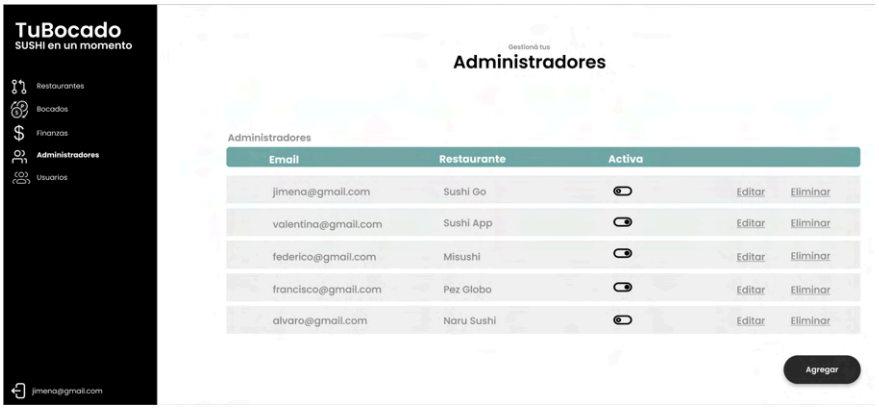
Sección Mi Cuenta V3



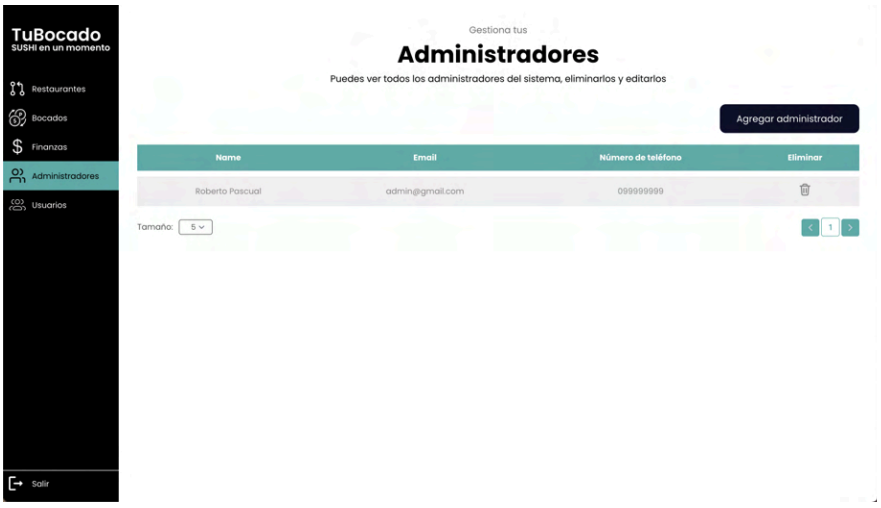
Sección Mi Cuenta mobile V4



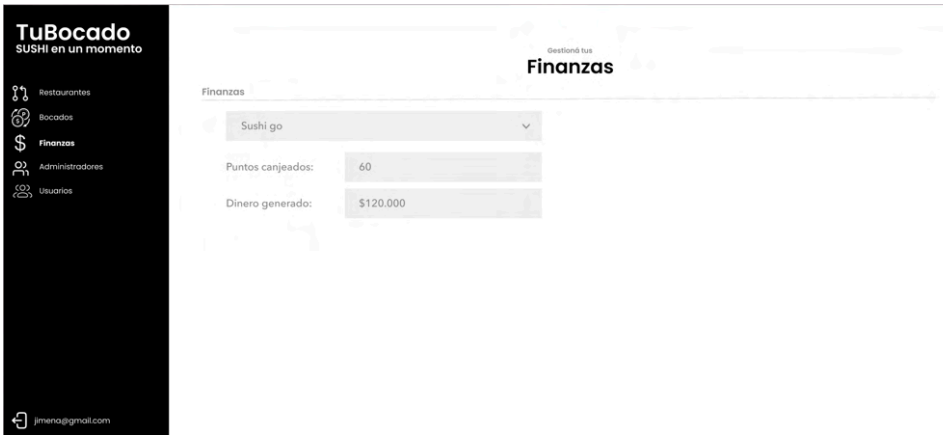
Sección Administradores V1



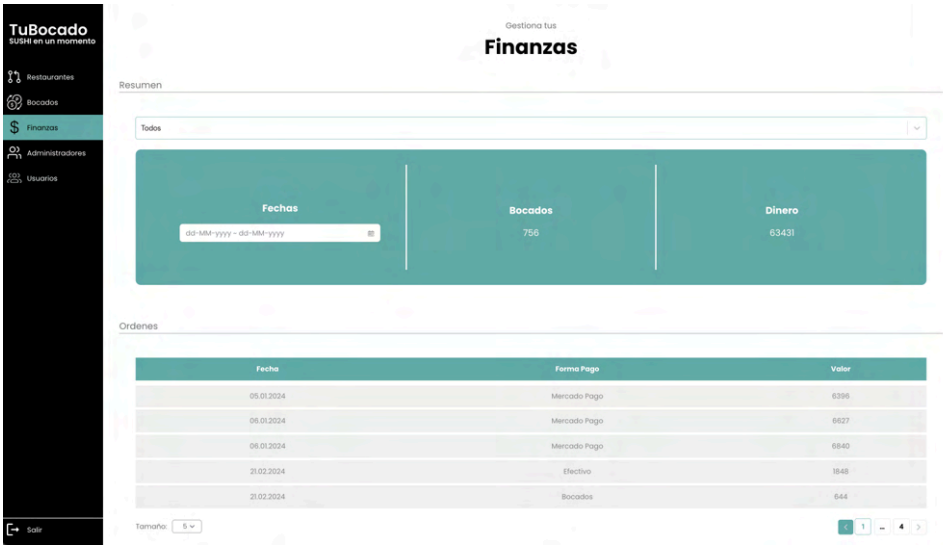
Sección Administradores V2



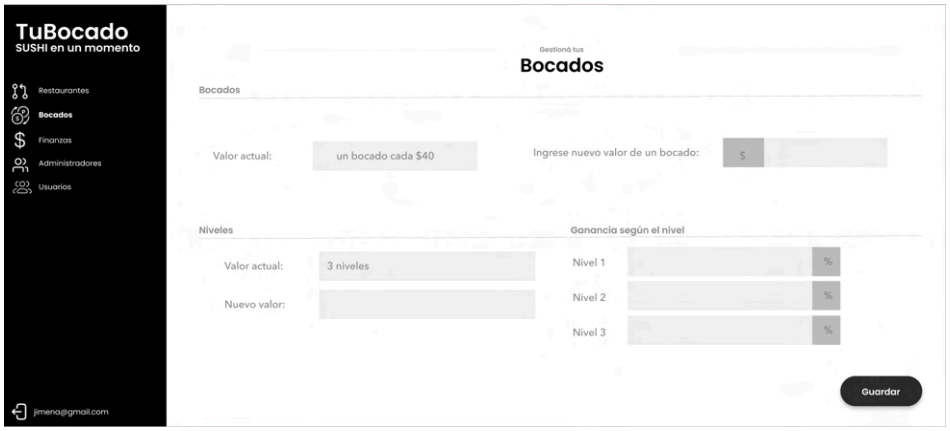
Sección Administradores V4



Sección Finanzas V1



Sección Finanzas V4



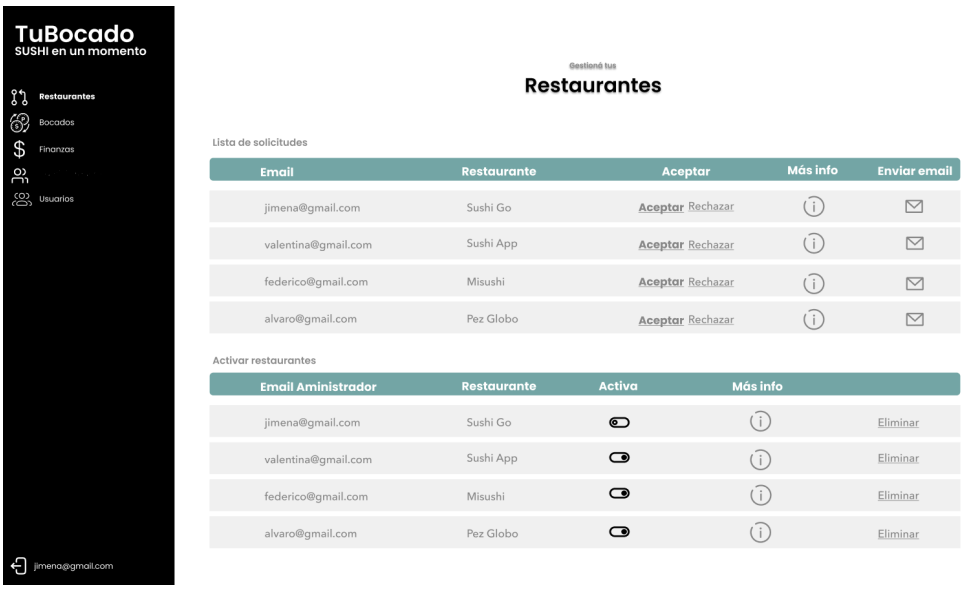
Sección Gestión de Bocados V1



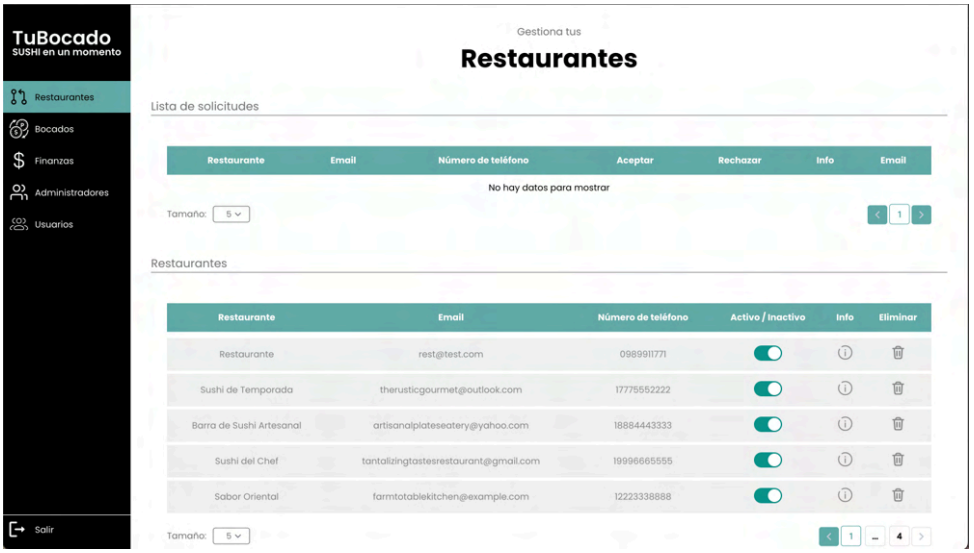
Sección Gestión de Bocados V2



Sección Gestión de Bocados V4



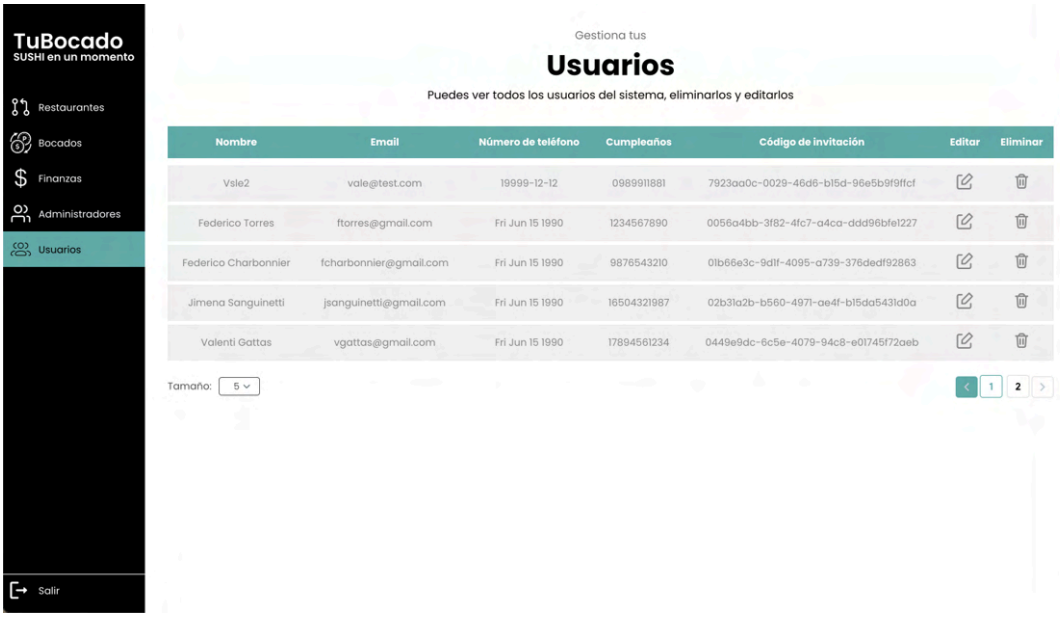
Sección Restaurantes V1



Sección Restaurantes V4



Sección Usuarios V1



Sección Usuarios V4

A.14. Validación con el cliente y pruebas con usuarios - pretotipo y prototipo

Pretotipo

Prueba realizada 14/06

Página administrados

Opción habilitar restaurantes: se debe cambiar por solamente poner rechazar y aceptar la primera vez y que luego una vez que se aceptó se debe poner un switch, no es aceptar y rechazar nada más. Además cuando pongo rechazar/aceptar se manda un mail y el asunto del rechazo en caso de ser necesario.

Solicitud de bocados: no se va a mostrar la pantalla de aceptar cotización bocado de cada restaurante. El restaurante lo va a hacer y siempre tiene que seguir la cotización base, solo hay excepciones en las que puede generar promociones pero lo hace el restaurante mismo en la página del restaurante. Además el restaurante puede cambiar el valor de la milla pero siempre tiene que cumplir el mínimo.

Parámetros del sistema: se quiere que esté todo junto y ahí van a estar las dos. Tiene que estar por un lado la cotización de pesos a millas y por otro lado la base de millas a pesos para todos los restaurantes. Además también se debe cambiar la de nivel para allí así se tienen todos los parámetros del sistema juntos. Falta un parámetro ya que parte de los parámetros es c**uantas millas se gana por nivel y eso falta.**

Requisito a tener en cuenta:

La cotización de millas tiene que estar asociada a una fecha, vale 10 hasta que lo cambio. NO TIENE QUE AFECTARME LO OTRO RETROACTIVO. Las millas varían con el tiempo entonces si veo un estado de cuenta antes tengo que ponerle el valor de la milla en ese momento no la que hay ahora porque puede cambiar. OJO con esto hay que manejar el historial de millas con fecha para las cuentas.

Tabla de tiempo para ambos lados para pesos → milla y para millas → pesos. Voy a tener una única globbal en ambas y los restaurantes pueden poner promos en los momentos y eso se van a guardar como transacción de tipo “promo” pero vamos a guardar una mínima, las promo siempre tiene que dar más pesos que la cotización actual.

Pagina restaurantes

En general bien. Se deben agregar reportes que en las estadísticas aparezcan las millas también. Además se debe agregar una pantalla promociones o dentro del menú para que el restaurante pueda poner sus promociones, estas tienen que ser únicamente con millas y el restaurante puede poner el valor de millas -> precio siempre y cuando respete la cotización básica.

Página usuarios

En general bien, se debe detallar mejor el proceso de cambio de millas ya que no funcionara mas canjeando millas con cupones. Las promos son por millas no por plata. IMPORTANTE.

Agregar pantalla de millas que vea cuántas millas tiene un usuario y allí poner el link para referenciar y ver los niveles; además se tiene que decir en detalle cuánto va ganando un usuario y cuánto va ganando por nivel de referencia. Es muy importante este dashboard, además por compartir también se darán millas pero eso es avanzado, no se necesita ya.

Feedback Prototipo Release 1

Aplicación web para consumidores

- Modificar el color del layout, cambiar el color negro utilizado por ver oscuro.
- En el registro de consumidor agregar el loco de TuBocado.
- En el login de consumidor alinear los inputs con el botón de inicio de sesión.
- En el login de consumidor, en el campo de calendario agregar un símbolo de un calendario para que aparezca un display para seleccionar la fecha.

- En el login de consumidor, aumentar el ancho del botón “login”, mismo largo que los campos.
- Remover el carrito del header.
- En la vista de home, modificar el color de la barra superior a negro.
- En la vista de home, cambiar el estilo de filtro.
- En la vista de pedidos, simplificar la vista. Separar la información del pedido con el header, para que sea más agradable visualmente.
- Dentro de la vista ordenes, cambiar de lugar el carrito de compras. Ponerlo en la parte superior a la derecha, junto a los pedidos.
- En la vista de órdenes, cuando se realiza una compra exitosamente, que aparezca un mensaje notificando al usuario.

Aplicación web para dueños de restaurante

- En las vistas de login y registro, agregar un campo para que el usuario ingrese el rut de su restaurante.

Feedback Prototipo Release 2

Aplicación web para consumidores

- En la vista de login y registro, remover el logo de la aplicación.
- En la vista de login y registro, en el campo de la contraseña agregar el símbolo de un “ojo”, para que el usuario pueda ver su contraseña si lo desea.
- En la vista de registro agregar un botón para volver al login.
- En la vista de login, agregar un botón de “me olvide la contraseña”.
- En la vista home, remover la opción de “ver”.
- En la vista home, actualizar el logo al nuevo
- En la vista home, modificar el diseño de las direcciones. Mostrar la calle y número únicamente.
- En la vista de pedidos modificar la información que se ve del mismo. Remover la hora que se realizó el pedido, mostrar solamente la fecha.
- En la vista de perfil, separar más los campos de información del consumidor y aumentar el tamaño de la letra.
- En la vista agregar el historial de órdenes del consumidor.

Aplicación web para dueños de restaurante

- Cambiar el color del layout a violeta. Utilizar distintas tonalidades del color para toda la plataforma.
- En las vistas de login y registro, agregar un símbolo de un ojo en el campo de la contraseña. Así el usuario tiene la opción de verla.
- En la vista de registro, agregar un botón que redirige al login.
- En la vista de platos y categorías, agregar la opción para poder editar una categoría.
- En la vista de platos y categorías, cambiar la UI de los platos para un diseño más moderno.
- En la vista de platos y categorías, cambiar los estilos de los botones de editar y eliminar categorías. Utilizar el mismo estilo que los botones del la vista login.
- En la vista de platos y categorías, en la ventana de editar un plato cambiar el diseño a una vista más moderna.
- En la vista de órdenes, modificar los botones de detalle del pedido para que coincida con el resto de la plataforma.
- En la vista de órdenes, agregar los campos: dirección, estado de pedido y Bocados.
- En la vista de finanzas, remover la sección de contrato.
- En la vista de finanzas, agregar paginado para que sea más agradable visualmente.

Aplicación web para manager de TuBocado

- En la vista de login y registro, agregar un símbolo de un ojo al campo de la contraseña. Así el usuario tiene la opción de ver su contraseña si lo desea.
- En la vista de login, agregar la opción de “me olvide la contraseña”.
- En la vista de managers, modificar para que no se pueda editar los datos de los managers, solo borrar y leer.
- En la vista de managers, dentro de donde se muestra la información de los restaurantes. Cambiar el diseño a una vista más moderna.

Feedback Prototipo Release 3

Aplicación web para consumidores

- En las vistas de registro y login, que los botones tengan el mismo radius en los border y utilizar el mismo efecto de sombra para todos los botones. Utilizar como guía el estilo del botón de registrarse.
- En la vista de registro, que el ancho del botón de registrarse sea del mismo ancho que los campos.
- En la vista de pedidos, cambiar donde se muestran las reseñas de cada restaurante. Mostrarlas a la derecha de cada restaurante.
- En la vista de pedidos, agregar la UI del historial de pedidos al lado de la opción para ver el perfil.
- En la vista de pedidos, agregar la opción para que un consumidor pueda agregar una reseña desde el historial de sus pedidos.
- En la vista de órdenes, si la misma es pickup mostrar esa información, a la derecha de donde se muestra la dirección.

Aplicación web para dueños de restaurante

- En la vista de registro, centrar los campos de los datos con el botón de registrarse.
- En la vista de login y registro, modificar las sombras y el radius de los botones para que se igual a los botones de la aplicación web de los consumidores.

Aplicación web para manager de TuBocado

- En la vista de finanzas, modificar el estilo de la misma. Que tenga el mismo estilo que la vista de managers.

Feedback Prototipo Release 4

Aplicación web para consumidores

- En el login, agregar la opción para recordar al consumidor. Asi no tiene que ingresar sus datos cada vez.

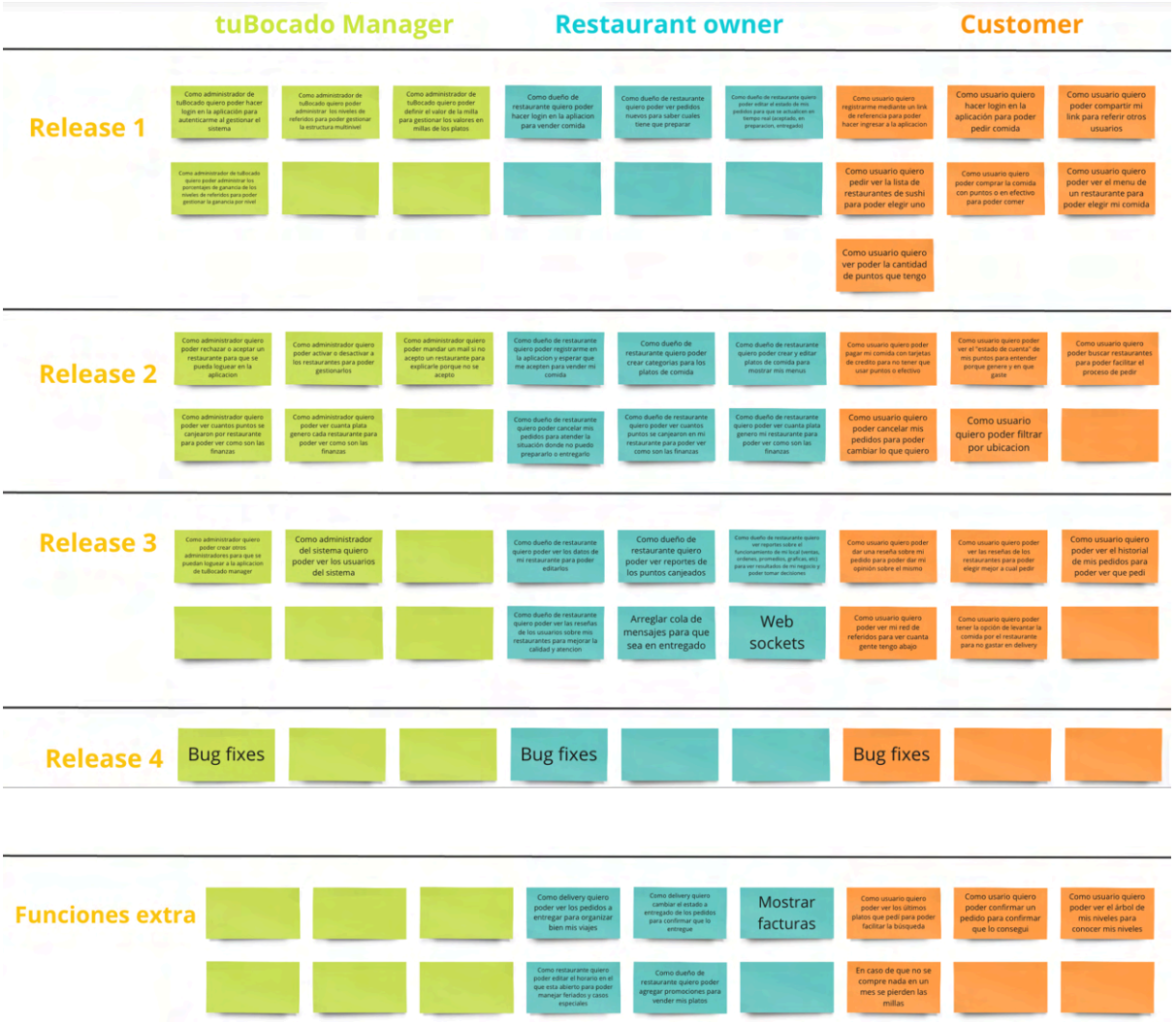
Aplicación web para dueños de restaurante

- En la vista de pedidos, implementar colores a las órdenes dependiendo de los estados que tienen. Por ejemplo si la orden está en pendiente mostrarla en amarillo. Si tiene estado entregado en verde, si fue cancelada en rojo.
- Agregar la posibilidad de manejar más de un restaurante con una única cuenta.

Aplicación web para manager de TuBocado

- En la vista de informes agregar, un historial de cómo varía el valor de los Bocados, con respecto al tiempo.

A.15. Miró completo



A.16. Product Backlog completo

Requerimientos Funcionales

Sprint 1

Títulо	**Register Customer**
Narrativa	**As a** customer**,** **I want to** be able to register through a specialized link **So that** I can easily create an account and start using the service
Criterios de Aceptación	**Given** I am a prospective user without an account, **When** I receive a specialized registration link, **Then** clicking on the link should direct me to a registration page. **Given** I have clicked on the specialized registration link, **When** the registration page loads, **Then** I should see fields to input my registration details such as username, email, and password. **Given** I am on the registration page accessed via the specialized link, **When** I fill in my details correctly and click the "Register" button, **Then** I should receive a confirmation message and be redirected to a welcome or dashboard page. **Given** I am on the registration page accessed via the specialized link, **When** I input incomplete or incorrect details and try to register, **Then** I should see an error message indicating that the registration was unsuccessful
Story Points	5

Títulо	**TuBocado manager Login**
Narrativa	**As a** TuBocado manager, **I want to** be able to login to the manager site **So that** I can securely access management functionalities and content.
Criterios de Aceptación	**Given** that I am a registered TuBocado manager with valid credentials, **When** I enter my valid username/email and password and press "ingresar", **Then** the system redirects me to the restaurant's management page. **Given** that I am a registered TuBocado administrator, **When** I enter an invalid username/email and a valid password, and press "ingresar", **Then** the system displays an error message indicating incorrect credentials.

	Given that I am a registered TuBocado administrator, **When** I enter my valid username/email and an incorrect password, and press "ingresar", **Then** the system displays an error message indicating incorrect credentials. **Given** that I am a registered TuBocado administrator, **When** I enter an invalid username/email and an invalid password, and press "ingresar", **Then** the system displays an error message indicating incorrect credentials. **Given** that I am on the admin login page, **When** I successfully log in and access the admin dashboard, **Then** my session should be maintained securely and authenticated throughout my time in the admin application. **Given** that I access the admin login page, **When** I attempt to log in from different devices and browsers, **Then** the login page should be responsive and function correctly across these platforms
Story Points	5

Tíitulo	**Customer Login**
Narrativa	**As a** customer, **I want to** be able to login **So that** I can order sushi and access personalized features and content.
Criterios de Aceptación	**Given** routes are developed, **When** creating routes, **Then** an API endpoint should be implemented to handle user login requests. **Given** a user submits their credentials via the front-end, **When** the back-end receives this data at the login API endpoint, **Then** it should validate the user's credentials against the stored data. **Given** a user's credentials are validated successfully, **When** the back-end confirms the credentials, **Then** it should generate an authentication token and send it back to the client. **Given** a user submits incorrect credentials, **When** the back-end validates and finds them incorrect, **Then** it should return an error message indicating that the login was unsuccessful. **Given** the back-end is processing login requests, **When** receiving and transmitting data, **Then** it should be done over a secure protocol (e.g., HTTPS).
Story Points	5

Títulu	**Restaurants owner Navigate between pages**
Narrativa	**As a** restaurant owner, **I want to** be able to navigate between pages using a sidebar **So that** I can access different sections of the application efficiently
Criterios de Aceptación	**Given** that I am logged into the application as a restaurant admin, **When** I open the web app, **Then** I should see a sidebar on the left-hand side with options (datos generales, platos, reportes, reseñas, pedidos, finanzas) to navigate to different sections of the application. **Given** that I am on the landing page of the application, **When** I click on a sidebar link corresponding to a specific section (e.g. platos), **Then** the application should smoothly and responsively navigate to the selected section's page, showing relevant content. **Given** that I am on a specific section's page (e.g. platos), **When** I navigate to that section using the sidebar link, **Then** the sidebar link for that section should be visually highlighted or marked as active to indicate my current location within the app. **Given** that I am using a device with a smaller screen (e.g., tablet or mobile), **When** I access the application, **Then** the sidebar should have the option to be collapsed, allowing me to expand or collapse it as needed to maximize screen space.
Story Points	3

Títulu	**TuBocado manager Navigate between pages**
Narrativa	**As a** TuBocado manager, **I want to** navigate between pages using a sidebar **So that** I can access different sections of the application efficiently
Criterios de Aceptación	**Given** that I am logged into the application as a manager, **When** I open the web app, **Then** I should see a sidebar on the left-hand side with options (restaurantes, Bocados, finanzas, administradores, usuarios) to navigate to different sections of the application. **Given** that I am on the landing page of the application, **When** I click on a sidebar link corresponding to a specific section (e.g. Restaurantes), **Then** the application should smoothly and responsively navigate to the selected section's page, showing relevant content. **Given** that I am on a specific section's page (e.g. Restaurantes), **When** I navigate to that section using the sidebar link, **Then** the sidebar link for that section should be visually highlighted or marked as active to indicate my current location within the app. **Given** that I am using a device with a smaller screen (e.g., tablet or

	mobile), **When** I access the application, **Then** the sidebar should have the option to be collapsed, allowing me to expand or collapse it as needed to maximize screen space.
Story Points	5

Tíitulo	**View menu**
Narrativa	**As a** customer, **I want to** be able to view the menu of each restaurant **So that** I can choose a dish to order.
Criterios de Aceptación	**Given** that I am viewing the details of a specific restaurant, **When** I navigate to the menu section of the restaurant, **Then** the system should display a list of dishes available for ordering. **Given** that I am viewing the restaurant's menu, **When** I click on a specific dish, **Then** the system should show me detailed information about that dish, including its description, price, and available customizations. **Given** that I am on the restaurant's menu page, **When** I use the search functionality to find a specific dish by name or keyword, **Then** the system should filter and display the relevant results based on my search query. **Given** that I am browsing the restaurant's menu, **When** I view a dish's details, **Then** the system should indicate whether the dish is currently available for ordering or not. **Given** that I am exploring a dish's details, **When** the dish offers customization options (e.g., toppings, sides), **Then** the system should present these options clearly and allow me to customize my order. **Given** that I am viewing a specific dish's details, **When** I want to go back to the restaurant's menu, **Then** the system should provide a straightforward way to navigate back to the menu page. **Given** that I am accessing the restaurant's menu on various devices, **When** I view the menu, **Then** the system should ensure a responsive and accessible interface for browsing and selecting dishes.
Story Points	4

Sprint 2

Títulu	**See TuBocado page with a share link and the number of Bocados**
Narrativa	**As a** customer, **I want to** be able to see TuBocado page with a share link and the number of Bocados, **So that** I can easily share the page and keep track of how many Bocados are available.
Criterios de Aceptación	**Given** I am a registered user **When** I navigate to the TuBocado page **Then** I should see the total number of Bocados displayed on the page. **Given** I am on the TuBocado page **When** I look at the page's interface **Then** I should see a share link that allows me to share the TuBocado page via social media, email, or other means. **Given** I have selected a sharing option, **When** I proceed to share, **Then** the link to the TuBocado page should be correctly embedded or attached in the chosen sharing medium.
Story Points	5

Títulu	**See header on the home screen**
Narrativa	**As a** customer **I want to** see the header on the home screen **So that** I can navigate easily
Criterios de Aceptación	**Given** I am a user on the home screen of the app or website, **When** the home screen loads, **Then** I should see a header at the top of the page. **Given** the header is displayed on the home screen, **When** I look at the header, **Then** I should see the app or website's logo on the left corner of the header. **Given** the header is displayed on the home screen, **When** I look at the header, **Then** I should see main navigation links . **Given** I am a registered and logged-in user, **When** I look at the header on the home screen, **Then** I should see options to access "My Account" or "Profile".
Story Points	4

Títulu	**Login to the restaurant page**
Narrativa	**As a** restaurant owner,

	I want to be able to log in to the restaurant page **So that** I can be recognized and access my restaurant's dashboard.
Criterios de Aceptación	**Given** that I am a registered restaurant owner with valid credentials, **When** I enter my valid username/email and password, and click "Log In," **Then** the system should redirect me to my restaurant dashboard or management page. **Given** that I am a registered restaurant owner, **When** I enter an invalid username/email and a valid password, and click "Log In," **Then** the system should display an error message indicating that the entered credentials are incorrect. **Given** that I am a registered restaurant owner, **When** I enter my valid username/email and an incorrect password, and click "Log In," **Then** the system should display an error message indicating that the entered credentials are incorrect. **Given** that I am a registered restaurant owner, **When** I enter an invalid username/email and an invalid password, and click "Log In," **Then** the system should display an error message indicating that the entered credentials are incorrect. **Given** that I am on the login page, **When** I successfully log in and access my restaurant dashboard, **Then** my session should be securely maintained and authenticated throughout my interaction with the application. **Given** that I am a registered restaurant owner and have forgotten my password, **When** I click on the "Forgot Password" link, provide my associated email address, and click "Send," **Then** the system should send me an email with clear instructions for securely resetting my password. **Given** that I access the login page, **When** I attempt to log in from different devices and browsers, **Then** the login page should be responsive and function correctly across these various platforms.
Story Points	4

Tíitulo	**See a location page**
Narrativa	**As a** customer, **I want to** be able to see a location page, **So that** I can input my location to view restaurants nearby
Criterios de Aceptación	**Given** I am a registered user, **When** I open the application or website, **Then** I should be prompted with a location page before being able

	to view restaurants. **Given** I am on the location page, **When** I look at the page's interface, **Then** I should see a field or option to input or select my location. **Given** I have previously inputted my location, **When** I revisit the application or website, **Then** the system should remember my last inputted location (optional based on the platform and privacy constraints). **Given** I want to change my location, **When** I am on the restaurant list page, **Then** there should be an easily accessible option for me to update my location.
Story Points	5

Tíitulo	**Change order status**
Narrativa	**As a** restaurant owner, **I want to** have the ability to view orders for my restaurant and change the status to accepted/declined/delivered/canceled, **So that** I can efficiently manage the order fulfillment process and provide timely service to my customers.
Criterios de Aceptación	**Given** that I am logged into the application as a restaurant admin, **When** I access the order management section, **Then** I should see a list of incoming orders displayed in the new table, showing essential details such as order number, items, customer information, and order status. **Given** that I am viewing the list of incoming orders, **When** I select a specific order from the list, **Then** I should have the option to accept the order, triggering a confirmation process that updates the order status to "Accepted". **Given** that I am viewing the list of incoming orders, **When** I select a specific order from the list, **Then** I should have the option to decline the order, triggering a confirmation process that updates the order status to "Declined" and notifies the customer with an appropriate message. **Given** that I have accepted or declined an order, **When** I revisit the order management section, **Then** the order's status should reflect my previous action (Accepted or Declined), and I should be able to track the progress of each order as it moves through the fulfillment process. **Given** that I am viewing the details of a specific order, **When** I click on an order, **Then** I should be able to see comprehensive order details, including items, quantities, special instructions and a timestamp of when the order was placed. **Given** that I am logged into the application, **When** a new order is received,

	Then I should receive a real-time notification or alert to promptly attend to the incoming order, ensuring timely service to the customer. **Given** that I am a restaurant admin, **When** I access the order management section, **Then** I should have the ability to view a history of past orders, allowing me to track and review previous order data for analysis and reference.
Story Points	4

Tíitulo	**Manage levels**
Narrativa	**As a** TuBocado manager I **want to** manage levels **So that** i can manage multilevel logic
Criterios de Aceptación	**Given** I am in the multilevel logic administration section, **When** I want to create a new level, **Then** I should have the ability to define a name, description, and permissions associated with the new level. **Given** I have created levels, **When** I need to edit the properties or permissions of an existing level, **Then** I should be able to modify the level's name, description, and permissions.
Story Points	8

Tíitulo	**Define the value of the Bocados**
Narrativa	**As a** TuBocado manager, **I want** to be able to define the value of the Bocados **So that I** can manage the prices of dishes and the generation of points by customers.
Criterios de Aceptación	**Given** I am logged in as a TuBocado manager, **When** I navigate to the "Manage Bocados" section, **Then** I should see an option to define the value of each Bocado. **Given** I am defining the value of a Bocado, **When** I enter the price in the designated input field, **Then** the system should validate the input and provide real-time feedback, indicating whether the price format is correct. **Given** I have entered the price for a Bocado, **When** I save the changes, **Then** the system should confirm the successful update and display a notification indicating that the price has been defined for the Bocado. **Given** I am defining the value of a Bocado, **When** I leave the price field blank and attempt to save,

	Then the system should prompt me with a validation message, indicating that the price field cannot be empty. **Given** I am defining the value of a Bocado, **When** I enter a non-numeric value in the price field, **Then** the system should display an error message, prompting me to enter a valid numeric value. **Given** I am defining the value of a Bocado, **When** I try to define the value of a Bocado that already has a defined price, **Then** the system should confirm whether I want to update the existing price or keep the current one. **Given** I am defining the value of a Bocado, **When** I encounter any unexpected errors, **Then** the system should display a user-friendly error message, guiding me on how to proceed or contact support for assistance.
Story Points	5

Títulо	**Select a payment method**
Narrativa	**As a** customer **I want to** be able to select a payment method provided by the system **So that** i can create an order
Criterios de Aceptación	**Given** that I am a user on the payment page and the system provides me with a list of valid payment methods **When** I select a payment method from the list. **Then** the system should display a confirmation of my choice and allow me to proceed with order creation **Given** that I am a user on the payment page and the system provides me with a list of valid payment methods. **When** I attempt to select a payment method that is not valid or not available **Then** the system should show me an error message indicating that the selected payment method is not valid or not available. **Given** that I am a user on the payment page and I have already selected a payment method. **When** I decide to change the selected payment method. **Then** the system should allow me to select a new payment method from the list of valid methods and update the payment information on the page according to my new selection. **Given** that I am a user on the payment page and I have already selected a payment method. **When** I decide to cancel the selection of a payment method. **Then** the system should allow me to deselect the current payment method and display the list of valid payment methods so that I can either select another one or cancel the order creation.
Story Points	5

Tíitulo	Edit points levels and percentages
Narrativa	**As a** TuBocado manager, **I want to** be able to edit points levels and percentages **so that** I can manage and control point generation system
Criterios de Aceptación	**Given** that I am logged as TuBocado administrator, **When** I open the web app, **Then** I should have access to a dedicated section or interface for managing points levels and percentages called Bocados. **Given** that I am in the Points Management section of the application, **When** I select the value of levels, **Then** I should have the ability to edit the number of existing levels **Given** that I am in the Points Management section of the application, **When** I select a points level (e.g., Nivel 1, Nivel 2, etc), **Then** I should be able to adjust the percentage of points awarded for each one when the action is done (sell) **Given** that I have made changes to points levels or percentages, **When** I confirm my edits, **Then** the system should save the modifications and update the point generation system accordingly.
Story Points	4

Sprint 3

Tíitulo	Edit the status of orders in real-time
Narrativa	**As a** restaurant owner, **I want to** be able to edit the status of orders in real-time **So that** customers receive accurate updates on their order progress.
Criterios de Aceptación	**Given** that I am a restaurant owner logged into the system, **When** I navigate to the order management section and select an order, **Then** the system should allow me to edit the order's status to provide real-time updates. **Given** that I modify the status of an order in real-time, **When** I update the status, **Then** the system should immediately send a notification to the customer with the updated status via their preferred communication channel (e.g., SMS, email). **Given** that I update the status of an order in real-time, **When** I make the status change, **Then** the system should record an accurate timestamp for the change to provide an audit trail. **Given** that I've updated an order's status in real-time,

	When I view order details, **Then** the system should display the complete history of status changes, including real-time updates. **Given** that I am making real-time status changes to an order, **When** multiple staff members attempt to update the same order simultaneously, **Then** the system should prevent conflicting updates and ensure data integrity. **Given** that I am editing order statuses in real-time, **When** I make updates, **Then** the system should provide a responsive and user-friendly interface that facilitates quick and efficient status changes.
Story Points	4

Tíitulo	**Buy sushi with points or cash**
Narrativa	**As a** customer **I want to** buy sushi with points or cash **So that** i can order sushi
Criterios de Aceptación	**Given** I have selected to pay with points for sushi and i have sufficient points, **When** I proceed to checkout, **Then** the system should deduct the appropriate number of points from my account and confirm the purchase. **Given** I have selected to pay with cash for sushi, **When** I proceed to checkout, **Then** the system should display the total cost in cash, confirm the purchase and add the correct amount of points to my account. **Given** I have insufficient points to cover the cost of sushi, **When** I attempt to pay with points, **Then** the system should prompt me to use an alternative payment method. **Given** I successfully completed a purchase of sushi using points or cash, **When** I review my transaction history, **Then** the purchase should be accurately recorded, showing the payment method used. **Given** I encounter any errors during the purchasing process, **When** I attempt to buy sushi, **Then** the system should provide clear error messages and guide me on how to resolve the issue.
Story Points	8

Tíitulo	**Search on restaurant list**

Narrativa	**As a** customer **I want to** search on restaurant list **So that** I can find the restaurant I want
Criterios de Aceptación	**Given** I want to search for a specific restaurant, **When** I type my search query into the search bar, **Then** the application should display relevant results as I type. **Given** I have received search results, **When** I want to refine my search, **Then** I should be able to modify my search query and see updated results dynamically. **Given** I am using the application on various devices, **When** I use the search functionality, **Then** it should be responsive and user-friendly, ensuring a seamless experience across different screen sizes. **Given** I encounter errors while using the search feature, **When** errors occur (e.g., network issues, server errors), **Then** I should receive clear error messages and instructions on how to proceed or troubleshoot.
Story Points	4

Tíitulo	**View Bocados available from a restaurant**
Narrativa	**As a** customer**,** **I want to** be able to view the Bocados available from a restaurant **So that** I can pay with them
Criterios de Aceptación	**Given** that I am a user accessing the app, **When** I navigate to a restaurant's menu section, **Then** the system should display a list of available Bocados. **Given** that I am accessing the list of available Bocados on various devices, **When** I view the list, **Then** the system should ensure a responsive and accessible interface for browsing and selecting Bocados.
Story Points	3

Tíitulo	**See restaurant's profile and menu**
Narrativa	**As a** customer, **I want to** be able to see the restaurant's profile and menu **So that** I can see the menu and decide what to order.
Criterios de Aceptación	**Given** I am a user browsing the application, **When** I search for or click on a restaurant, **Then** I should be redirected to that restaurant's profile page.

	Given I am on a restaurant's profile page, **When** I look for basic information, **Then** I should see elements like the restaurant's name, location, and a brief description. **Given** I am on the restaurant's profile page, **When** I click on the 'Menu' tab or button, **Then** I should be redirected to a page displaying the restaurant's menu items. **Given** I am viewing the restaurant's menu, **When** I browse through it, **Then** I should see details such as the name, description, and price of each item.
Story Points	8

Tíıtulo	**See the icon logo**
Narrativa	**As a** customer, **I want to** see the icon logo of the page on the window **So that** it looks nice
Criterios de Aceptación	**Given** I am accessing the webpage, **When** the page loads, **Then** I should see the icon logo displayed prominently in the browser window tab. **Given** I resize the browser window, **When** the page layout adjusts, **Then** the icon logo should remain visible and appropriately sized, maintaining its clarity and aesthetics. **Given** I navigate to different pages within the website, **When** the page loads, **Then** the icon logo should persistently appear in the browser window tab, providing consistent branding. **Given** I access the webpage from different devices (e.g., desktop, tablet, mobile), **When** the page loads, **Then** the icon logo should be responsive, adapting its size and resolution to fit the device screen without distortion or loss of quality. **Given** I am using a browser with limited support for certain image formats, **When** I access the webpage, **Then** the icon logo should be provided in a format compatible with a wide range of browsers, ensuring it is displayed correctly across different platforms.
Story Points	1

Tíitulo	See restaurant list and select one
Narrativa	**As a** customer, **I want to** see the sushi restaurant list and select one **So that** i can make an order
Criterios de Aceptación	**Given** that I am a user visiting the app, **When** I navigate to the restaurants section, **Then** the system should display a list of restaurants available for ordering. **Given** that I am viewing the list of restaurants, **When** I click on a specific restaurant, **Then** the system should show me detailed information about that restaurant **Given** that I am on the restaurant list page, **When** I use the search functionality to look for a specific restaurant by name, **Then** the system should filter and display the relevant results based on my search query. **Given** that I am browsing the list of restaurants, **When** I view a restaurant's details, **Then** the system should indicate whether the restaurant is currently open. **Given** that I am viewing a specific restaurant's details, **When** I want to go back to the list of restaurants, **Then** the system should provide a clear way to navigate back to the restaurant list. **Given** that I am using different devices to access the app, **When** I access the restaurant list, **Then** the system should provide a responsive and accessible interface for viewing and selecting restaurants.
Story Points	4

Tíitulo	Set my location
Narrativa	**As a** customer **I want to** be able to set my location **So that** I can search for restaurants
Criterios de Aceptación	**Given** I am a registered user and logged into the app, **When** I navigate to the "Search" page, **Then** I should see an option to "Set My Location". **Given** I am on the "Set My Location" option, **When** I manually input my location, **Then** the input should be validated, and the map should re-center to the manually inputted location.
Story Points	4

Sprint 5

Tíitulo	**Cancel order**
Narrativa	**As a** customer, **I want to** be able to cancel an order **So that** i can order again something different
Story Points	5

Tíitulo	**Return Bocados to my account**
Narrativa	**As a** customer, if i cancel an order **I want to** see my Bocados return to my account **So that** i can use that Bocados again
Story Points	5

Tíitulo	**Get the information of a specific restaurant**
Narrativa	**As a** TuBocado manager, **I want to** get the information of a specific restaurant **So that** I can manage it correctly
Story Points	4

Tíitulo	**Accept or decline restaurant requests**
Narrativa	**As a** TuBocado manager, **I want to** be able to accept or decline restaurant requests **So that** they can use the system
Story Points	5

Tíitulo	**Delete customer points**
Narrativa	**As a** TuBocado manager **I want to** be able to delete customer points **So that** we could have a correct history of the customer transactions
Story Points	4

Tíitulo	View real orders
Narrativa	**As a** restaurant owner, **I want to** have the ability to view real orders for my restaurant, **So that** I can efficiently manage the order process.
Story Points	5

Tíitulo	Have an administrator
Narrativa	**As a** TuBocado manager **I want to** have an administrator **So that** I can use the application
Story Points	2

Sprint 6

Tíitulo	Filter restaurants
Narrativa	**As a** customer, **I want to** be able to filter restaurants by less time **So that** I can order the fastest restaurant
Story Points	5

Tíitulo	Send email to restaurant
Narrativa	**As a** TuBocado manager **I want to** send an email to a restaurant if it is accepted or declined **So that** it receives an explanation
Story Points	4

Tíitulo	Cancel order
Narrativa	**As a** restaurant owner **I want to** cancel an existing order **So that** i can attend a situation why i cannot prepare the order
Story Points	4

Tíitulo	
Narrativa	**As a** restaurant owner, **I want to** create and modify my dishes **So that** customers can buy them
Story Points	5

Tíitulo	**Check my account status**
Narrativa	**As a** customer, **I want to** be able to check my account status to see my bites transaction **So that** i can have traceability
Story Points	5

Tíitulo	**See Bocados metrics**
Narrativa	**As a** customer, **I want to** be able to see metrics on TuBocados page **So that** i can have traceability of my referrals
Story Points	4

Tíitulo	**Create and modify categories**
Narrativa	**As a** restaurant owner, **I want to** create and modify my categories **So that** i can organize my dishes
Story Points	5

Tíitulo	**Pay with credit card**
Narrativa	**As a** customer **I want to** pay an order with a credit card **So that** i can buy without cash or bites
Story Points	8

Tíitulo	**Filter restaurants**
Narrativa	**As a** customer **I want to** filter restaurants by location **So that** i can found them easily

Story Points	5

Sprint 7

Tíitulo	**Edit my personal info**
Narrativa	**As a** customer, **I want to** see my profile and edit my personal info **So that** i can keep my information updated
Story Points	5

Tíitulo	**See my Bocados**
Narrativa	**As a** customer, **I want to** see the numbers of Bocados I generated, **So that** I can buy sushi with them lat
Story Points	5

Tíitulo	**See my order in home**
Narrativa	**As a** customer, **I want to** see my order in the home page **So that** I can see thee different status
Story Points	5

Tíitulo	**Generate Bocados when order**
Narrativa	**As a** customer, **I want to** be able to generate Bocados when I create a new order **So that** I can continue buying
Story Points	4

Tíitulo	**Register my restaurant**
Narrativa	**As a** restaurant owner, **I want to** register my restaurant and wait to be accepted **So that** i can sell my dishes
Story Points	5

Tíitulo	Check account status
Narrativa	**As a** customer, **I want to** check my account status to see my bites transaction **So that** i can have traceability
Story Points	4

Tíitulo	See active restaurants
Narrativa	**As a** customer, **I want to** be able to see active restaurants **So that** i can make an order
Story Points	4

Sprint 8

Tíitulo	See money generated by restaurants
Narrativa	**As a** TuBocado manager **I want to** be able to see how much money each restaurant generates **So that** I can see what the finances are like
Story Points	5

Tíitulo	See money generated by my restaurant
Narrativa	**As a** restaurant owner **I want to** be able to see how much money my restaurant generates **So that** I can see what the finances are like
Story Points	5

Tíitulo	How many points were redeemed
Narrativa	**As a** restaurant owner **I want to** be able to see how many points were redeemed at my restaurant **So that** I can see what the finances are like
Story Points	5

Tíitulo	**Get all and edit customers**
Narrativa	**As a** TuBocado manager, **I want to** edit new customers and get all of them **So that** i can have control of the system
Story Points	5

Tíitulo	**Give my review**
Narrativa	**As a** customer **I want to** give my review about my order **So that** i can give my opinion about it
Story Points	2

Tíitulo	**Get all and add TuBocado Managers**
Narrativa	**As a** TuBocado Manager, **I want to** create new TuBocado managers and get all of them **So that** i can have control of the system
Story Points	4

Tíitulo	**See reviews**
Narrativa	**As a** restaurant manager, **I want to** view all the reviews from my restaurant **So that** i can have an idea of customers opinion and improve quality and attention
Story Points	3

Tíitulo	**See restaurant reviews**
Narrativa	**As a** customer **i want to** see the restaurants reviews **So that** i can choose where to eat
Story Points	4

Tíitulo	**See history of my orders**
Narrativa	**As a** customer **i want to** see the history of my orders **So that** i can keep track of my orders

Story Points	5

Tíitulo	**See my restaurant reports**
Narrativa	**As a** restaurant owner **I want to** see all the reports about my restaurant such as sales, orders, averages, graphs **So that** i can see my restaurant results and make decisions
Story Points	5

Sprint 9

Tíitulo	**See points redeemed per restaurant**
Narrativa	**As a** TuBocado manager **I want to** get how many points were redeemed per restaurant **So that** i can see what the finances are like
Story Points	5

Tíitulo	**Generate Bocados if i'm active**
Narrativa	**As a** customer, **I want to** generate Bocados if i'm active that month **So that** i keep active on the system
Story Points	2

Sprint 10

Tíitulo	**Restore password**
Narrativa	**As a** customer, **I want to** restore my password **So that** I can login to the app
Story Points	5

Tíitulo	**Restore password**
Narrativa	**As a** restaurant owner, **I want to r**estore my password **So that** I can login to the app

Story Points	4

Tíitulo	**Restore password**
Narrativa	**As a** TuBocado manager, **I want to** restore my password **So that** I can login to the app
Story Points	3

Tíitulo	**Pick up**
Narrativa	**As a** customer **I want to** order food with pick up **So that** I can not spend on *delivery*
Story Points	4

Tíitulo	**Close restaurant**
Narrativa	**As a** restaurant owner **I want to** be able to close my restaurant **So that** i can go inactive if im not selling right now
Story Points	2

Tíitulo	**Order status update**
Narrativa	**As a** customer **I want to** be able to see the order status update in real time **So that** i keep informed about my order
Story Points	5

Sprint 11

Tíitulo	**Receive feedback**
Narrativa	**As a** restaurant owner, **I want to** receive feedback like a sound every time I receive a new order **So that** i can be always updated
Story Points	5

Sprint 12

Tíitulo	**See referral tree**
Narrativa	**As a** Customer **I want** to see my referral tree with drag option **So that** i can be aware of my network
Story Points	5

Sprint 13

Tíitulo	**Edit personal info**
Narrativa	**As a** Restaurant Owner **I want** to edit my personal information **So that** customers can see the correct info
Story Points	5

Requerimientos No Funcionales

Identificación de requerimientos no funcionales	**Descripción del Requerimiento**
US-1 Cambios del cliente	**Cambios del cliente:** El sistema debe desarrollarse de modo que si el cliente plantea durante el desarrollo algún cambio, el costo de hacerlos sea lo más bajo posible.
US-2 Extensibilidad	**Extensibilidad:** El sistema debe ser desarrollado de forma que si se desea incorporar nuevas funcionalidades, conlleven el menor coste posible.
US-3 Reutilización	**Reutilización:** El sistema deberá estar modularizado para poder reutilizar partes de él en futuros proyectos del cliente.
US-4 Métodos de autenticación y autorización	**Métodos de autenticación y autorización:** El sistema debe contar con medidas de seguridad para el acceso a datos y acciones realizadas por el usuario según su rol, haciendo autenticaciones para cada petición de ser necesario.
US-5 Métodos de pago	**Métodos de pago:** El sistema debe contar con medidas de seguridad a modo de proteger las transacciones de pago de los consumidores utilizando una api de terceros confiable.
US-6 Fácil de aprender	**Fácil de aprender:** El sistema debe permitir el fácil aprendizaje a los consumidores, dueños de restaurantes y managers de TuBocado brindando una interfaz gráfica que cumpla con las Heurísticas de Nielsen.
US-7 Interfaz atractiva	**Interfaz atractiva:** El sistema debe brindar una interfaz atractiva y eficiente que cumpla con los prototipos iniciales.
US-8 Responsive	**Responsive:** La web de los consumidores debe ser responsive para poder adaptarse a todas las pantallas de los móviles y computadoras
US-9 Compatibilidad de navegadores	**Compatibilidad de navegadores:** La aplicación web debe ser compatible y funcionar de manera óptima en los navegadores Safari y Chrome,

	garantizando una experiencia de usuario consistente y satisfactoria en ambas plataformas
US-10 Reintento de generación de puntos	**Reintento de generación de puntos:** Se debe poder reintentar la generación de puntos en caso de que no logren cumplirse, sea cual sea el motivo.
US-11 Disponible todo el tiempo	**Disponible todo el tiempo:** La aplicación debe tener una disponibilidad del 98% para que los usuarios puedan realizar órdenes en cualquier momento.
US-12 Calidad de las imágenes de los platos	**Calidad de las imágenes de los platos:** La calidad de imágen de los platos y productos debe ser mayor o igual a 720p
US-13 Tiempos de respuesta notificaciones	**Tiempos de respuesta notificaciones:** Se debe poder ver el estado de una orden en tiempo real. Tanto del lado del dueño de restaurante como del lado del consumidor.
US-14 Tiempo de respuesta creación ordenes	**Tiempo de respuesta creación ordenes:** Se debe poder procesar una orden nueva en menos de 5s.
US-15 Módulo de gestión de puntos desacoplado	**Módulo de gestión de puntos desacoplado:** El sistema de Bocados debe ser un servicio aparte del resto, de modo que pueda ser fácilmente integrado en otros sistemas o aplicaciones sin requerir modificaciones extensivas.
US-16 Comunicación entre servicios	**Comunicación entre servicios:** Debe asegurar la integración fluida con el servicio de gestión de puntos y otros posibles servicios.
US-17 Gestión de errores y fallas	**Gestión de errores y fallas:** El sistema debe brindar la información necesaria que permita hacer un diagnóstico preciso y eficiente sobre las causas de cualquier error que se presente.
US-18 Utilizar contenedores	**Utilizar contenedores:** El sistema debe ser contenerizado utilizando Docker, permitiendo un despliegue continuo.
US-19 Integración continua	**Integración continua:** Los sistemas deben implementar prácticas de Integración Continua (CI) y Despliegue Continuo (CD) para mejorar la deployabilidad del sistema. Esto asegura que las actualizaciones del *software* puedan ser desarrolladas, probadas y desplegadas de manera rápida, manteniendo la calidad del sistema.

A.17. Matrices de Pugh - tecnologías

Hosting

Atributo	Peso	Azure	AWS
Alcance global	1	0	1
Servicios ofrecidos	1	1	1
Precios	2	0	1
Comunidad	1	1	1
Facilidad de uso	3	1	0
Soporte y documentación	2	1	1
Seguridad	2	1	1
Fiabilidad	1	1	1
Deploy de aplicaciones	2	1	1
Bases de datos	2	1	1
Experiencia del equipo	2	-1	1
Interes del cliente	3	0	1
Suma		12	19
Ranking		2	1

Lenguajes de *Backend*

Atributo	Peso	NodeJS	Ruby on Rails (Ruby)	Laravel (PHP)
Rendimiento	2	1	0	1
Ecosistema y Bibliotecas	1	1	1	1
Comunidad	2	1	0	1
Curva de aprendizaje	2	1	-1	0
Mantenibilidad	2	1	1	1
Seguridad	2	1	1	1
Integración con Frontend	2	1	-1	1
Uso en la industria	1	1	1	1
Experiencia del equipo	3	1	-1	0
Suma		17	-1	12
Ranking		1	3	2

Lenguajes de *Frontend*

Atributo	Peso	ReactJs	AngularJs	VueJs
Definición de la tecnología	1	1	1	1
Reutilizacion de codigo	2	1	0	1
Performance	2	1	0	1
Lenguaje de programación	2	1	1	1
Seguridad	3	1	1	1
Soporte	2	1	0	1
Costo	1	1	1	1
Curva de aprendizaje	1	1	0	1
Fiabilidad	2	1	1	1
Experiencia del equipo	3	1	0	-1
Suma		19	9	13
Ranking		1	3	2

Tipos de base de datos

Atributo	Peso	Relacional	No relacional
Esquema	1	1	1
Consultas	2	1	1
Transacciones	2	1	0
Seguridad	3	1	1
Cuando usarlo	2	1	0
Experiencia del equipo	3	1	0
Suma		13	4
Ranking		1	2

Motor de base de datos

Atributo	Peso	SQL Server	PostgresSql
Licenciamiento	1	0	1
Rendimiento	2	1	1
Comunidad	1	1	1
Seguridad	2	1	1
Extensibilidad	2	0	1
Sistemas operativos compatibles	2	0	1
Concurrencia	2	1	1
Compatibilidad con AWS/Azure	3	1	1
Experiencia del equipo	3	1	0
Suma		13	15
Ranking		2	1

A.18. *Definition of Done* de cada Release

***Definition of Done Release* 1**

- Se debe haber mergeado todo a la rama Main
- Se debe haber llevado a cabo el proceso de las tareas correspondientes a la integración continua que brindan las Github Actions.
- Se deben haber ejecutado las pruebas automáticas.
- Debe estar generada la imagen de Docker
- Debe estar subida la imagen de docker a Amazon ElasticBeanstalk.
- Debe actualizarse el cluster de ECS con nueva imagen
- Cada historia de usuario/funcionalidad está codificada, compilada, probada y desplegada.
- Las historias de usuario/funcionalidades cumplen todos los criterios de aceptación acordados con el *product owner*.
- Las funcionalidades/historias de usuario comprometidas funcionan en todos los ambientes comprometidos, es decir, local y producción.
- El Código desarrollado está comentado y estandarizado.
- El Código está almacenado en el repositorio de control de versiones.
- La documentación comprometida para el *Sprint* está realizada:
    - Documento de arquitectura.
    - Documento de diseño.
    - Otros documentos.
- Se cumple con los requisitos no funcionales, por ejemplo:
    - Estándares de usabilidad y UX.
    - Navegadores y versión de los mismos sobre los que debe funcionar.
    - Otros.

- Las clases de negocio cuentan con pruebas unitarias que verifican su correcto funcionamiento.
- El porcentaje de cobertura de pruebas unitarias al código es >= a un 85%.
- Se realizaron pruebas de aceptación con un resultado positivo.
- Se realizaron pruebas con el PO y con los usuarios.
- Se realizaron pruebas de integración con un resultado positivo.
- Se realizaron encuestas de satisfacción.

Definition of Done Release 2

- Se debe haber mergeado todo a la rama Main
- Se debe haber llevado a cabo el proceso de las tareas correspondientes a la integración continua que brindan las Github Actions.
- Se deben haber ejecutado las pruebas automáticas.
- Debe estar generada la imagen de Docker
- Debe estar subida la imagen de docker a Amazon ElasticBeanstalk.
- Debe actualizarse el cluster de ECS con nueva imagen
- Cada historia de usuario/funcionalidad está codificada, compilada, probada y desplegada.
- Las funcionalidades/historias de usuario comprometidas funcionan en todos los ambientes comprometidos, es decir, local y producción.
- El Código desarrollado está comentado y estandarizado.
- El Código está almacenado en el repositorio de control de versiones.
- La documentación comprometida para el *Sprint* está realizada:
    - Documento de arquitectura.
    - Documento de diseño.
    - Otros documentos.
- Se cumple con los requisitos no funcionales, por ejemplo:
    - Estándares de usabilidad y UX.
    - Navegadores y versión de los mismos sobre los que debe funcionar.
    - Otros.
- Las clases de negocio cuentan con pruebas unitarias que verifican su correcto funcionamiento.
- El porcentaje de cobertura de pruebas unitarias al código es >= a un 85%.
- Se realizaron pruebas de aceptación con un resultado positivo.
- Se realizaron pruebas con el PO y con los usuarios.
- Se realizaron pruebas de integración con un resultado positivo.
- Se realizaron encuestas de satisfacción.

Definition of Done Release 3

- Se debe haber mergeado todo a la rama Main
- Se debe haber llevado a cabo el proceso de las tareas correspondientes a la integración continua que brindan las Github Actions.
- Se deben haber ejecutado las pruebas automáticas.
- Debe estar generada la imagen de Docker
- Debe estar subida la imagen de docker a Amazon ElasticBeanstalk.
- Debe actualizarse el cluster de ECS con nueva imagen
- Cada historia de usuario/funcionalidad está codificada, compilada, probada y desplegada.
- Las historias de usuario/funcionalidades cumplen todos los criterios de aceptación acordados con el *product owner*.

- Las funcionalidades/historias de usuario comprometidas funcionan en todos los ambientes comprometidos, es decir, local y producción.
- El Código desarrollado está comentado y estandarizado.
- El Código está almacenado en el repositorio de control de versiones.
- La documentación comprometida para el *Sprint* está realizada:
    - Documento de arquitectura.
    - Documento de diseño.
    - Otros documentos.
- Se cumple con los requisitos no funcionales, por ejemplo:
    - Estándares de usabilidad y UX.
    - Navegadores y versión de los mismos sobre los que debe funcionar.
    - Otros.
- Las clases de negocio cuentan con pruebas unitarias que verifican su correcto funcionamiento.
- El porcentaje de cobertura de pruebas unitarias al código es >= a un 85%.
- Se realizaron pruebas de aceptación con un resultado positivo.
- Se realizaron pruebas con el PO y con los usuarios.
- Se realizaron pruebas de integración con un resultado positivo.
- Se realizaron encuestas de satisfacción.

Definition of Done Release 4

- Se debe haber mergeado todo a la rama Main
- Se debe haber llevado a cabo el proceso de las tareas correspondientes a la integración continua que brindan las Github Actions.
- Se deben haber ejecutado las pruebas automáticas.
- Debe estar generada la imagen de Docker
- Debe estar subida la imagen de docker a Amazon ElasticBeanstalk.
- Debe actualizarse el cluster de ECS con nueva imagen
- Cada historia de usuario/funcionalidad está codificada, compilada, probada y desplegada.
- Las historias de usuario/funcionalidades cumplen todos los criterios de aceptación acordados con el *product owner*.
- Las funcionalidades/historias de usuario comprometidas funcionan en todos los ambientes comprometidos, es decir, local y producción.
- El Código desarrollado está comentado y estandarizado.
- El Código está almacenado en el repositorio de control de versiones.
- La documentación comprometida para el *Sprint* está realizada:
    - Documento de arquitectura.
    - Documento de diseño.
    - Otros documentos.
- Se cumple con los requisitos no funcionales, por ejemplo:
    - Estándares de usabilidad y UX.
    - Navegadores y versión de los mismos sobre los que debe funcionar.
    - Cumpla heurísticas de Nielsen
    - Otros.
- Las clases de negocio cuentan con pruebas unitarias que verifican su correcto funcionamiento.
- El porcentaje de cobertura de pruebas unitarias al código es >= a un 85%.
- Se realizaron pruebas de aceptación con un resultado positivo.
- Se realizaron pruebas con el PO y con los usuarios.
- Se realizaron pruebas de integración con un resultado positivo.
- Se realizaron encuestas de satisfacción.

A.19. Historias de usuario estimadas - hoja de cálculo

User Stories / Tasks / Fixes /Enviroment Setup	STORY POINTS ESTIMATION	STORY POINTS	Esperado (hs)	Total (hs-p) (hs)
As a customer, I want to be able to register through a specialized link So that I can easily create an account and start using the service.		5		9,1
As a tuBocado manager, I want to be able to login to the manager site so that I can securely access management functionalities and content.		5		9,58
As a customer, I want to be able to login so that I can order sushi and access personalized features and content.		5		8,93
As a tuBocado manager, I want to have restaurants in the database so that customers can see the ones that are available.		4		8
As a restaurant owner, I want to be able to navigate between pages using a sidebar so that I can access different sections of the application efficiently		3		3,4
As a tuBocado manager, I want to navigate between pages using a sidebar so that I can access different sections of the application efficiently		5		8,33
As a customer, I want to be able to view the menu of each restaurant so that I can choose a dish to order.		4		6,73
As a customer, I want to have initial menus so that restaurants can display their available dishes to customers.		3		3,71
As a developer I want to set up TuBocado - Backend		5		9,78
As a developer I want to set up Points System - Backend		**4**		6,88
As a developer I want to set up Frontend - Restaurant		**4**		6,53
As a developer I want to set up Frontend - TuBocado		**2**		2,78
Sprint 1				
As a tuBocado manager, I want to have at least one customer registered so that the customer can be used to refer other customers.		2		1,33
As a customer, I want to be able to see TuBocado page with a share link and the number of Bocados, so that I can easily share the page and keep track of how many Bocados are available.		5		10,21
As a customer I want to see the header on the home screen so that I can navigate easily		4		5,59
As a restaurant owner, I want to be able to log in to the restaurant page so that I can be recognized and access my restaurant's dashboard.		4		5,31
As a customer, I want to be able to see a location page, so that I can input my location				
As a restaurant owner, I want to have the ability to view orders for my restaurant and change the status to accepted/declined/delivered/canceled, so that I can efficiently manage the order fulfillment process and provide timely service to my customers.		4		11,82
As a TuBocado manager I want to manage levels so that i can manage multilevel logic		8		21,53
As a TuBocado manager, I want to be able to define the value of the Bocados so that I can manage the prices of dishes and the generation of points by customers.		5		18,38
As a tuBocado manager, I want to have missing restaurant attributes so I can have the information complete		3		3,5
As a customer I want to be able to select a payment method provided by the system so i can create an order		5		9,25
As tuBocado manager, I want to be able to edit points levels and percentages so that I can manage and control point generation system		4		6,07
Sprint 2				
As a restaurant owner, I want to be able to edit the status of orders in real-time so that customers receive accurate updates on their order progress.		4		6,33
As a customer I want to buy sushi with points or cash so that i can order sushi		8		57,78
As a customer I want to search on restaurant list so that I can find the restaurant I want		4		4,93
As a customer, I want to be able to view the Bocados available from a restaurant so that I can pay with them		3		3,3
As a customer, I want to be able to see the restaurant's profile and menu So that I can see the menu and decide what to order.		8		41,8
As a customer I want to see the icon logo of the page on the window so that it looks nice		1		1
As a customer I want to see the sushi restaurant list and select one so that i can make an order		4		7,52
As a customer I want to initialize points when I register so that i can start accumulating		3		3,73
As a customer I want to be able to set my location so that I can search for restaurants		4		4,55
As a developer I should be able to run the database with the migration and scrpts so I can see the database with data		4		5,97
As a developer I should be able to have a command to seed initial information so that I can have everything in one place		4		4,12
Fix UI errors that come from the Backend		3		3,56
Fix added description to dish seed		4		5,33
Fix: TuBocado Backend revert commit. TypeScript errors in admin and restaurantAdmin controllers		3		3,5

Fix: Save session user after login in restaurant and TuBocado admin		4		7
Sprint 3				
As a developer, I want to configure "Deployment - TuBocado Backend" so I can deploy it		4		6
As a developer, I want to configure configure "Deployment - Points Backend" so I can deploy it		4		5,65
As a developer, I want to configure configure "Deployment - Administrators" so I can deploy it		4		6
As a developer, I want to configure configure "Deployment - Users" so I can deploy it		4		5,5
Sprint 4				
TOTAL RELEASE 1	0	175	0	371,57
As a customer, i should be able to cancel an order so that i can order again something different	5	5	11	9,32
As a customer, If i cancel an order I want to see my bocados return to my account	5	5	14	11,33
As a tuBocado manager, i want to get the information of a specific restaurant so that I can manage it correctly	4	4	8	4,68
As a tuBocado manager, i should be able to accept or decline restaurant requests so that they can use the system	8	5	29	23,5
As a tuBocado manager I want to have inicial information for customer, levels and parameters.	3	3	4	3,5
As a tuBocado manager I should be able to delete customer points so that we could have a correct history of the customer transactions	4	4	4,5	4,31
As a restaurant owner, I want to have the ability to view real orders for my restaurant, so that I can efficiently manage orders process.	5	5	19	17,36
As a tuBocado manager I want to have an admistrator so I can use the application	2	2	2,5	2,33
As a developer I should have images in dishes and restaurants so I can test front	2	3	3	3,12
As a developer I should be able to generate orders so I can test the front	4	2	4,5	2,34
As a developer, I want to configure S3 so i can upload images	2	2	2	1,35
As a developer, I want to create pending transactions when an order is created	4	4	6	5,67
Fix/delete database before running	2	2	1,5	2
Fix: Math floor on user bocados	1	1	0,5	0,5
Fix: Display errors on the Frontend	1	1	0,5	0,5
Fix: Get orders for restaurants to manage undefined	2	2	1,5	2
Fix, change user seed UUID and use the same instance of the database for the seeds	3	3	4	3,5
Fix seed category	1	1	1	0,5
Fix: Always generate the same information with scripts	2	2	3	2,5
Fix: Migration name orders	2	2	3	2,56
Fix: Double import error and error handling after login	2	2	2,5	2,65
Fix: add logo	1	1	1	0.5
Fix: show correct bocados	1	1	1	1
Fix: inputs order in sign up	2	1	2	1
Fix: add button to login and mach the style with the rest of the page	2	1	1,3	1
Fix: remove shopping cart from header in login, home and orders views	1	1	1	1
Fix: change colour of the bar in home view to black	1	1	1	1
Fix: in home view change style of filter	1	1	1	0.5
Fix: in restaurant login view, add an option for the rut of the restaurant	1	1	1	0,5
Sprint 5				
As a customer, I want to be able to filter restaurants by less time so that I can order the fastest restaurant	5	5	16	12,23
As a customer i want to filter restaurants by location so that i can found them easily	5	5	21	12,79
As a tuBocado manager i want to send an email to a restaurant if it is accepted or declined so that it receives an explanation	4	4	8	7,18
As a restaurant owner i want to cancel an existing order so that i can attend a situation why i cannot prepare the order	4	4	6	5,3
As a restaurant owner, i want to create and modify my dishes so that costumers can buy them	8	5	25	17,43
As a customer, i should be able to check my account status to see my bites transaction so that i can have traceability	5	5	9	9,31
As a customer, I should be able to see metrics on TuBocados page	5	4	8,5	6,35
As a restaurant owner, i want to create and modify my categories so that i can organize my dishes	5	5	20	18,21
As a customer i want to pay an order with a credit card so that i can buy without cash or bites	8	5	24,5	23,1
Fix/configure S3 images	2	2	3	2,5
Fix: Math floor bocados	3	2	3,5	2,67
Fix: Something went wrong bocados	2	2	2	1,5
Fix: Amount in cart	5	2	10	2,56
Fix: Shipping price	5	5	10	12,5
Fix: Admin sites UI errors	2	2	2	2,33
Fix: 500 Backend error for TuBocado	2	2	2	2,56
Fix: Preparing time	2	3	3	3,5
Fix: change layout of the chekout option in orders view	1	1	1	1
Fix: change shopping cart to the top with the orders in orders view	2	2	2,5	2
Sprint 6				
As a customer i want to pay an order with a credit card so that i can buy without cash or bites	5	5	9	8,7
As a customer, I want to see my profile and edit my personal info	5	5	17	15,87
As a customer, I want to update my credentials	5	5	11	9,56
As a customer, I want to see the numbers of bocados I generated, so that I can buy sushi with them later	5	5	15	11,23

As a customer, I should be able to see my generates points at the end of the day	5	4	8,5	7,43
As a customer, I want to see my order in the home page so that I can see thee different status	5	5	10	11,35
As a customer, I should be able to generate bocados when I create a new order so that I can continue buying	4	4	6	5,67
As a restaurant owner, i want to register my restaurant and wait to be accepted so that i can sell my dishes	5	5	10	8,63
As a customer, I want to be able to review an order	5	4	10	6,34
As a customer, i want to check my account status to see my bites transaction so that i can have traceability	4	4	6	6,65
As a customer I should be able to see active restaurants	4	4	7	6,9
Fix: Change shipping address to address	3	3	4	3,5
Fix: update script	4	2	6	2,78
Fix: Restaurants system tables with pagination	3	3	4	3,5
Fix: Type errors	4	4	5	5,5
Fix: Change the name of the migration 20233108432144_add_image_column_to_dishes	1	2	0,5	0,6
Fix: Points refund	4	3	5	3,21
Sprint 7				
TOTAL RELEASE 2	220	200	445,3	373,46
As a tuBocado manager I should to be able to see how much money each restaurant generates so I can see what the finances are like	5	5	22	19,33
As a restaurant owner I should be able to see how much money my restaurant generates so I can see what the finances are like	5	5	23	21,79
As a restaurant owner I should be able to see how many points were redeemed at my restaurant so I can see what the finances are like	5	5	15	13,78
As a tuBocado manager, i want to edit new customers and get all of them	5	5	14	13,61
As a customer i want to give my review about my order so that i can give my opinion about it	1	2	0,5	1,2
As a tuBocado Manager, I want to create new tuBocado managers and get all of them	4	4	6	8,2
As a customer i should be able to give my review about my order so that i can give my opinion about it	4	4	7	7,21
As a restaurant manager, I want to view all the reviews from a restaurant	3	3	3,5	3,35
As a customer i want to see the restaurants reviews so that i can choose where to eat	4	4	8	7,75
As a customer i want to see the history of my orders so that i can keep track of my orders	5	5	9	8,68
As a restaurant owner I want to see all the reports about my restaurant so that i can see my restaurant results and make decisions	5	5	10	8,97
Fix: Batch	2	3	3	3
Fix: remove logo form login	1	1	1	0,5
Fix: make the password to be visibile if the user wants in login	2	2	2	1,5
Fix: add button to go back to login	2	1	1,5	1
FiX: add link to forgot password in login	1	1	0,5	0,5
Fix: home view, change logo to the new one	1	1	1	0,5
Fix: in home view, change directions layout	1	1	1	1
Fix: in profile view, change layout to the new version (figma)	2	2	2	3
Fix: in profile view, add orders history	3	3	4	3,5
Fix: in restaurant view chage colour to violet	2	1	2	1
Fix: in restaurnt login add option to se the password	2	2	2	1,5
Fix: add login button on restaurant sign up view	2	1	1,5	1
Fix: add option to edit the categories	2	2	2	2,5
Sprint 8				
As a restaurant owner I want to edit my information so that customers can see the correct info	5	5	12	11,2
As a tuBocado manager I want to get how many points where redeemed per restaurant so that i can see what the finances are like	8	5	24,5	22,42
As a customer i want to see my referral network so that i can see how many people i have below	5	5	20	18,23
As a tuBocado manager, i should be able to create new managers so that new manager can login to de app	4	4	5	4,23
As a tuBocado manager, I want to see the customers of the system so that they can have control of it	4	3	5,5	3,5
As a restauran owner I want to see the reviews from the customers of my restaurant so that i can improve que quality and attention	3	1	4	1
As a customer, I should be able to	2	2	2	2,5
As a customer, I want to generate Bocados if i'm active that month	4	2	5	2,3
Fix: Add more dishes in different categories within a restaurant, decrease bocados for user Valerio to 33	1	2	1	2,5
Fix: Differentiate with colors inputs that are edited and those that are not in admin forms	2	2	2	2
Fix: change button on order restuarant view to match the style of the rest	3	3	3,4	3
Fix: restaurant order view add direction, order status and Bocados options	1	2	1	2
Fix: add pagintation to finance view	2	1	2	1
Fix: add option to see password on managers login	2	2	2	1,5
Fix: Refresh plates	4	4	6	5
Fix: Restaurant seed	4	4	7	5,5
Fix: add forgot password on managers login	3	2	3,5	1,5
Fix: only edit/add admins, not delete	3	3	4	3,5
Sprint 9				
As a customer, I want to restore my password so that I can login to the app	5	5	13	11,64

As a restaurant owner, I want to restore my password so that I can login to the app	4	4	5	5,3
As a tuBocado manager, I want to restore my password so that I can login to the app	3	3	4	3,2
As a customer, I want to see the restaurants review on the home page	3	2	3,5	2,4
As a customer i should be able to order food with pick up so that I can not spend on delivery	5	4	9	7,92
As a restaurant owner I want to be able to close my restaurant	3	2	3,5	1,4
As a restaurant owner i should de able to see reports about my restaurant such as sales, orders, averages, graphs so that i can see my restaurant results and make decisions	5	5	9	11,18
As a restaurant owner i should be able to see the reviews from the customers of my restaurant so that i can improve que quality and attention	5	5	15	14,5
As a customer I want to be able to see the order status update in real time	5	5	11	13,4
As a customer I should be able to select "pickup" so I can pick up my food	4	4	5	4,38
As a developer, I want to save in the database saves restaurant open and close time like this "08:30 AM" and not like this "8:30"	5	5	14	15,5
Fix: Add tests	1	2	1	2
Fix: change the layout of edit dish view	3	3	4	3,5
Fix: change text style in the buttons, edit/delete category	3	2	4	2,5
Fix: Seed	4	4	5	4,5
Fix: Cancel order MercadoPago	4	2	4,5	2,5
Fix: change layout of dishes to the new design	3	2	3,5	3
Fix: Change the way the reports endpoint receives parameters and fix logic	5	4	10	5,76
Sprint 10				
TOTAL RELEASE 3	199	183	375,4	341,83
As a restaurant owner, I should be able to receive feedback like a sound every time I receive a new order	5	5	12,3	15,4
As a restaurant owner I should be able to se the map when i update my information	5	5	16	12
As a customer I want a space to handle my locations so I can manage them	5	5	8	9,6
Configure new deployment enviroment	5	5	12	14,3
Fix: Restaurant filters	4	3	5	3,4
Fix: Set address, the map does not appear sometimes	4	3	5	4,23
Fix: By default when an admin accepts a new restaurant, it should not be visible	3	3	5	3,56
Fix: Emails	4	4	5	5,2
Fix: Restaurant update, hours are not updating	4	4	6	5,1
Fix: Add close button to edit menu	2	2	2	1,5
Fix: Bug when ordering with pickup	3	3	3,5	3
Fix: Remove empty white space in the login	2	2	2,5	2
Fix: Add search bar to mobile version	2	1	1,5	1
Fix: in login view, match radius and shade of the button to the rest of the views	2	1	1,5	1
Fix: sign in button, same size as the filds of the form	2	2	1,5	1,5
Fix: show reviews in orders view, and the side of the restaurant	3	2	4	3,2
Fix: show orders history on the perfil view	2	4	3	5
Fix: multi mapping locations	2	3	2,5	3
Fix: improve hovers	3	3	3	3
Fix: add option to write reviews on the orders history view	2	4	3	5
Sprint 11				
As a Customer I should be able to see my referral tree with drag option	**5**	**5**	**12**	11,4
As a Restaurant Owner I should be able to change the closing and opening hours depending on the day	**5**	**5**	**12**	12,6
As a customer I should be able to see an error screen when my creditcard fails	5	5	12	13
As a TuBocado manager I should be able to see a 404 screen when I enter a non existent route	4	4	4	4,2
As a restaurant owner I should be able to see a 404 screen when I enter a non existent route	4	4	4	4,1
As a customer I should be able to see a 404 screen when I enter a non existent route	4	4	4,5	3,5
As a TuBocado manager I should be able to edit my personal information	5	5	13	12
Fix: login button	1	1	1	0,5
Fix: enter not work when login	4	4	6	6,5
Fix: Restaurant info details, improve placement of information	2	2	2	2,5
Fix: fonts	2	2	2	1,4
Fix: restaurant finances UI	3	2	3,1	2
Fix: improve image UI	3	2	3,2	2,5
Fix: Bug in the cart when shopping two similar items	**3**	**2**	**4**	2
Fix: Map	3	3	5	3,3
Fix: edit user TuBocado manager	2	2	2	2,1
Fix: view order on click breaks	4	4	4,6	4,1
Fix: update reviews withoouot refresh in all systems	4	4	4,3	4
Fix: match finance view layout to the rest ot the proyect	2	2	3	2
Sprint 12				
Fix: Errors in UI on the edit restaurant info page	5	4	10	6,2
Fix: Active restaurants	3	2	3,5	3
Fix: Date picker orders	**2**	**1**	**2**	1,4
Configure Sengrid enviroment prod	4	4	4,5	3,5
As a Customer I should be able to ask for my bocados in less time	5	5	12,3	15,4
As a Restaurant Owner I should be able to edit my personal information	4	5	9	14,6
Fix: restaurant profile error UI	2	2	2	2,4

Fix: admin modals layout	3	2	3,5	2
Fix: Improve password layout	1	1	1	0,5
Sprint 13				
TOTAL RELEASE 4	148	142	223,5	217,29

A.20. Resultado de *Sprint Plannings*

Sprint 1

Sprint 1	
Fecha Inicio	07/08/2023
Fecha Fin	21/08/2023
Objetivo	Setup de Backend y Frontend. Registro y autenticación de usuarios. Ver menús.
Esfuerzo estimado	-
Velocidad estimada	-

Sprint 2

Sprint 2	
Fecha Inicio	21/08/2023
Fecha Fin	03/09/2023
Objetivo	Generar link para compartir. Agregar ubicación. Ver Bocados. Login restaurantes. Manejo de pedidos. Gestion de niveles y valores de puntos.
Esfuerzo estimado	-
Velocidad estimada	-

Sprint 3

Sprint 3	
Fecha Inicio	04/09/2023
Fecha Fin	17/09/2023
Objetivo	Comprar sushi con puntos o efectivo. Editar el estado de pedidos en tiempo real. Búsqueda en la lista de restaurantes. Consultar bocados disponibles.
Esfuerzo estimado	-
Velocidad estimada	-

Sprint 4

Sprint 4	
Fecha Inicio	18/09/2023
Fecha Fin	25/09/2023
Objetivo	Deploy.
Esfuerzo estimado	-
Velocidad estimada	-

Sprint 5

Sprint 5	
Fecha Inicio	25/09/2023
Fecha Fin	09/10/2023
Objetivo	Cancelar y reembolsar órdenes. Solicitudes de restaurantes para administradores de TuBocado. Incorporación de un administrador para la aplicación.
Esfuerzo estimado	134,3 hrs
Velocidad estimada	74 Story Points

Integrante	**Capacidad por integrante (Hs)**
Valentina	40
Federico Torres	20
Federico Charbonnier	40
Jimena	40
Total:	140

Sprint 6

Sprint 6	
Fecha Inicio	09/10/2023
Fecha Fin	22/10/2023
Objetivo	Filtrar restaurantes por tiempo y ubicación. Envío de emails a restaurantes para notificar aceptación/echaz Creación, modificación y cancelación de categorías de platos Realizar pagos con tarjeta de crédito.
Esfuerzo estimado	177 hrs
Velocidad estimada	75 Story Points

Integrante	**Capacidad por integrante (Hs)**
Valentina	25
Federico Torres	35
Federico Charbonnier	35
Jimena	35
Total:	130

Sprint 7

Sprint 8

Sprint 7	
Fecha Inicio	23/10/2023
Fecha Fin	06/11/2023
Objetivo	Gestión del perfil y la información personal. Seguimiento de bocados generados y puntos acumulados. Visualización del estado de órdenes en la página de inicio. Generación de bocados al realizar un nuevo pedido. Deploy.
Esfuerzo estimado	134 hrs
Velocidad estimada	71 Story Points

Integrante	Capacidad por integrante (Hs)
Valentina	40
Federico Torres	30
Federico Charbonnier	30
Jimena	30
Total:	130

Sprint 8	
Fecha Inicio	07/11/2023
Fecha Fin	20/11/2023
Objetivo	Ver las ganancias generadas y los puntos canjeados. Gestión de clientes por parte de los administradores. Crear nuevos administradores de TuBocado. Escribir review de un pedido. Consultar historial de ordenes. Acceder a los reportes
Esfuerzo estimado	141,5 hrs
Velocidad estimada	69 Story Points

Integrante	Capacidad por integrante (Hs)
Valentina	35
Federico Torres	35
Federico Charbonnier	35
Jimena	35
Total:	140

Sprint 9

Sprint 9	
Fecha Inicio	21/11/2023
Fecha Fin	04/12/2023
Objetivo	Restaurantes ditar su información. Ver reviews de los clientes sobre tu restaurante. Visualización de la red de referidos para clientes. Generación de Bocados solo para clientes activos.
Esfuerzo estimado	109,9 hrs
Velocidad estimada	60 Story Points

Integrante	Capacidad por integrante (Hs)
Valentina	30
Federico Torres	30
Federico Charbonnier	30
Jimena	30
Total:	120

Sprint 10

Sprint 10	
Fecha Inicio	05/12/2023
Fecha Fin	19/12/2023
Objetivo	Restaurar contraseñas. Ver reviews de un restaurante. Pedir comida con opción Pickup. Cerrar un restaurante. Ver reportes de mi restaurante. Actualización real-time del estado de los pedidos. Deploy
Esfuerzo estimado	124 hrs
Velocidad estimada	70 Story Points

Integrante	Capacidad por integrante (Hs)
Valentina	30
Federico Torres	30
Federico Charbonnier	20
Jimena	40
Total:	120

Sprint 11

Sprint 11	
Fecha Inicio	19/12/2023
Fecha Fin	01/01/2024
Objetivo	incorporar notificaciones para restaurantes. Ver mapa. Gestionar ubicaciones. Correccion de bugs.
Esfuerzo estimado	102,3 hrs
Velocidad estimada	64 Story Points

Integrante	Capacidad por integrante (Hs)
Valentina	35
Federico Torres	25
Federico Charbonnier	25
Jimena	35
Total:	120

Sprint 12

Sprint 12	
Fecha Inicio	02/01/2024
Fecha Fin	16/01/2024
Objetivo	Ver arbol de referidos interactivo. Cambiar horarios de apertura y cierre segun el dia. Implementar un claro manejo de errores. Correccion de bugs.
Esfuerzo estimado	101,7 hrs
Velocidad estimada	65 Story Points

Integrante	Capacidad por integrante (Hs)
Valentina	20
Federico Torres	40
Federico Charbonnier	25
Jimena	35
Total:	120

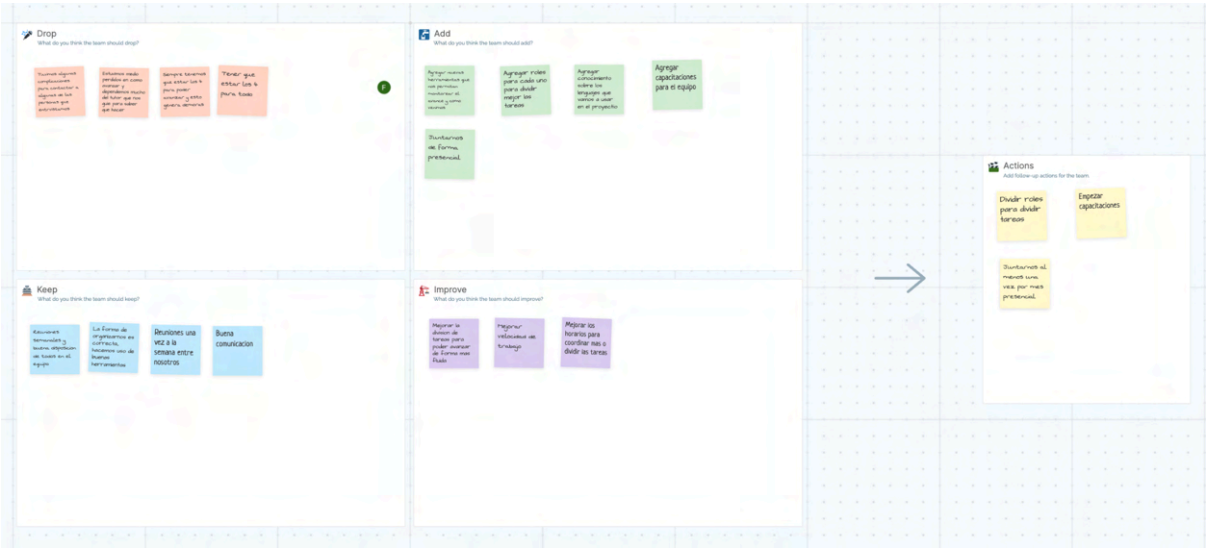
Sprint 13

Sprint 13	
Fecha Inicio	17/01/2024
Fecha Fin	31/01/2024
Objetivo	Correccion de bugs faltantes. Deploy
Esfuerzo estimado	47,8 hrs
Velocidad estimada	29 Story Points
Integrante	**Capacidad por integrante (Hs)**
Valentina	35
Federico Torres	35
Federico Charbonnier	35
Jimena	25
Total:	130

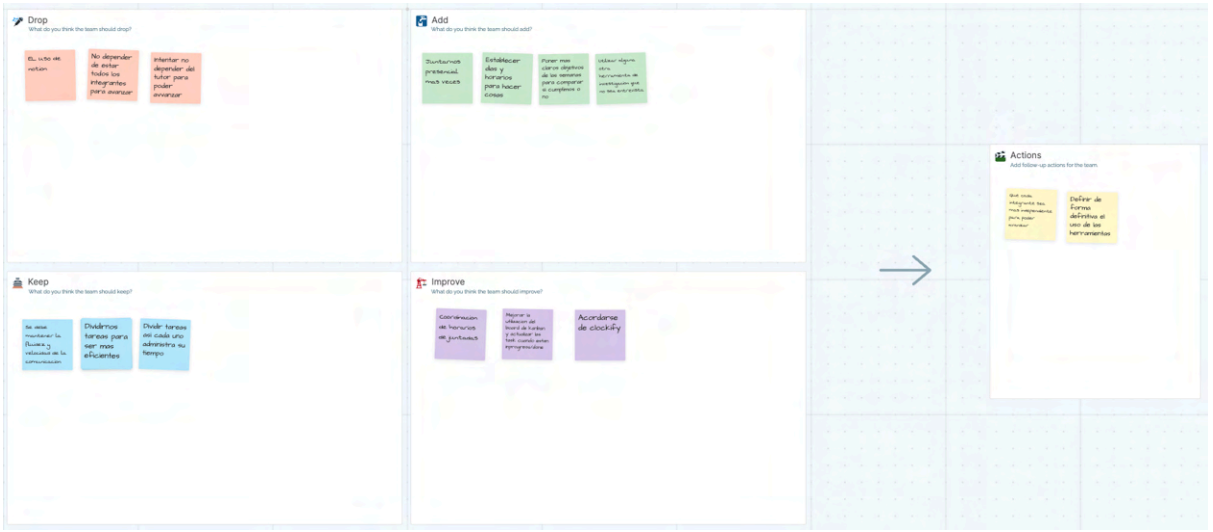
A.21. Retros

Retrospectivas Kanban

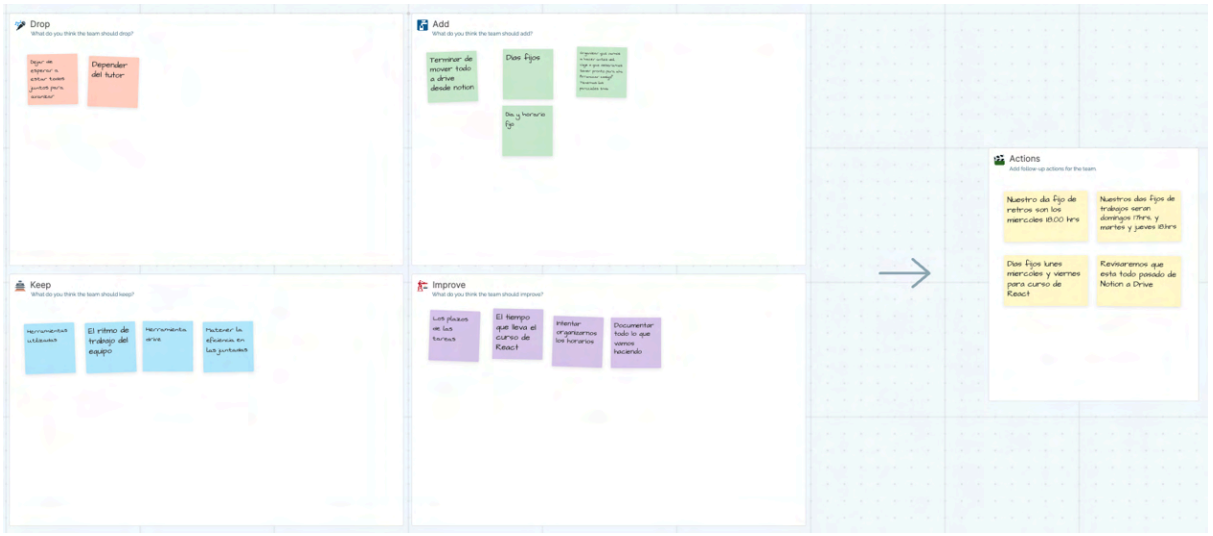
Retro 31/05/2023



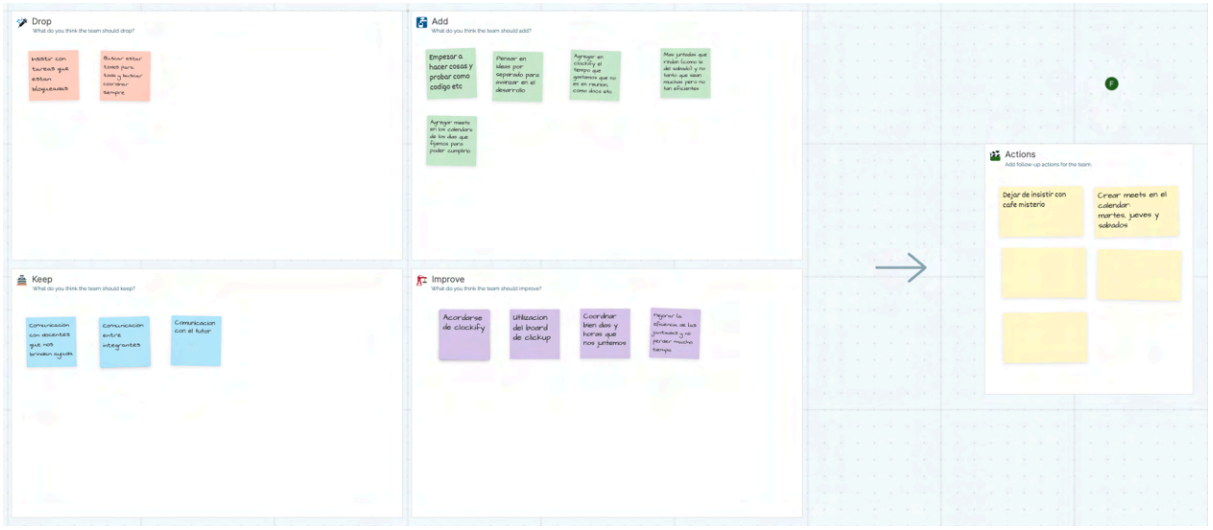
Retro 07/06/2023



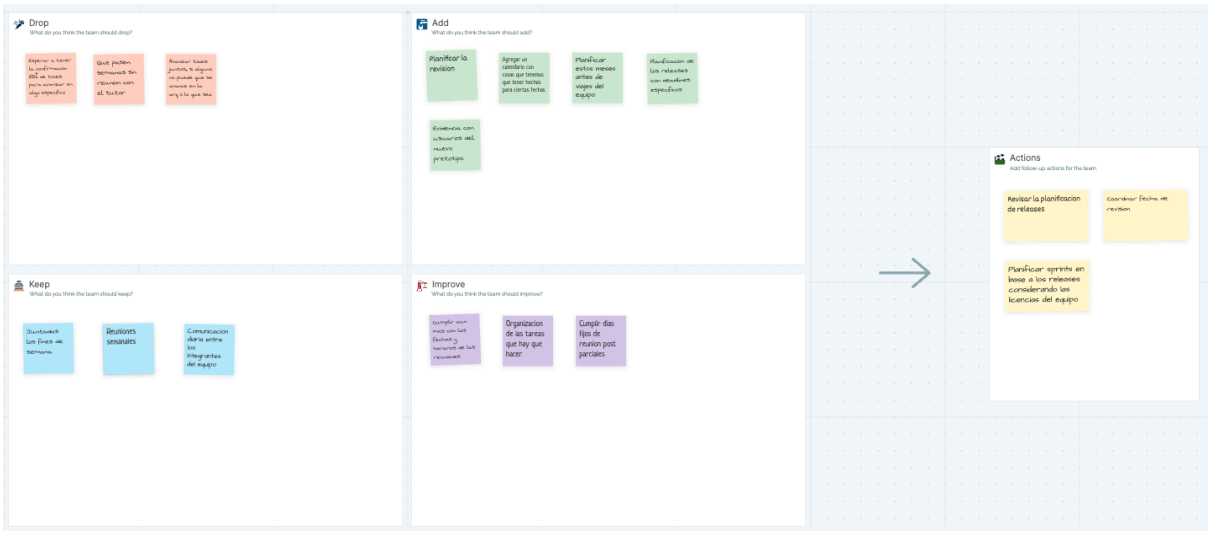
Retro 14/06/2023



Retro 21/06/2023

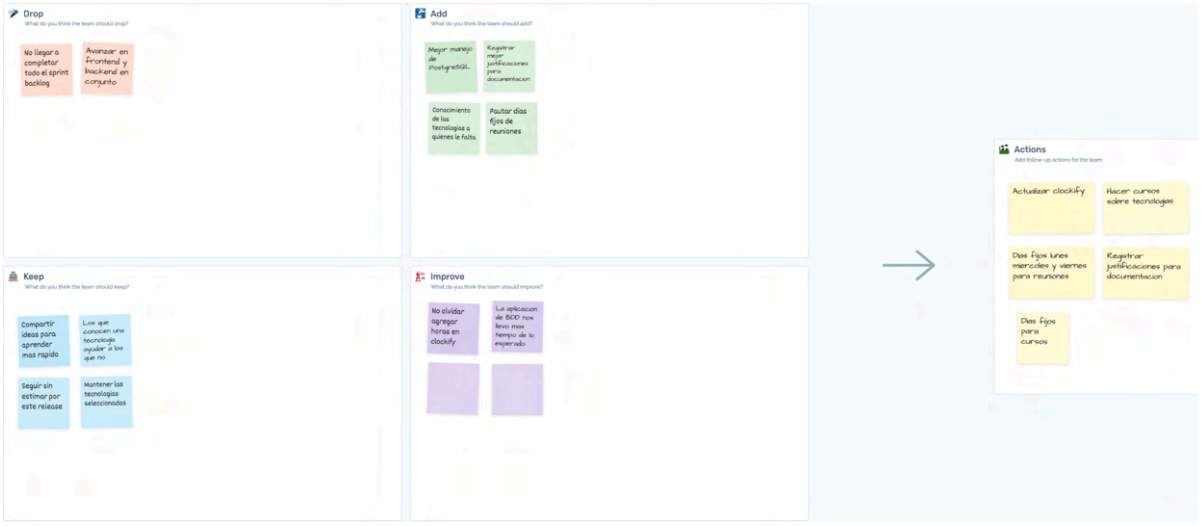


Retro 28/06/2023

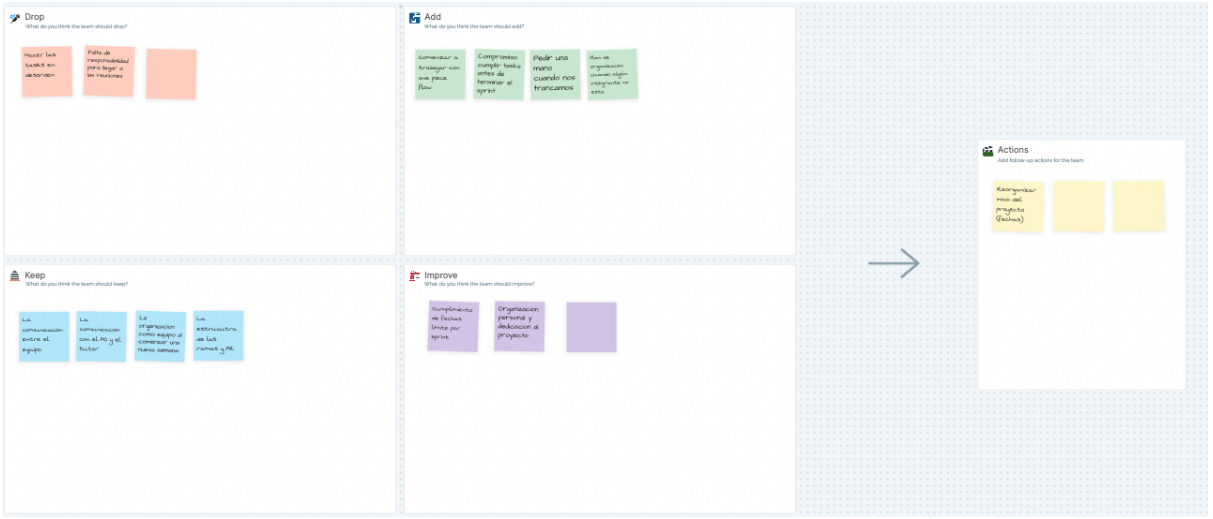


Retrospectivas SCRUM

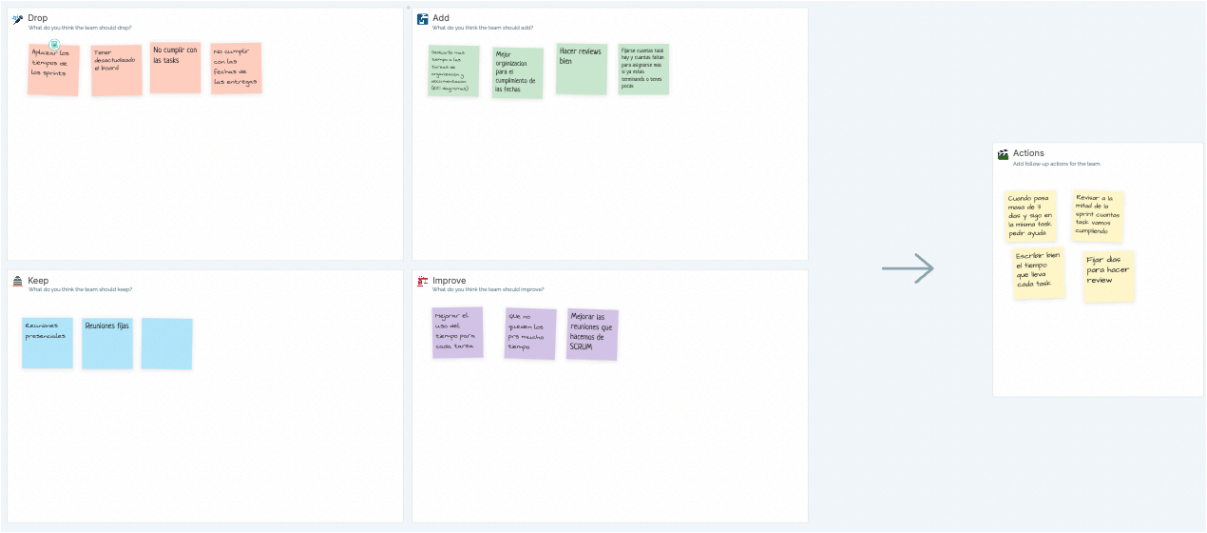
Sprint 1



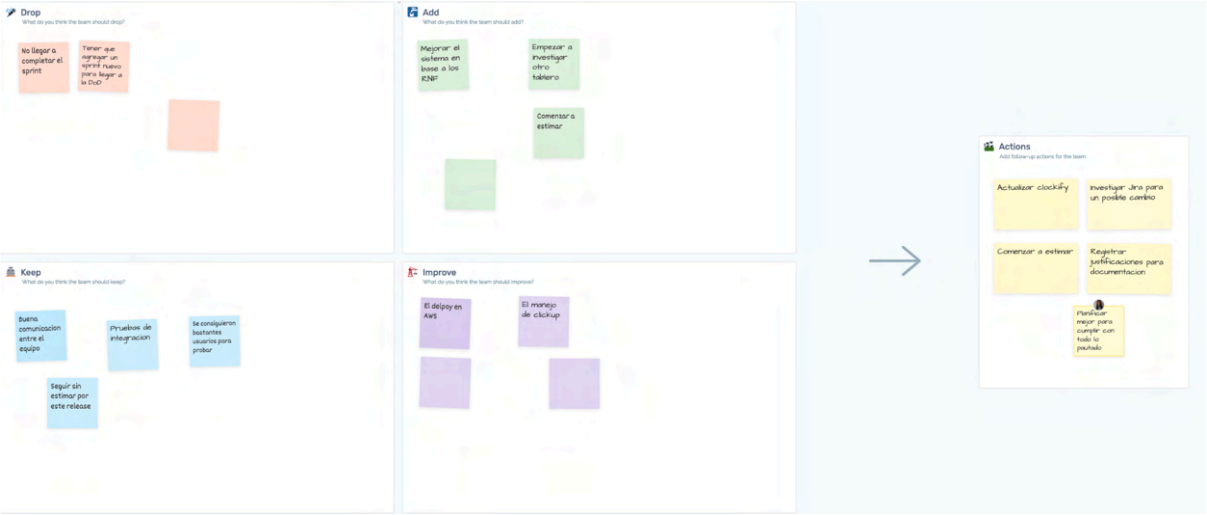
Sprint 2



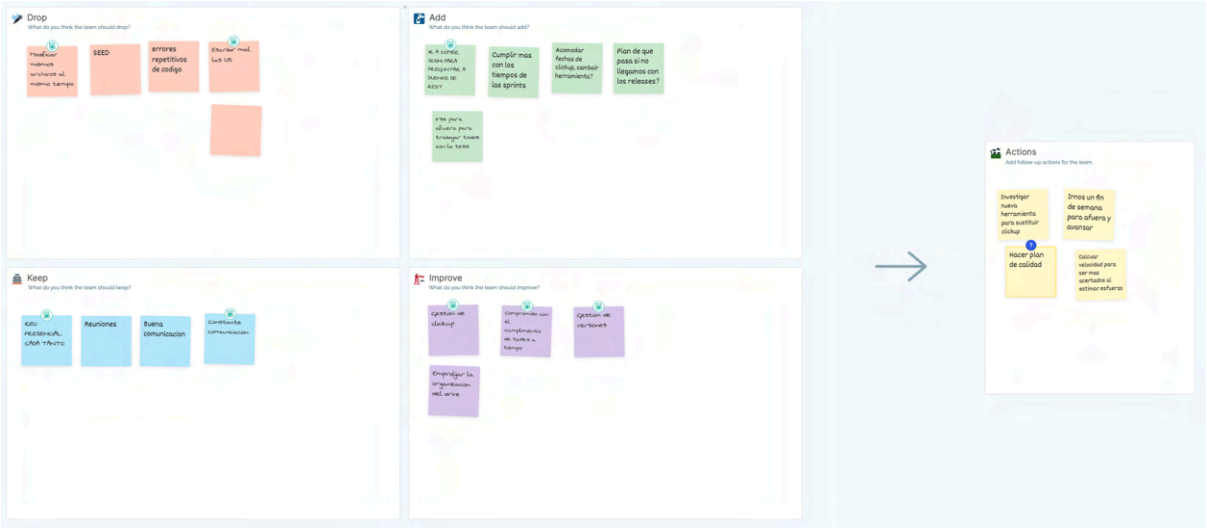
Sprint 3



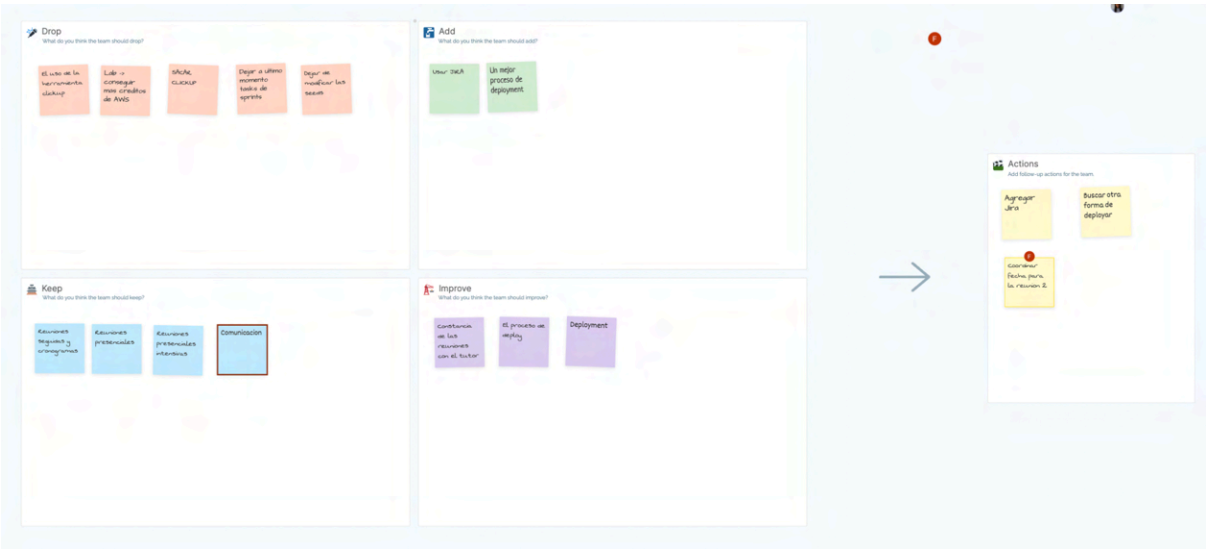
Sprint 4



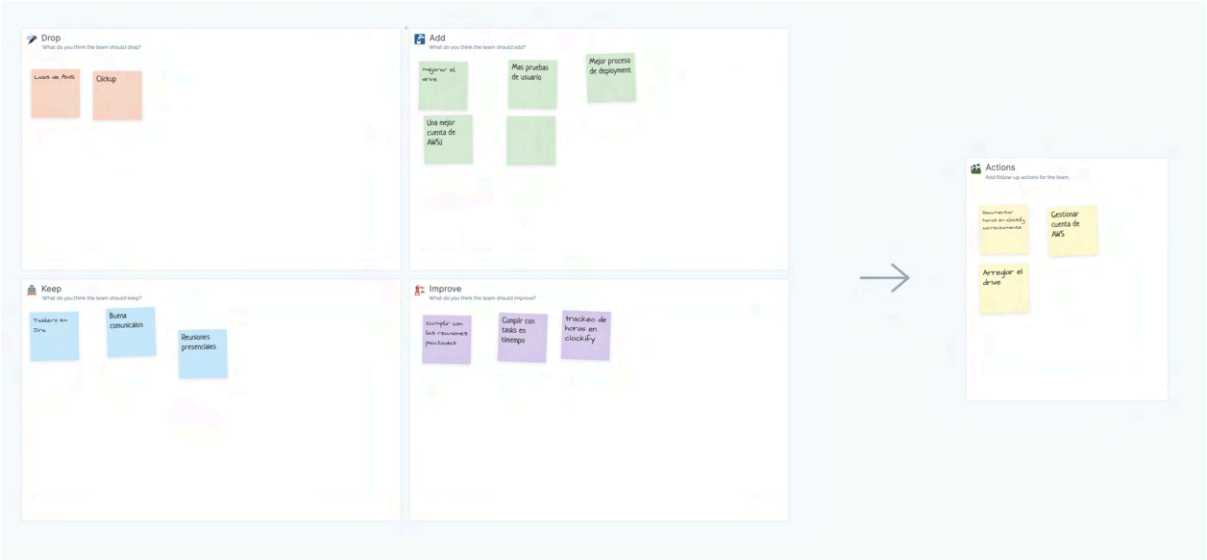
Sprint 5



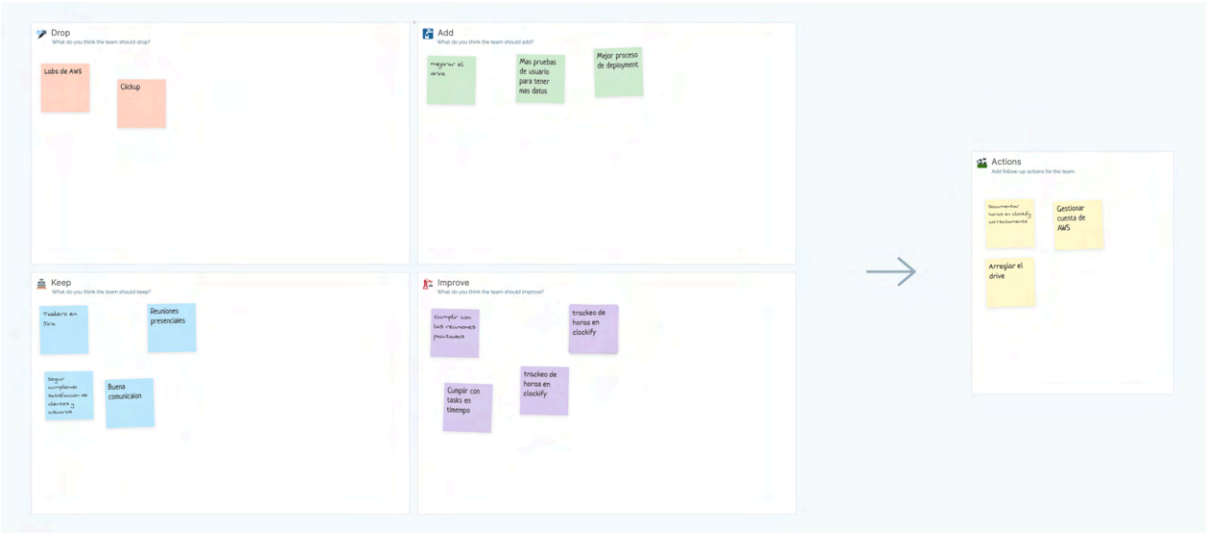
Sprint 6



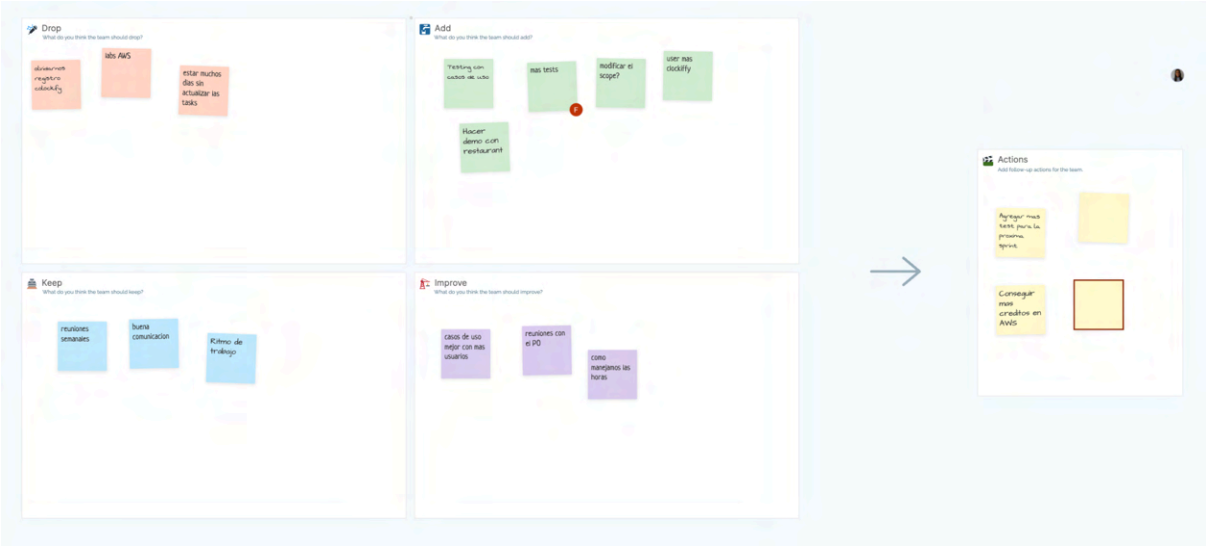
Sprint 7



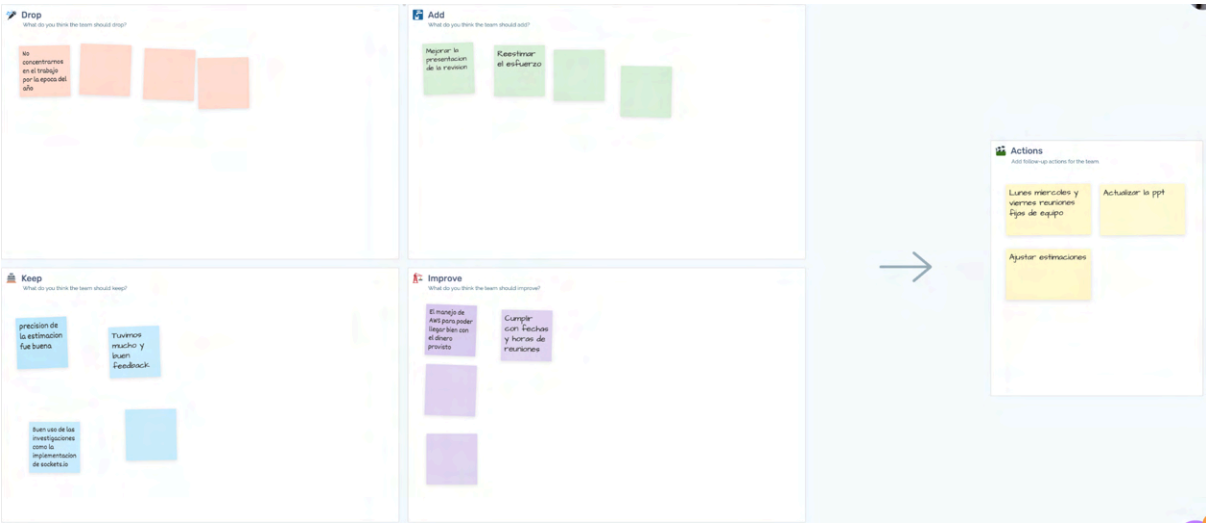
Sprint 8



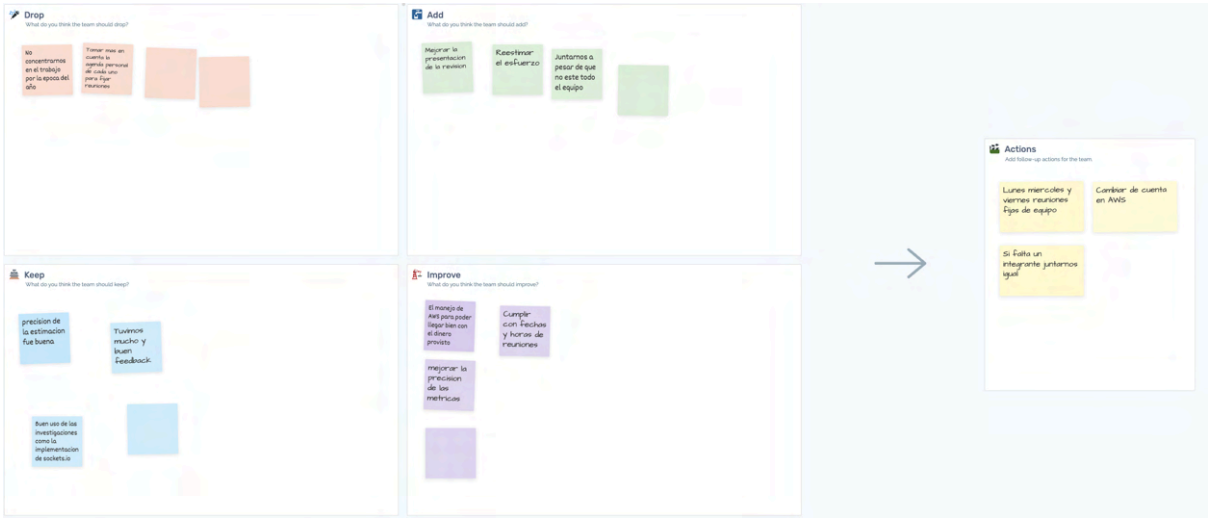
Sprint 9



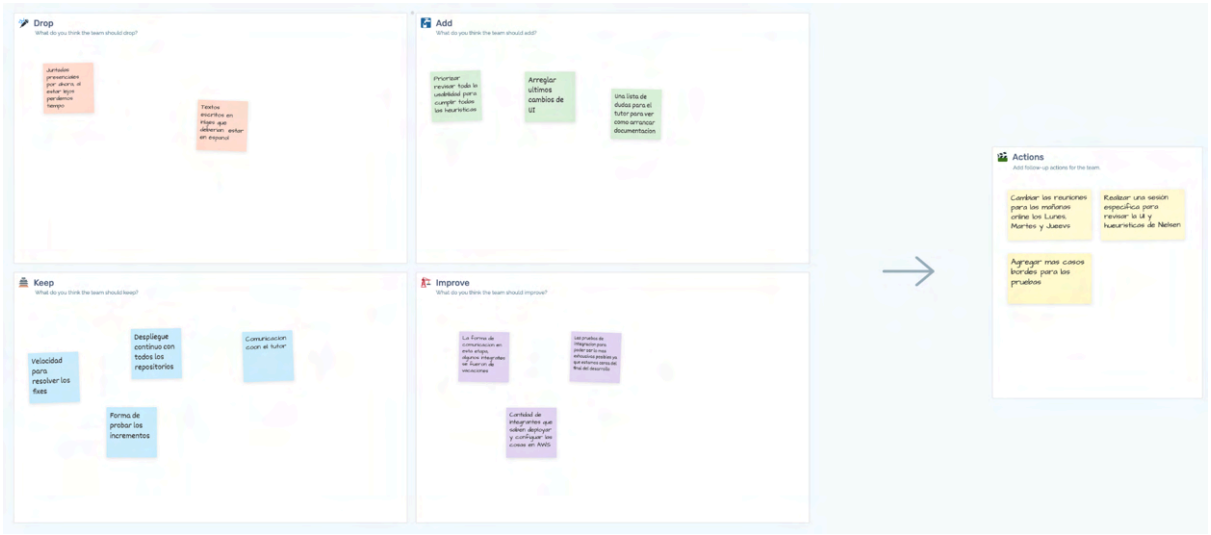
Sprint 10



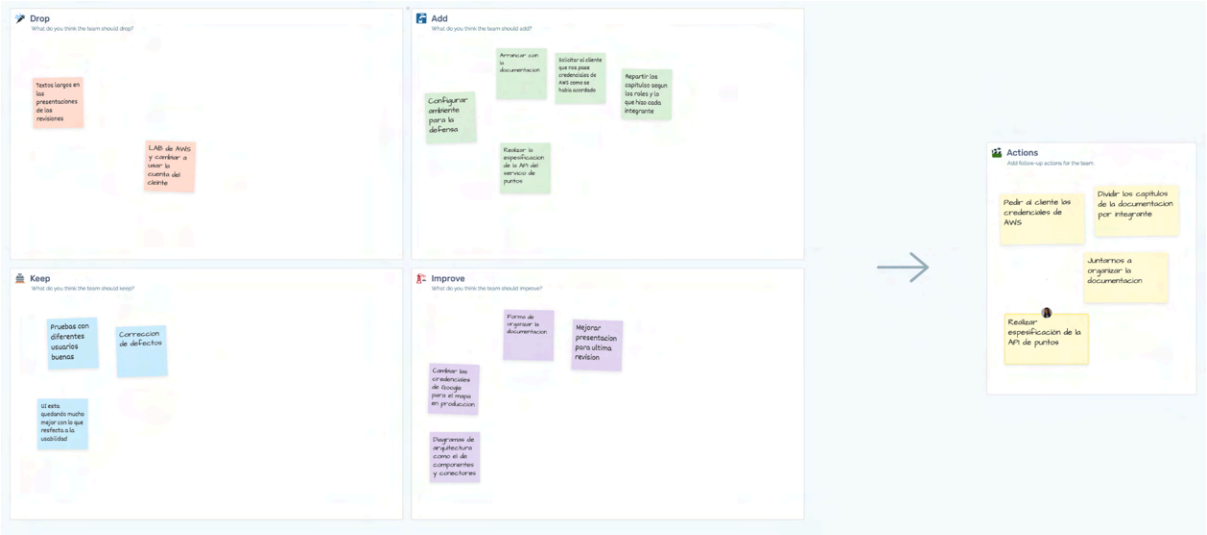
Sprint 11



Sprint 12



Sprint 13



A.22. Riesgos identificados por cada release y por cada *Sprint*

***RELEASE* 1**
Requisitos cambiantes
Falta de tiempo del equipo (estudio, trabajo y viajes)
Estimación inadecuada de esfuerzo y tamaño
Carencia de conocimiento técnico

***RELEASE* 2**
Diseño inadecuado de la arquitectura
Falta de experiencia en el diseño de conexión entre servicios
Requisitos cambiantes
Estimación inadecuada de esfuerzo y tamaño

***RELEASE* 3**
Requisitos cambiantes
Disponibilidad de agenda por parte del cliente
Falta de experiencia en el diseño de conexión entre servicios

***RELEASE* 4**
Falta de tiempo del equipo (estudio, trabajo y viajes)

Disponibilidad de agenda por parte del cliente

RELEASE 1			
***Sprint* 1**	***Sprint* 2**	***Sprint* 3**	***Sprint* 4**
Estimación inadecuada de esfuerzo y tamaño	Estimación inadecuada de esfuerzo y tamaño	Estimación inadecuada de esfuerzo y tamaño	Estimación inadecuada de esfuerzo y tamaño
Carencia de conocimiento técnico	Carencia de conocimiento técnico	Carencia de conocimiento técnico	Carencia de conocimiento técnico
		Requisitos cambiantes	Requisitos cambiantes
		Falta de tiempo del equipo (estudio, trabajo y viajes)	Falta de tiempo del equipo (estudio, trabajo y viajes)

RELEASE 2		
***Sprint* 5**	***Sprint* 6**	***Sprint* 7**
Diseño inadecuado de la arquitectura	Diseño inadecuado de la arquitectura	Diseño inadecuado de la arquitectura
Falta de experiencia en el diseño de conexión entre servicios	Falta de experiencia en el diseño de conexión entre servicios	Falta de experiencia en el diseño de conexión entre servicios
Estimación inadecuada de esfuerzo y tamaño	Requisitos cambiantes	Requisitos cambiantes
	Estimación inadecuada de esfuerzo y tamaño	

RELEASE 3		
***Sprint* 8**	***Sprint* 9**	***Sprint* 10**
Requisitos cambiantes	Requisitos cambiantes	Disponibilidad de agenda por parte del cliente
Falta de experiencia en el diseño de conexión entre servicios	Disponibilidad de agenda por parte del cliente	Falta de experiencia en el diseño de conexión entre servicios
	Falta de experiencia en el diseño de conexión entre servicios	

RELEASE 4		
***Sprint* 11**	***Sprint* 12**	***Sprint* 13**
Disponibilidad de agenda por	Falta de tiempo del equipo	Falta de tiempo del equipo

parte del cliente	(estudio, trabajo y viajes)	(estudio, trabajo y viajes)
	Disponibilidad de agenda por parte del cliente	Disponibilidad de agenda por parte del cliente

A.23. Riesgos identificados a lo largo del proyecto

ID	Título	Contexto	Acción Preventiva	Acción Correctiva
R1	Falta de tiempo del equipo (estudio, trabajo y viajes)	**Dado** que el equipo no pueda dedicar el suficiente tiempo y esfuerzo al proyecto debido a otras responsabilidades como estudio de otras materias, actividad laboral de cada uno y viajes al exterior. **Cuando** se desarrolla y ejecuta el proyecto. **Entonces** puede haber retrasos en la entrega del producto, disminución de la productividad y afectación de la calidad del producto final o puede que se deba recortar el alcance.	Establecer un plan de trabajo claro, asignar responsabilidades específicas y fomentar la comunicación abierta para mantener la motivación y el compromiso del equipo.	Afectar la calidad del producto o recortar el alcance en caso de que se detecte una falta de dedicación por parte del equipo.
R2	Estimación inadecuada de esfuerzo y tamaño	**Dado** que el equipo no tiene suficiente experiencia en la estimación precisa de la duración y el esfuerzo requerido para completar las tareas del proyecto. **Cuando** se realiza la estimación. **Entonces** puede haber desviaciones en los plazos, sobrecarga de trabajo, falta de recursos, recorte de alcance o incumplimiento de los objetivos establecidos.	Capacitar al equipo en técnicas de estimación y utilizar referencias previas para obtener estimaciones más precisas.	Reevaluar las estimaciones y reajustar el alcance en caso de que se identifiquen desviaciones significativas en el progreso del proyecto.
R3	Carencia de conocimiento técnico	**Dado** que el equipo no tiene suficiente familiaridad o experiencia con las tecnologías específicas necesarias para el desarrollo del proyecto. **Cuando** se ejecute la implementación y desarrollo de las funcionalidades. **Entonces** puede haber dificultades en la implementación correcta de las tecnologías, errores en el código, baja calidad del producto, retrasos en la entrega o resentimiento en el alcance.	Proporcionar capacitación y recursos de aprendizaje para que el equipo adquiera el conocimiento necesario de las tecnologías requeridas.	Buscar apoyo externo, como consultores o especialistas, para brindar orientación y asistencia en la aplicación de las tecnologías y corregir cualquier error o deficiencia identificada o recortar el alcance.
R4	Diseño inadecuado de la arquitectura	**Dado** que el equipo no cuenta con los conocimientos o la experiencia suficiente en diseño de arquitectura de *software* para el tipo de proyecto. **Cuando** se ejecuta el diseño y planificación de la arquitectura del sistema. **Entonces** puede haber problemas de rendimiento, escalabilidad, mantenibilidad o seguridad, entre otros, en el producto final.	Involucrar a arquitectos o expertos en diseño de *software* para guiar, apoyar al equipo en la definición de la arquitectura adecuada y también leer literatura que apoye este tipo de arquitectura. Además, realizar revisiones y análisis exhaustivos	Seguir adelante y aplicar parches o acortar el alcance.

			del diseño de la arquitectura, y realizar modificaciones o mejoras en caso de que se identifiquen deficiencias o problemas.	
R5	Disponibilidad de agenda por parte del cliente	**Dado** que el cliente no puede brindar la atención o el compromiso necesarios para el desarrollo del proyecto, como retrasos en la toma de decisiones, falta de disponibilidad para la revisión y retroalimentación de las entregas. **Cuando** a lo largo de la ejecución del proyecto. **Entonces** puede haber demoras en la obtención de información y validación de requisitos, cambios de alcance no gestionados adecuadamente o insatisfacción del cliente.	Establecer una comunicación clara con el cliente desde el inicio del proyecto, establecer expectativas y plazos realistas, y fomentar una colaboración cercana para mantener su compromiso	Revisar y ajustar los planes y cronogramas del proyecto para adaptarse a los retrasos o cambios en la disponibilidad del cliente, y comunicarse de manera efectiva para gestionar las expectativas y buscar soluciones conjuntas.
R6	Falta de experiencia en el diseño de conexión entre servicios	**Dado** que el equipo no posee dominio de las tácticas, técnicas y prácticas de las arquitecturas basadas en microsistemas. **Cuando** cuando llevamos a cabo el diseño de la arquitectura del producto. **Entonces** existe la posibilidad de que la arquitectura resultante sea inadecuada, lo que podría provocar retrasos en el cronograma debido a la necesidad de realizar modificaciones en el diseño, o comprometer la calidad del producto a construir.	Capacitar al equipo en tácticas, técnicas, prácticas y tecnologías de arquitecturas basadas en conexión entre servicios para adquirir el conocimiento y las habilidades necesarias.	Contratar expertos que realicen el diseño de la arquitectura y brinden supervisión al equipo en su aplicación, de manera que se cuente con la experiencia requerida.
R7	Requisitos cambiantes	**Dado** que el *product owner* puede cambiar de opinión acerca de lo que quiere sobre el producto porque puede no tener tan claro los requerimientos ya que depende de sus clientes/usuarios, como los dueños de restaurantes, consumidores finales, entre otros. **Cuando** durante la ejecución del proyecto. **Entonces** existe la posibilidad de que la entrega se retrase, o haya desmotivación del equipo debido a la reconfiguración continua del trabajo.	Definir y acordar claramente los requerimientos al inicio del proyecto, establecer un proceso formal de gestión de cambios e ir hablando continuamente con el cliente. Estar abiertos al cambio y pensar en una arquitectura de manera que sea adaptable y tener la opción de intercambiar requisitos.	En caso de cambios, realizar una evaluación del impacto y reajustar la planificación del proyecto o recortar el alcance.

A.24. Análisis de riesgos

ID	Impacto	Probabilidad	Magnitud
R2	2,1	0,65	1,365

R3	3,7	0,75	2,775
Exposición			**4,0**

Análisis de riesgos *Sprint* 1 (07/08/20239)

ID	Impacto	Probabilidad	Magnitud
R2	2,1	0,65	1,365
R3	3,7	0,65	2,405
Exposición			**3,7**

Análisis de riesgos *Sprint* 2 (21/08/20239)

ID	Impacto	Probabilidad	Magnitud
R1	4,2	0,5	2,1
R2	1,9	0,7	1,33
R3	3,9	0,65	2,535
R7	3,6	0,55	1,98
Exposición			**7,9**

Análisis de riesgos *Sprint* 3 (04/09/2023)

ID	Impacto	Probabilidad	Magnitud
R1	4,2	0,4	1,7
R2	1,9	0,7	1,33
R3	3,9	0,65	2,535
R7	3,6	0,55	1,98
Exposición			**7,6**

Análisis de riesgos *Sprint* 4 (18/09/2023)

ID	Impacto	Probabilidad	Magnitud
R2	1,9	0,7	1,33
R4	3,5	0,6	2,1
R6	3,1	0,4	1,24
Exposición			**4,67**

Análisis de riesgos *Sprint* 5 (25/09/2023)

ID	Impacto	Probabilidad	Magnitud
R2	1,9	0,7	1,33
R4	3,5	0,4	1,4

R6	3	0,4	1,2
R7	2,5	0,5	1,25
Exposición			**5,18**

Análisis de riesgos *Sprint* 6 (09/10/2023)

ID	Impacto	Probabilidad	Magnitud
R4	3,5	0,4	1,4
R6	3	0,4	1,2
R7	2,5	0,5	1,25
Exposición			**3,85**

Análisis de riesgos *Sprint* 7 (23/10/2023)

ID	Impacto	Probabilidad	Magnitud
R6	3	0,4	1,2
R7	2,5	0,6	1,5
Exposición			**2,7**

Análisis de riesgos *Sprint* 8 (07/11/2023)

ID	Impacto	Probabilidad	Magnitud
R5	1	0,6	0,6
R6	3	0,4	1,2
R7	2,5	0,6	1,5
Exposición			**3,3**

Análisis de riesgos *Sprint* 9 (21/11/2023)

ID	Impacto	Probabilidad	Magnitud
R4	3,5	0,4	1,4
R5	1	0,6	0,6
Exposición			**2**

Análisis de riesgos *Sprint* 10 (05/12/2023)

ID	Impacto	Probabilidad	Magnitud
R5	2	0,6	1,2
Exposición			**1,2**

Análisis de riesgos *Sprint* 11 (19/12/2023)

ID	Impacto	Probabilidad	Magnitud
R1	4,5	0,5	2,25
R5	2	0,6	1,2
Exposición			**3,45**

Análisis de riesgos *Sprint* 12 (02/01/2024)

ID	Impacto	Probabilidad	Magnitud
R1	4,5	0,3	1,35
R5	2	0,6	1,2
Exposición			**2,55**

Análisis de riesgos *Sprint* 13 (17/01/2024)

A.25. Evidencia de aplicación de estándares de calidad

Utilizar “let” y “const” en lugar de “var”

```
const INITAL_POINTS = 0;
```

Evidencia de “const”

```
const [open, setIsOpen] = useState<boolean>(false);

const { data: restaurant, isLoading } = useRestaurantByIdQuery({ restaurantId });
```

Evidencia de “const” en variables

Nombres de variables en camelCase

```
const queryParameters = new URLSearchParams(window.location.search);
const restaurantId = queryParameters.get("id");
```

Evidencia de nombres de variables con camelCase

```
const windowWidth = useWindowWidth();
const isSmallScreen = windowWidth <= M;
```

Evidencia de nombres de variables con camelCase

```
getUserInfo = async (user: User): Promise<UserInfo> => {

    const locations = await this.repos.users.getShippingAddress(us
    const { points } = await this.repos.users.getUserPoints(user.u
    const referrals = await this.repos.users.getUserReferrals(user

    const safeUser = {
```

Evidencia de nombres de variables con camelCase

Nombres de funciones en camelCase

```
export function getStatusName(status: Status): string {
  switch (status) {
    case Status.Pending:
      return PENDING;
    case Status.Accepted:
      return ACCEPTED;
    case Status.InTransit:
      return IN_TRANSIT;
    case Status.Delivered:
      return DELIVERED;
    case DECLINED:
      return REJECTED;
    case Status.Canceled:
      return CANCELED;
    default:
      return UNKNOWN;
  }
}
```

Evidencia de nombres de funciones con camelCase

```
const submitForm = (values: SignupFormType): void => {
  const { email, name, phone, referredBy, bday, password } = values;
  signup({ email, name, phone, password, bday, referredBy, points: INITAL_POINTS });
}
```

Evidencia de nombres de funciones con camelCase

Nombres de enumeraciones en PascalCase

```
export enum Status {
  Pending = 1,
  Accepted,
  InTransit,
  Delivered,
  Declined,
  Canceled,
}
```

Evidencia de nombres de enum con PascalCase

Nombre de constantes usar CONSTANT_CASE

```
export const DELIVERED = 'delivered';
export const DONE = 'done';
export const DELIVERY = 'Envío'
export const DIRECTION = 'Dirección';

export const EDIT = 'Editar';
export const EDIT_YOUR_INFO = 'Edita tu información personal.';
export const ERROR_TRY_AGAIN_LATER = "Error al realizar el pago, por favor inténtalo nuevamente
export const EMAIL = 'Email';
export const EMAIL_KEY = 'email';
export const ESTIMATED_DELIVERY_TIME = "Hora estimada de entrega"
export const EXPLORE_RESTAURANTS = "Explorar restaurantes";
export const EXPLORE_RESTAURANTS_KEY = "explore_restaurants";

export const FORGOT_PASSWORD = '¿Te olvidaste tu contraseña?'

export const GO_TO_SIGNUP = 'No tienes cuenta? Registrate aquí';

export const ORDER = 'Pedir';

export const INVALID_CREDENTIALS = 'Credenciales inválidas';
export const INVIATION_REQUIRED = 'Para poder registrarte pídele a algun amigo que te mande el
export const INVITE_NEW_FRIENDS = "Invitar amigos";
export const INVITE_NEW_KEY = "invite_friends";
export const IN_PROGRESS = 'in_progress';
```

Evidencia de constantes con CONSTANT_CASE

Nombres de tipos e interfaces con PascalCase

```
declare global {
  interface FormInputType {
    key: string;
    name: string;
    width?: string;
    isPassword?: boolean;
    isFullWidth?: boolean;
  }
}

type SignupFormType = {
  email: string;
  name: string;
  phone: string;
  referredBy: string;
  bday: string;
  password: string
}
```

Evidencia de tipos e interfaces con PascalCase

```
export interface Tree {
  name: string;
  children: Tree[]
}

export type CurrentUser = {
  userId: string;
  name: string;
  email: string;
  avatar: string;
  bday: string;
  phone: string;
  referredBy: string;
  points?: number;
  locations?: UserLocation[];
  shareLink: string;
  referrals: { tree: Tree; amount: number, levels: number }
}
```

Evidencia de tipos e interfaces con PascalCase

```ts
export type ReferralRelations = {
  referentId: string;
  referredId: string;
  levelDifference: number;
};
```

Evidencia de tipos e interfaces con PascalCase

No utilizar tipo: any

```ts
export const encryptPassword = (password: string): Promise<string> => bcrypt.hash(password, salt);
```

Evidencia de utilización de tipos

```ts
return useMutation(
  async (params: UserParams) => {
    const { data } = await api.post<SignUpResponse>(ROUTES.auth, params);
    return data;
  },
  {
    onSuccess(data) {
      localStorage.setItem('accessToken', data.token);
      navigate(ROUTES.home);
    },
  },
);
```

Evidencia de utilización de tipos

```ts
return useQuery(
  `${ordersKey}-${userId}`,
  async () => {
    const { data } = await api.get<{ orders: OrderResponse[] }>(`${ROUTES.orders}/${userId}?step=7`);
    return data?.orders;
  },
  {
    cacheTime: Infinity,
    staleTime: Infinity,
    retry: false,
    enabled: !!userId
  }
)
```

Evidencia de utilización de tipos

Definir tipos de interfaces y tipos de retorno para funciones

```
export class GetUserInfo implements UserInfoInterface {

    repos: IRepos;
    constructor({ repos }: { repos: IRepos }) {
        this.repos = repos;
    }
```

Evidencia de utilización de tipos e interfaces

```
export const encryptPassword = (password: string): Promise<string> => bcrypt.hash(password, salt);
```

Evidencia de utilización de tipos e interfaces

```
getUserInfo = async (user: User): Promise<UserInfo> => {
```

Evidencia de utilización de tipos e interfaces

Usar *undefined* en lugar de *null*

```
const [search, setSearch] = useState<string | undefined>(undefined);
const [selectedFilter, setSelectedFilter] = useState<string>(undefined);
const { data: pendingOrders, refetch } = usePendingOrdersQuery({ user_id: user.user_id });
const [currentLocation, setCurrentLocation] = useState<UserLocation>();
```

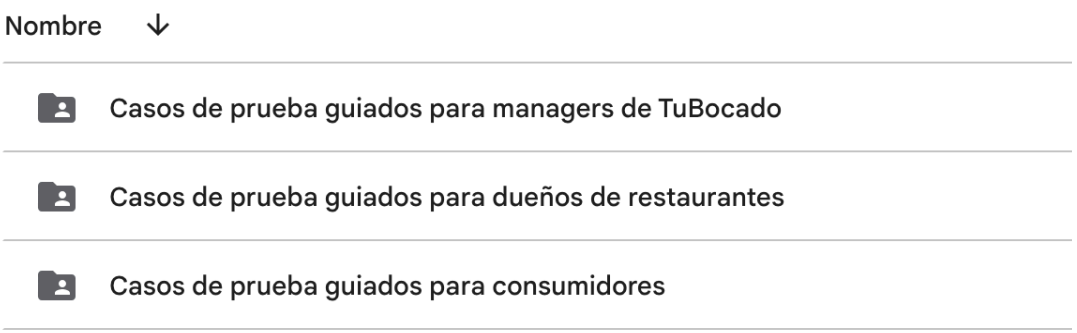
Evidencia de utilización de *undefined* en lugar de *null*

A.26. Evidencia pruebas de aceptación con usuarios

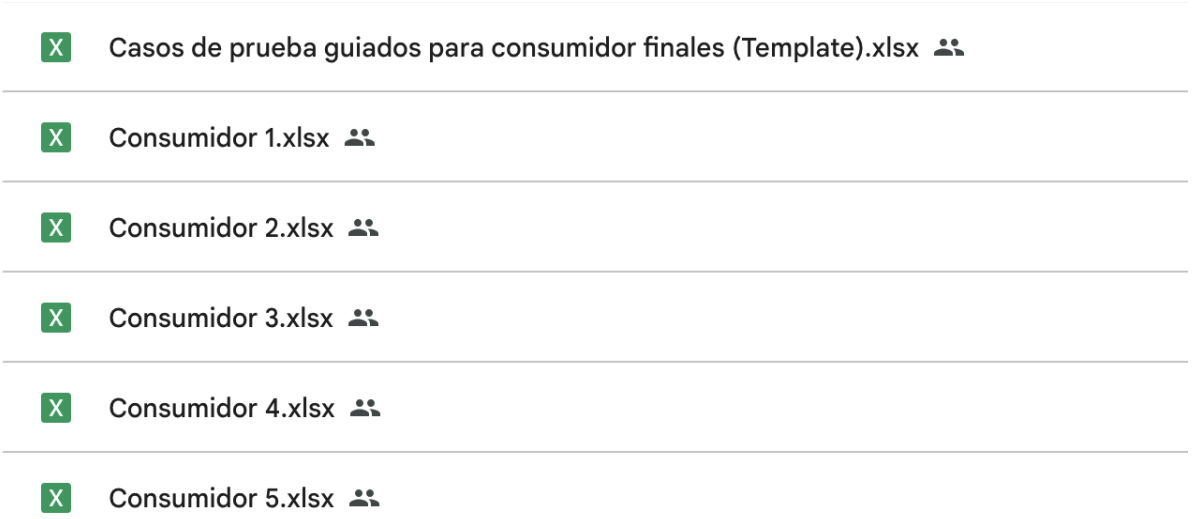
Para cada *release* se organizaban carpetas por tipos de usuario y allí dentro se guardaba cada *template* del usuario correspondiente con los resultados de sus pruebas.



Estructuración de documentación de pruebas de aceptación por release



Estructuración de documentación de pruebas de aceptación dentro de cada release



Estructuración de documentación de pruebas de aceptación dentro de rol

Para construir las pruebas guiadas para usuarios finales, se realizó una hoja de cálculo por cada rol. El equipo siguió un template brindado por el tutor en el cual incluimos las columnas “Historias de usuario”, “Criterios de aceptación”, “Entradas”, “Resultado esperado”, “Resultado obtenido”, “Comentarios”, “Capturas de pantalla”, “Horas estimadas”. A continuación se muestra cómo se estructuran las pruebas en el release 1 y un ejemplo de resultado.

Tempale Pruebas guionadas Release 1 - Consumidores

A continuación, se muestran únicamente las columnas escritas en el template para una mejor lectura.

Página con un enlace compartido y el número de Bocados	Ver con éxito el número total de Bocados como usuario registrado	Apretar la opcion "bocados" en la parte de arriba de la página	Ver la pantalla con datos de los bocados.
	Ver con éxito los bocados de un usuario	Una vez ingresado ver la seccion donde muestran los bocados.	Ver el numero de bocados del usuario.
	Ver con éxito el enlace compartido en la página TuBocado	Una vez ingresado seleccionar para copiar el enlace de invitación	Ver un enlace compartido que me permite compartir la página TuBocado.
	Ver con éxito la página TuBocado a través del medio elegido	Probar el enlace copiado	El enlace a la página TuBocado debe estar incrustado correctamente
Navegar usando el encabezado en la pantalla de inicio	Ver con éxito el encabezado en la pantalla de inicio	-	Ver un encabezado en la parte superior de la página
	Ver con éxito el logo	-	Ver el logo del sitio web en la esquina izquierda del encabezado
	Ver con éxito los enlaces de navegación principales	Apretar los diferentes botones para la navegación	Debería ver los enlaces principales de navegación y poder navegar entre ellos
	Ver con éxito el acceso a la cuenta de usuario	-	Ver opciones para acceder a la información de mi cuenta
Ver restaurantes cercanos	Ver con éxito los restaurantes activos segun ubicacion	Seleccionar ubicacion de entrega	Ver restaurantes activos
	Ver con éxito el campo de entrada de ubicación o la visibilidad de la opción en la página de ubicación	Entrar a la pagina de ubicación y seleccionar una	Ver un campo u opción para ingresar o seleccionar mi ubicación.
	Recordar con éxito la última ubicación ingresada por el usuario	-	Ver la última ubicación guardada
	Cambiar con éxito la ubicación desde la página de lista de restaurantes	-	Opción fácilmente accesible para actualizar mi ubicación

Seleccionar un método de pago proporcionado por el sistema	Selección exitosa del método de pago	Ingresar puntos o efectivo	Seleccionado correctamente el método de pago deseado
Comprar sushi con puntos o efectivo	Selección exitosa de puntos o efectivo para la compra		Hacer con éxito el pedido
	Orden no exitosa, puntos insuficientes	Se eleccionan artículos con mas bocados de los que tiene el usuario	Mensaje de error que indica puntos insuficientes
Perfil y menú del restaurante	Visualización exitosa de la información básica del restaurante	Hacer clic en un restaurante	Ver elementos como el nombre del restaurante, la ubicación y una breve descripción
	Exitosa descubrimiento de perfiles de restaurantes		Redirigido a la página de perfil de ese restaurante
	Acceso exitoso al menú del restaurante		Redirigido a una página que muestra los elementos del menú del restaurante
	Ver con éxito la vista detallada de los elementos del menú		Ver detalles como el nombre, la descripción y el precio de cada elemento
Ver la lista de restaurantes de sushi y seleccionar uno	Ver con éxito la lista de restaurantes y hacer clic en uno		Ver la página del restaurante seleccionado

Estructuración de *template* del usuario consumidor para el *release* 1

Resultados Prueba Aceptación Release 1 - Consumidor 1

Historia de usuario	Casos de prueba guiados para usuarios finales Criterios de aceptación	Entradas	Resultado esperado	Resultado obtenido	Comentarios	Capturas de pantalla	Estimated test time (in hours)
Registro de enlace	Éxito al recibir un enlace de registro especializado	Ingresar a "https://develop.d35l02a6sd2hn.amplifyapp.com/signup?invitation_code=9843d4c9c-a1ae-4f71-91ff-021c60dd3456" para poder realizar el registro	Al hacer clic en el enlace se debe dirigir a una página de registro con el código de invitación ingresado	No Cumplido	Primera vez da página no encontrada		0,40
	Registro exitoso a través de un enlace especializado	Ingreso de todos los campos requeridos	Recibir un mensaje de confirmación y ser redirigido a una página de inicio	Cumplido	Devuelve página no encontrada		
	Datos invalidos	Algún campo vacio	Mensaje de error que refleja que los campos son requeridos	Cumplido			
		Email mal formado	Mensaje de error que refleja que el email está mal formado	Cumplido			
		Código de invitación faltante	Mostrar pantalla que indique que el codigo de invitacion es requerido	Cumplido			
	Registro no exitoso	Email existente	Mostrar mensaje de error de email ya existente	Cumplido			
		Código de invitación inválido. Escribir "https://develop.d35l02a6sd2hn.amplifyapp.com/signup?invitation_code=9843d4c9c-a1ae-4f71-91f"	Mensaje de error que indica que el codigo de invitación es invalido	Cumplido	Muy lindo		
Inicio de sesión	Inicio de sesión exitoso con credenciales correctas	Ingresar con usuario "user@test.com"	Iniciado sesión y redirigido a un panel personalizado o página de inicio.	Cumplido			0,10
	Inicio de sesión no exitoso con credenciales incorrectas	Ingresar credenciales: "Something123!"	Mensaje de error indicando que las credenciales son incorrectas	Cumplido			
		Ingresar contraseña inválida: "test123"	Mensaje de error indicando que las credenciales son inválidas	Cumplido			
		Ingresar email invalido	Mensaje de error indicando que las credenciales son inválidas	Cumplido			
Cerrar sesión	Funcionalidad de cierre de sesión exitosa	-	Sesión cerrada y redirigido a la página de inicio.	Cumplido			0,05
Página con un enlace compartido y el numero de bocados	Ver con éxito el número total de Bocados como usuario registrado	Apretar la opción "bocados" en la parte de arriba de la página	Ver la pantalla con datos de los bocados.	Cumplido			
	Ver con éxito los bocados de un usuario	Una vez ingresado ver la sección donde muestran los bocados	Ver el numero de bocados del usuario.	Cumplido			
	Ver con éxito el enlace compartido en la página TuBocado	Una vez ingresado seleccionar para copiar el enlace de invitación	Ver un enlace compartido que me permite compartir la página TuBocado.	Cumplido			
	Ver con éxito la página TuBocado a través del medio elegido	Probar el enlace copiado	El enlace a la página TuBocado debe estar incrustado correctamente	Cumplido			
Navegar usando el encabezado en la pantalla de inicio	Ver con éxito el encabezado en la pantalla de inicio	-	Ver un encabezado en la parte superior de la página	Cumplido			
	Ver con éxito el logo	-	Ver el logo del sitio web en la esquina izquierda del encabezado	Cumplido			
	Ver con éxito los enlaces de navegación principales	Apretar los diferentes botones para la navegación	Debería ver los enlaces principales de navegación y poder	Cumplido			
	Ver con éxito el acceso a la cuenta de usuario	-	Ver opciones para acceder a la información de mi cuenta	Cumplido			
Ver restaurantes cercanos	Ver con éxito los restaurantes activos según ubicación	Seleccionar ubicación de entrega	Ver restaurantes activos	Cumplido			0,10
	Ver con éxito el campo de entrada de ubicación o la visibilidad	Entrar a la página de ubicación y seleccionar una	Ver un campo u opción para ingresar o seleccionar mi	Cumplido			
	Recordar con éxito la última ubicación ingresada por el usuario	-	Ver la última ubicación guardada	Cumplido			
	Cambiar con éxito la ubicación desde la página de lista de	-	Opción fácilmente accesible para actualizar mi ubicación	Cumplido			
Seleccionar un método de pago	Selección exitosa del método de pago	Ingresar puntos o efectivo	Seleccionado correctamente el método de pago deseado	Cumplido			0,40
Comprar sushi con puntos o efectivo	Selección exitosa de puntos o efectivo para la compra	-	Hacer con éxito el pedido	Cumplido			
Perfil y menú del restaurante	Orden no exitosa, puntos insuficientes	Se eligieron artículos con mas bocados de los que	Mensaje de error que indica puntos insuficientes	Cumplido			
	Visualización exitosa de la información básica del restaurante	Hacer clic en un restaurante	Ver elementos como el nombre del restaurante, la	Cumplido			0,10
	Éxitosa descubrimiento de perfiles de restaurantes	-	Redirigido a la página de perfil de ese restaurante	Cumplido			
	Acceso exitoso al menú del restaurante	-	Redirigido a una página que muestra los elementos del	Cumplido			
	Ver con éxito la vista detallada de los elementos del menú	-	Ver detalles como el nombre, la descripción y el precio de	Cumplido			
Ver la lista de restaurantes de sushi y seleccionar uno	Ver con éxito la lista de restaurantes y hacer clic en uno	-	Ver la página del restaurante seleccionado	Cumplido	Tuve que poner click dos veces		0,10
Totales							**1,75**

Resultados prueba de aceptación de un consumidor para el *release* 1

Template Pruebas guionadas Release 1 - Dueños de restaurantes

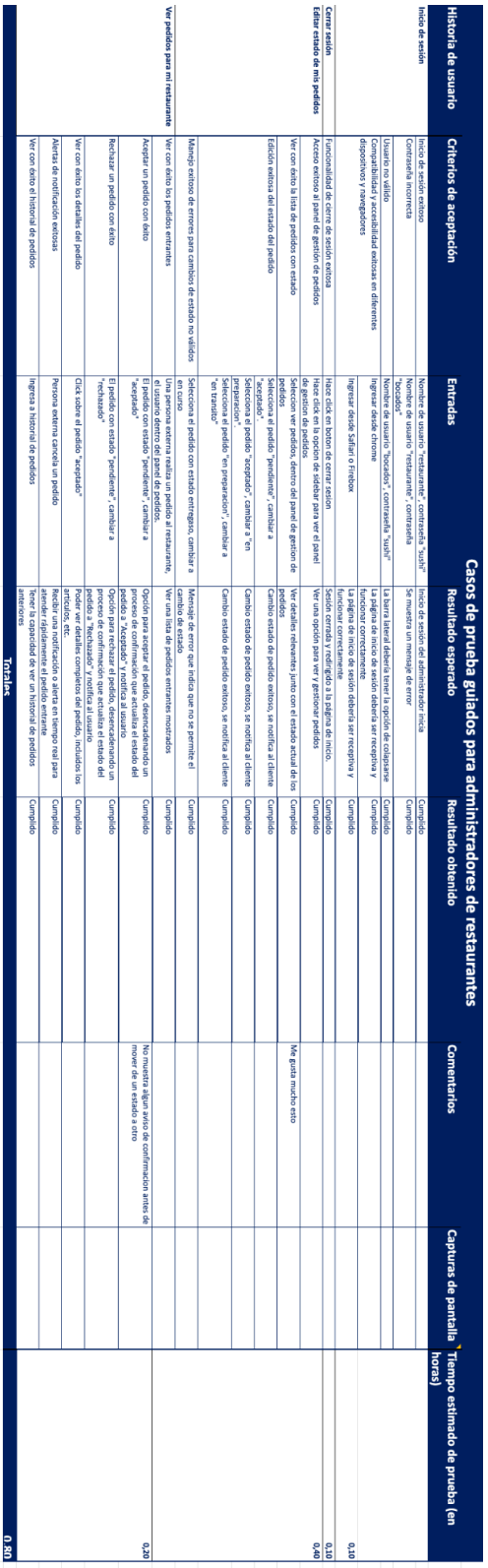
Historia de usuario	Criterios de aceptación	Entradas	Resultado esperado
Inicio de sesión	Inicio de sesión exitoso	Nombre de usuario "restaurante", contraseña "sushi"	Inicio de sesión del administrador inicia
	Contraseña incorrecta	Nombre de usuario "restaurante", contraseña "bocados"	Se muestra un mensaje de error
	Usuario no válido	Nombre de usuario "bocados", contraseña "sushi"	La barra lateral debería tener la opción de colapsarse
	Compatibilidad y accesibilidad exitosas en diferentes dispositivos y navegadores	Ingresar desde chrome	La página de inicio de sesión debería ser receptiva y funcionar correctamente
		Ingresar desde Safiari o Firebox	La página de inicio de sesión debería ser receptiva y funcionar correctamente
Cerrar sesión	Funcionalidad de cierre de sesión exitosa	Hace click en boton de cerrar sesion	Sesión cerrada y redirigido a la página de inicio.
Editar estado de mis pedidos	Acceso exitoso al panel de gestión de pedidos	Hace click en la opcion de sidebar para ver el panel de gestion de pedidos	Ver una opción para ver y gestionar pedidos
	Ver con éxito la lista de pedidos con estado	Seleccion ver pedidos, dentro del panel de gestion de pedidos	Ver detalles relevantes junto con el estado actual de los pedidos
	Edición exitosa del estado del pedido	Selecciona el pedido "pendiente", cambiar a "aceptado".	Cambio estado de pedido exitoso, se notifica al cliente
		Selecciona el pedido "aceptado", cambiar a "en preparacion".	Cambio estado de pedido exitoso, se notifica al cliente
		Selecciona el pedido "en preparacion", cambiar a "en transito"	Cambio estado de pedido exitoso, se notifica al cliente
	Manejo exitoso de errores para cambios de estado no válidos	Selecciona el pedido con estado entregaso, cambiar a en curso	Mensaje de error que indica que no se permite el cambio de estado

Estructuración de *template* del usuario dueño de restaurantes para el *release* 1

Ver pedidos para mi restaurante	Ver con éxito los pedidos entrantes	Una persona externa realiza un pedido al restaurante, el usuario dentro del panel de pedidos.	Ver una lista de pedidos entrantes mostrados
	Aceptar un pedido con éxito	El pedido con estado "pendiente", cambiar a "aceptado"	Opción para aceptar el pedido, desencadenando un proceso de confirmación que actualiza el estado del pedido a "Aceptado" y notifica al usuario
	Rechazar un pedido con éxito	El pedido con estado "pendiente", cambiar a "rechazado"	Opción para rechazar el pedido, desencadenando un proceso de confirmación que actualiza el estado del pedido a "Rechazado" y notifica al usuario
	Ver con éxito los detalles del pedido	Click sobre el pedido "aceptado"	Poder ver detalles completos del pedido, incluidos los artículos, etc.
	Alertas de notificación exitosas	Persona externa cancela un pedido	Recibir una notificación o alerta en tiempo real para atender rápidamente el pedido entrante
	Ver con éxito el historial de pedidos	Ingresa a historial de pedidos	Tener la capacidad de ver un historial de pedidos anteriores

Estructuración de *template* del usuario dueño de restaurantes para el *release* 1

Resultados prueba de aceptación Release 1 - Dueños de restaurantes 1



Casos de prueba guiados para administradores de restaurantes							
Historia de usuario	Criterios de aceptación	Entradas	Resultado esperado	Resultado obtenido	Comentarios	Capturas de pantalla	Tiempo estimado de prueba (en horas)
Inicio de sesión	Inicio de sesión exitoso	Nombre de usuario "restaurante", contraseña "sushi"	Inicio de sesión del administrador inicia	Cumplido			
	Contraseña incorrecta	Nombre de usuario "restaurante", contraseña "tacos"	Se muestra un mensaje de error	Cumplido			
	Usuario no válido	Nombre de usuario "tacos", contraseña "sushi"	La barra lateral debería tener la opción de colapsarse	Cumplido			
	Compatibilidad y accesibilidad exitosas en diferentes dispositivos y navegadores	Ingresar desde chrome	La página de inicio de sesión debería ser receptiva y funcionar correctamente	Cumplido			0,10
		Ingresar desde Safari o Firebox	La página de inicio de sesión debería ser receptiva y funcionar correctamente	Cumplido			0,10
Cerrar sesión	Funcionalidad de cierre de sesión exitosa	Hace click en botón de cerrar sesion	Sesión cerrada y redirigido a la página de inicio.	Cumplido			0,10
Editar estado de mis pedidos	Acceso exitoso al panel de gestión de pedidos	Hace click en la opción de sidebar para ver el panel de gestion de pedidos	Ver una opción para ver y gestionar pedidos	Cumplido			0,40
	Ver con éxito la lista de pedidos con estado	Seleccion ver pedidos, dentro del panel de gestion de pedidos	Ver detalles relevantes junto con el estado actual de los pedidos	Cumplido	Me gusta mucho esto		
	Edición exitosa del estado del pedido	Selecciona el pedido "pendiente" , cambiar a "aceptado"	Cambio estado de pedido exitoso, se notifica al cliente	Cumplido			
		Selecciona el pedido "aceptado" , cambiar a "en preparacion".	Cambio estado de pedido exitoso, se notifica al cliente	Cumplido			
		Selecciona el pedido "en preparacion", cambiar a "en transito"	Cambio estado de pedido exitoso, se notifica al cliente	Cumplido			
	Manejo exitoso de errores para cambios de estado no válidos	Selecciona el pedido con estado entregado, cambiar a en curso	Mensaje de error que indica que no se permite el cambio de estado	Cumplido			
Ver pedidos para mi restaurante	Ver con éxito los pedidos entrantes	Una persona externa realiza un pedido al restaurante, el usuario dentro del panel de pedidos.	Ver una lista de pedidos entrantes mostrados	Cumplido			
	Aceptar un pedido con éxito	El pedido con estado "pendiente" , cambiar a "aceptado"	Opción para aceptar el pedido, desencadenando un proceso que actualiza el estado del pedido a "Aceptado" y notifica al usuario	Cumplido	No muestra algun aviso de confirmacion antes de mover de un estado a otro		0,20
	Rechazar un pedido con éxito	El pedido con estado "pendiente", cambiar a "rechazado"	Opción para rechazar el pedido, desencadenando un proceso de confirmación que actualiza el estado del pedido a "Rechazado" y notifica al usuario	Cumplido			
	Ver con éxito los detalles del pedido	Click sobre el pedido "aceptado"	Pedido con detalles completos del pedido, incluidos los productos.	Cumplido			
	Alertas de notificación exitosas	Persona externa cancela un pedido	Recibir una notificación o alerta en tiempo real para atender rápidamente el pedido entrante	Cumplido			
	Ver con éxito el historial de pedidos	Ingresa a historial de pedidos	Tener la capacidad de ver un historial de pedidos anteriores	Cumplido			
			Totales				**0,80**

Resultados prueba de aceptación de un dueño de restaurante para el *release* 1

Template pruebas guionadas Release 1 - Managers de TuBocado

Historia de usuario	Criterios de aceptación	Entrada	Resultado esperado
Inicio de sesión	Inicio de sesión exitoso	Nombre de usuario "manager", contraseña "tuBocado"	Inicio de sesión del administrador inicia
	Contraseña incorrecta	Nombre de usuario "manager", contraseña "sushi"	Se muestra un mensaje de error
	Usuario no válido	Nombre de usuario "bocados" contraseña "tuBocado"	La barra lateral debería tener la opción de colapsarse
	Compatibilidad y accesibilidad exitosas en diferentes dispositivos y navegadores	Ingresar desde chrome	La página de inicio de sesión debería ser receptiva y funcionar correctamente
		Ingresar desde Safari o Firebox	La página de inicio de sesión debería ser receptiva y funcionar correctamente
Cerrar sesión	Funcionalidad de cierre de sesión exitosa	Hace click en boton de cerrar sesion	Sesión cerrada y redirigido a la página de inicio.
Navegar entre páginas utilizando la barra lateral	Éxito al navegar a diferentes secciones utiliznndo el panel de managers	Hace click en la seccion "Restaurantes"	Se va a la seccion clickeado
		Hace click en la seccion "Bocados"	Se va a la seccion clickeado
		Hace click en la seccion "Finanzas"	Se va a la seccion clickeado
		Hace click en la seccion "Administradores"	Se va a la seccion clickeado
		Hace click en la seccion "Usuarios"	Se va a la seccion clickeado
	Ver resaltado exitoso de la sección activa	Click en cualquier seccion del panel	El enlace de la barra lateral para esa sección debería destacarse visualmente

Estructuración de *template* del usuario manager de TuBocado para el *release* 1

Editar niveles y porcentajes de puntos	Acceso exitoso a la sección de gestión de bocados "BOCADOS"	Hace click en la seccion "Bocados"	Tener acceso a una sección o interfaz dedicada para gestionar niveles y porcentajes de puntos
	Edición exitosa de niveles de bocados	Entra en la gestion de "Bocados", cambiar nivel de puntos de 3 a 4	Capacidad para editar la cantidad de niveles existentes
	Modificación exitosa de los porcentajes de generación de bocados	Entra a la gestion de "Bocados", cambiar el valor para la generacion de bocados de 500 a 600.	Poder ajustar el porcentaje de puntos otorgados por cada uno cuando se realiza la acción

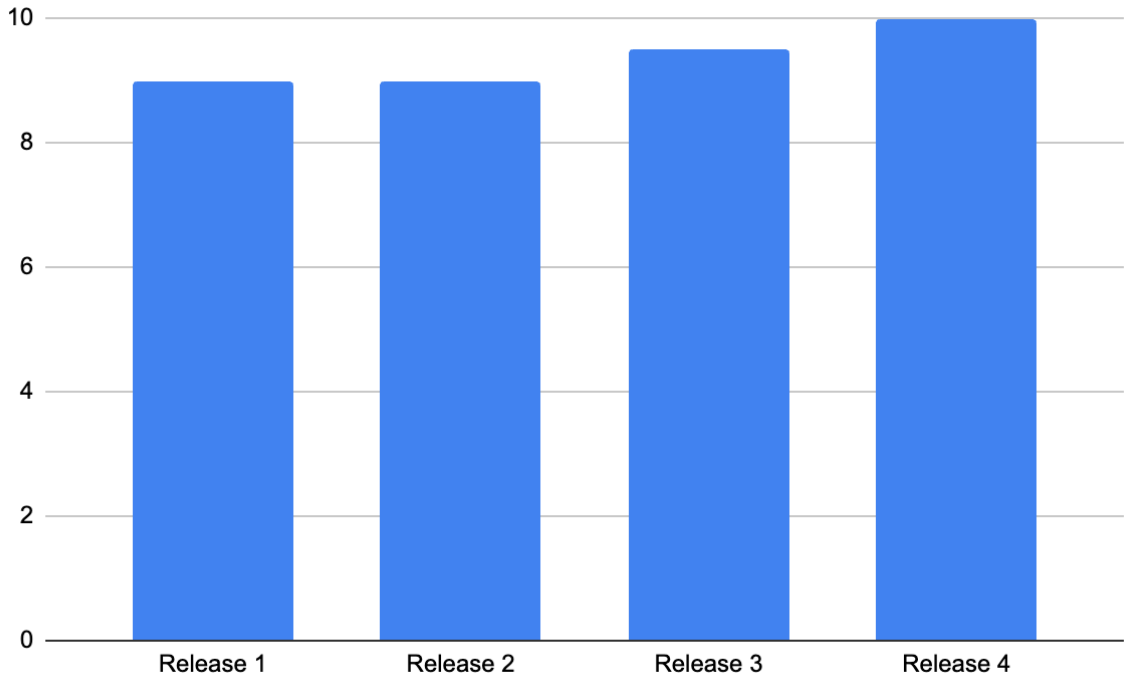
Estructuración de *template* del usuario manager de TuBocado para el *release* 1

Resultados prueba de aceptación Release 1 - Manager de TuBocado

Historia de usuario	Criterios de aceptación	Caso de pruebas guiados para administradores Entrada	Resultado esperado	Resultado obtenido	Comentarios	Capturas de pantalla	Tiempo estimado de prueba (en horas)
Inicio de sesión	Inicio de sesión exitoso	Nombre de usuario "manager", contraseña "tuBocado"	Inicio de sesión del administrador inicia	Cumplido			0,1
	Contraseña incorrecta	Nombre de usuario "manager", contraseña "sadh"	Se muestra un mensaje de error	Cumplido			
	Usuario no existe	Nombre de usuario "tbocados" contraseña "tubocado"	La barra lateral debería tener la opción de colapsarse	Cumplido			
	Comprobar la accesibilidad exitosa en diferentes dispositivos y navegadores	Ingresar desde chrome	La página de inicio de sesión debería ser receptiva y funcionar correctamente	Cumplido			
Cerrar sesión	Funcionalidad de cierre de sesión exitosa	Ingresar desde Safari o Firebox	La página de inicio de sesión debería ser receptiva y funcionar correctamente	Cumplido			
Navegar entre páginas utilizando la barra lateral		Hace click en botón de cerrar sesión	Sesión cerrada y redirigido a la página de inicio	Cumplido			0,05
		Hace click en la opción "Administradores"	Se va a la seccion clickeado	Cumplido			0,1
		Hace click en la sección "Bocados"	Se va a la seccion clickeado	Cumplido			
	Éxito al navegar a diferentes secciones utilizando el panel de managers	Hace click en la sección "Finanzas"	Se va a la seccion clickeado	Cumplido			
		Hace click en la sección "Administradores"	Se va a la seccion clickeado	Cumplido			
		Hace click en la sección "Usuarios"	Se va a la seccion clickeado	Cumplido			
	Ver resultado exitoso de la sección activa	Click en cualquier seccion del panel	El enlace de la barra lateral para esa sección debería estar activo	Cumplido			
Editar niveles y porcentajes de puntos	Acceso exitoso a la sección de gestión de bocados "BOCADOS"	Hace click en la sección "Bocados"	Se debería cargar una interfaz dedicada para gestionar niveles y porcentajes de puntos	Cumplido			0,8
	Edición exitosa de niveles de bocados	Entra en la gestion de "Bocados", cambiar nivel de puntos de 3 a 4	Capacidad para editar la cantidad de niveles existentes	Cumplido			
	Modificación exitosa de los porcentajes de generación de bocados	Entra a la gestion de "Bocados", cambiar el valor para la generacion de bocados de 300 a 600.	Poder ajustar el porcentaje de puntos otorgados por cada uno cuando se realiza la acción	Cumplido	Ojo con la trazabilidad aca, hay que asegurarse que cuando se modifica se muestre la cotizacion de ese momento y no la modifico y esta actual		
		Totales					**1,05**

Resultados prueba de aceptación del manager de TuBocado para el *release* 1

A.27. Resultados encuesta “¿Cuán probable es que…” a P.O



Gráfica resultados encuesta recomendación producto con P.O

RELEASE 1		
Usuario	**Respuesta**	**Comentario**
Product Owner	9	-

Resultados encuesta recomendación producto con P.O - Release 1

RELEASE 2		
Usuario	**Respuesta**	**Comentario**
Product Owner	9	-

Resultados encuesta recomendación producto con P.O - Release 2

RELEASE 3		
Usuario	**Respuesta**	**Comentario**
Product Owner	9,5	-

Resultados encuesta recomendación producto con P.O - Release 3

RELEASE 4

Usuario	Respuesta	Comentario
Product Owner	10	-

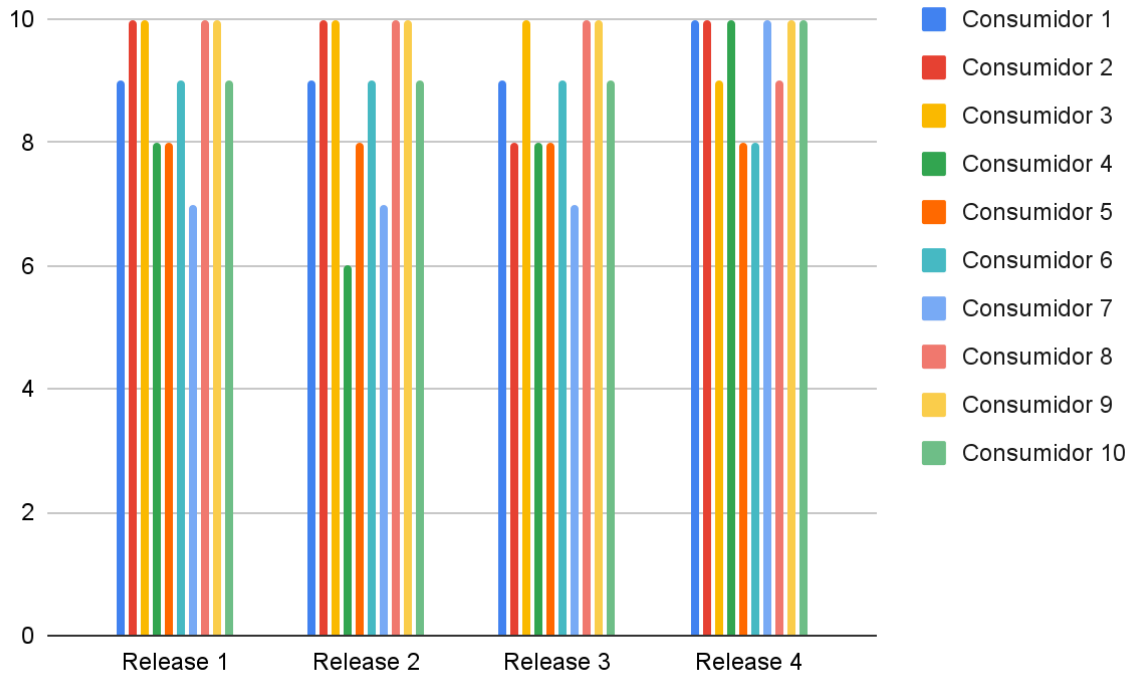
Resultados encuesta recomendación producto con P.O - Release 4

A.28. Cálculo key result NPS "¿Cuán probable es que…" a P.O

		Product Owner		Cantidad	Detractores	Pasivos	Promotores	NPS
PO	Release 1	9		1	0	0	100	100
	Release 2	9			0	0	100	100
	Release 3	9,5			0	0	100	100
	Release 4	10			0	0	100	100

Cálculo NPS producto a P.O

A.29. Resultados encuesta "¿Cuán probable es que…" a Consumidores



Gráfica resultados encuesta recomendación producto con Consumidores

RELEASE 1		
Usuario	**Respuesta**	**Comentario**

Consumidor - 1	9	*Definitivamente recomendaría este producto a mis amigos. Es increíblemente útil y conveniente.*
Consumidor - 2	10	*Se merece esta puntuación sin lugar a dudas.*
Consumidor - 3	10	*Recomendaría este producto a cualquiera que busque una solución rápida para pedir sushi. Estoy encantado con esto.*
Consumidor - 4	8	*Este producto me parece muy útil. Quiero ver más adelante como sigue.*
Consumidor - 5	8	*No puedo decir suficientes cosas buenas sobre este producto. Es excepcional y definitivamente lo recomendaría a mis amigos que siempre estamos buscando pedir sushi y está caro o tenemos que ir comparando precios.*
Consumidor - 6	9	*El producto es fácil de usar, puede tener potencial. Igual quiero ver como evoluciona.*
Consumidor - 7	7	*Me gusta mucho todo lo del sistema de puntos, seguro lo recomiendo. Aun creo que hay cosas para mejorar.*
Consumidor - 8	10	*No puedo esperar a que salga y lo use para pedir sushi. Merece una calificación muy buena.*
Consumidor - 9	10	*He utilizado muchos productos similares en el pasado, pero este se destaca claramente por todos los beneficios de los canjes de puntos. Lo sugeriría a cualquiera.*
Consumidor - 10	9	*Me gusta mucho, obvio que lo recomendaría a todo el mundo.*

Resultados encuesta recomendación producto con Consumidores - Release 1

RELEASE 2		
Usuario	**Respuesta**	**Comentario**
Consumidor - 1	9	*Me encanta la facilidad de uso de esto y lo recomendaría con una sólida alta puntuación.*
Consumidor - 2	10	*Esta app merece un sólido 10 de mi parte! Me parece muy conveniente, al fin voy a comprar sushi más seguido.*
Consumidor - 3	10	*Recomendaría a cualquiera que busque una solución rápida para ordenar sushi. Me pareció muy buena idea.*
Consumidor - 4	6	*Me gusta la idea, pero creo que el diseño de algunas cosas debería mejorar, como donde se ven las direcciones de cada uno.*
Consumidor - 5	8	*Me encanta que pueda referir a gente y gane puntos y lo recomendaría a todos mis amigos que buscan una forma fácil de ordenar sushi.*
Consumidor - 6	9	*La fiabilidad y facilidad de uso de este producto lo hacen destacar. Definitivamente, lo recomendaría.*
Consumidor - 7	7	*Me impresiona mucho el sistema de puntos de este producto. Creo que debe mejorar algunos aspectos como el tema de las direcciones.*

Consumidor - 8	10	*Aunque aún no he tenido la oportunidad de usar este producto para ordenar sushi, tengo la sensación de que merece una muy buena puntuación. Le veo potencial.*
Consumidor - 9	10	*Después de haber probado varios productos similares en el pasado, este se destaca claramente por sus beneficios en los canjes de puntos. Sin duda lo sugeriría a cualquiera.*
Consumidor - 10	9	*Me gusta mucho.*

Resultados encuesta recomendación producto con Consumidores - Release 2

RELEASE 3		
Usuario	**Respuesta**	**Comentario**
Consumidor - 1	9	*Es increíblemente fácil de usar y definitivamente lo recomendaría.*
Consumidor - 2	8	*Me gusto la UI siento que es muy linda y me gusta lo que hace la app.*
Consumidor - 3	10	*Recomendaría este servicio a cualquier persona que busque una manera rápida y eficiente de ordenar sushi. Estoy completamente impresionado y le doy una puntuación perfecta.*
Consumidor - 4	8	*Con total franqueza, encuentro que este producto es sumamente útil porque justamente el sushi es caro entonces impulsa a que lo compren.*
Consumidor - 5	8	*Me encanta la posibilidad de referir amigos y ganar puntos. Definitivamente lo recomendaría a aquellos que buscan una forma simple de pedir sushi.*
Consumidor - 6	9	*La facilidad de uso de esto es muy buena. Lo recomendaría sin dudarlo.*
Consumidor - 7	7	*El sistema de puntos de este producto me ha dejado muy impresionado. Por ello, lo recomendaría. Sin embargo, creo que deben mejorar algunos aspectos sobre todo con el flujo de ver mi pedido.*
Consumidor - 8	10	*Aunque aún no he tenido la oportunidad de probar este producto para pedir sushi realmente, estoy seguro de que merece una calificación perfecta.*
Consumidor - 9	10	*Merece una buena calificación y se lo recomendaría a cualquiera*
Consumidor - 10	9	*Me gusta mucho y que hayan incluido tarjeta aún más.*

Resultados encuesta recomendación producto con Consumidores - Release 3

RELEASE 4		
Usuario	**Respuesta**	**Comentario**
Consumidor - 1	10	*Me encanta, hace que pedir sushi sea pan comido. Le doy sin dudas la mayor calificación.*
Consumidor - 2	10	*¡Totalmente recomendado!*

Consumidor - 3	9	*Este servicio para ordenar sushi es una bomba. Quién no quiere un sushi con algún descuento?*
Consumidor - 4	10	*Lo de los puntos por referidos es lo máximo. Sin duda, recomendaría esta movida a todos los que quieran sushi fácil.*
Consumidor - 5	8	*Tremenda onda esta aplicación. Seguro que a muchos les vendría bien.*
Consumidor - 6	8	*El sistema de puntos de esta movida me dejó encantada. Creo que tiene algunas cosas para mejorar, pero igual los puntos son geniales!*
Consumidor - 7	10	*¡Esta cosa es una maravilla en términos de facilidad!*
Consumidor - 8	9	*Avísame cuando este público para todos que lo quiero usar, ¡le doy un 9 sin dudarlo! ¡Seguro que a muchos les encantaría!*
Consumidor - 9	10	*Esta idea de los puntos es buenísima, va a ser un éxito!*
Consumidor - 10	10	*Estoy seguro de que esto será lo mejor para pedir sushi. Estoy seguro de que será increíble*

Resultados encuesta recomendación producto con Consumidores - Release 4

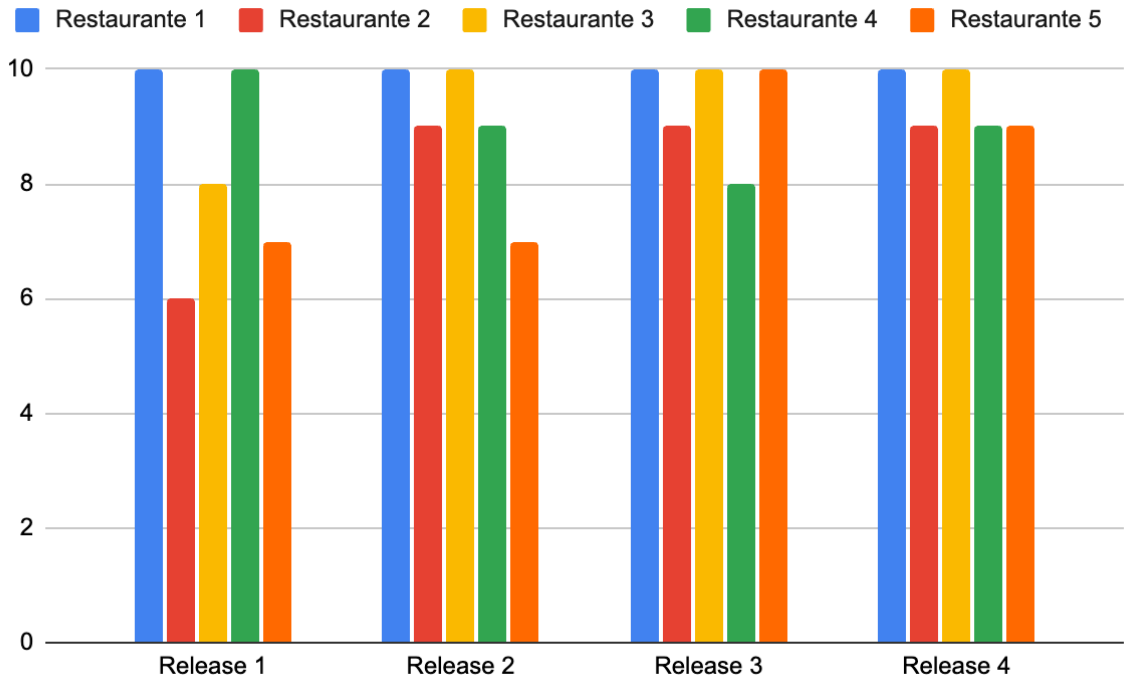
A.30. Cálculo key result NPS “¿Cuán probable es que…” a Consumidores

		Consumidor 1	Consumidor 2	Consumidor 3	Consumidor 4	Consumidor 5	Consumidor 6	Consumidor 7	Consumidor 8	Consumidor 9	Consumidor 10
CONSUMIDORES	Release 1	9	10	10	8	8	9	7	10	10	9
	Release 2	9	10	10	6	8	9	7	10	10	9
	Release 3	9	8	10	8	8	9	7	10	10	9
	Release 4	10	10	9	10	8	8	10	9	10	10

Cantidad	Detractores	Pasivos	Promotores	NPS
10	0	30	70	70
	10	20	70	60
	0	40	60	60
	0	20	80	80

Resultados cálculo NPS producto a Consumidores

A.31. Resultados encuesta “¿Cuán probable es que…” a Dueños de restaurantes



Gráfica resultados encuesta recomendación producto con Dueños de restaurantes

RELEASE 1		
Usuario	**Respuesta**	**Comentario**
D.R - 1	10	*Esta aplicación promete ser una muy buena herramienta para mi restaurante. Va a simplificar enormemente el proceso de pedidos y espero que sirva para aumentar nuestra clientela. Definitivamente la recomendaría.*
D.R - 2	6	*Le doy un 6 a esta aplicación. Parece fácil de usar pero no distingo mucho como se va a diferenciar con la competencia.*
D.R - 3	8	*Estoy bastante impresionado con las funcionalidades de esta aplicación. Me gusta especialmente cómo va a facilitar la comunicación con los clientes.*
D.R - 4	10	*Esta aplicación va a transformar la forma en que gestionamos los pedidos en nuestro restaurante. Definitivamente la recomendaría.*
D.R - 5	7	*La aplicación promete ser muy útil, cuando salga al mercado la voy probar en mi negocio.*

Resultados encuesta recomendación producto con Dueños restaurantes - Release 1

RELEASE 2		
Usuario	**Respuesta**	**Comentario**
D.R - 1	10	*La aplicación promete ser un cambio total para nuestro restaurante. No puedo esperar para comenzar a usarla. Sin lugar a dudas la recomendaría.*
D.R - 2	9	*Estoy muy impresionado con esta aplicación. Nos va a permitir gestionar mejor los tiempos de preparación y mejorar la experiencia del cliente. Obviamente la recomendaría.*
D.R - 3	10	*Como dueño de restaurante, probe varias aplicaciones de pedidos, y esta me llamo la atención. La interfaz promete ser intuitiva. La recomendaría sin dudarlo.*
D.R - 4	9	*La aplicación va a simplificar enormemente nuestras operaciones. Desde la gestión de menús hasta el seguimiento de pedidos.*
D.R - 5	7	*Aunque al principio estaba escéptico sobre cómo funciona esta aplicación, al comprender mejor el modelo de negocio, creo que me gustaría usarla. Algo que no me gusta son los colores naranja y negro, se ve raro.*

Resultados encuesta recomendación producto con Dueños restaurantes - Release 2

RELEASE 3		
Usuario	**Respuesta**	**Comentario**
D.R - 1	10	*Esta aplicación va a ser una verdadera revelación para nuestro negocio. Nos va a permitir llegar a nuevos clientes y gestionar los pedidos de manera mucho más eficiente. Definitivamente la recomendaría.*
D.R - 2	9	*Estoy muy impresionado con la funcionalidad y la facilidad de uso de esta aplicación.*
D.R - 3	10	*La aplicación parece que va a tener un impacto significativo en las ventas y en la satisfacción del cliente. La recomendaría sin dudarlo.*
D.R - 4	8	*Esta aplicación va a ser una herramienta invaluable para nuestro restaurante. Nos va a ayudar a optimizar nuestros procesos y a ofrecer un mejor servicio a nuestros clientes.*
D.R - 5	10	*La aplicación ha superado todas mis expectativas. Promete ser intuitiva y fácil de usar.*

Resultados encuesta recomendación producto con Dueños restaurantes - Release 3

RELEASE 4		
Usuario	**Respuesta**	**Comentario**
D.R - 1	10	*Esta aplicación va a ser un verdadero cambio de juego para nosotros. No solo va a simplificar nuestros pedidos, sino que también va a mejorar nuestra eficiencia operativa.*
D.R - 2	9	*Como propietario de un restaurante, puedo decir que esta aplicación va a ser una inversión que valga la pena. Va a mejorar la gestión de pedidos y la comunicación con los clientes.*
D.R - 3	10	*Estoy muy contento con los resultados que promete la aplicación. La idea va a aumentar nuestra visibilidad y va a facilitar que los clientes realicen pedidos.*
D.R - 4	9	*La aplicación va a ser una gran adición a nuestro negocio. Nos va a ayudar a optimizar nuestras operaciones y a brindar un mejor servicio al cliente.*
D.R - 5	9	*Esta aplicación va a ser una herramienta invaluable para nuestro restaurante. No solo va a simplificar nuestros pedidos, sino que también va a mejorar nuestra capacidad para gestionar la demanda en momentos de alto tráfico.*

Resultados encuesta recomendación producto con Dueños restaurantes - Release 4

A.32. Cálculo key results NPS “¿Cuán probable es que…” a Dueños de restaurantes

		Restaurante 1	Restaurante 2	Restaurante 3	Restaurante 4	Restaurante 5
DUEÑOS RESTAURANTES	Release 1	10	6	8	10	7
	Release 2	10	9	10	9	7
	Release 3	10	9	10	8	10
	Release 4	10	9	10	9	9

Cantidad	Detractores	Pasivos	Promotores	NPS
5	20	40	40	20
	0	20	80	80
	0	20	80	80
	0	0	100	100

Resultados cálculo NPS producto a Dueños de restaurantes

A.33. Encuesta satisfacción del cliente con producto - 4 temáticas

	Rendimiento y velocidad	Facilidad de uso	Cumplimiento de requerimientos	Evaluación general
Release 1	4	5	2	3
Release 2	3	5	3	4
Release 3	5	5	4	4
Release 4	5	5	5	5

Resumen resultados totales encuesta satisfacción del cliente con producto

Release 1		
Área de enfoque	**Respuesta**	**Comentario**
Rendimiento y velocidad	4	*Responde en tiempo razonable*
Facilidad de uso	5	*Me resultó muy fácil usar la plataforma*
Cumplimiento de requerimientos	2	*Faltan incorporar varias funciones*
Evaluación general	3	*Satisfecho con los que se me mostró*

Resultados encuesta satisfacción del cliente con producto - Release 1

Release 2		
Área de enfoque	**Respuesta**	**Comentario**
Rendimiento y velocidad	3	*Algunas veces me demoro mas de lo esperado en cargar la página*
Facilidad de uso	5	*Super fácil de usar la aplicación*
Cumplimiento de requerimientos	3	*Gran mejora, siguen faltando varias funciones*
Evaluación general	4	*Globalmente satisfecho, mejorar en la consistencia del diseño*

Resultados encuesta satisfacción del cliente con producto - Release 2

Release 3		
Área de enfoque	**Respuesta**	**Comentario**
Rendimiento y velocidad	5	*Perfectos tiempos de carga*
Facilidad de uso	5	*Muy fácil de usar*
Cumplimiento de requerimientos	4	*Cumple con la mayoría de las funciones*
Evaluación general	4	*Muy completo, falta algunas funcionalidades*

Resultados encuesta satisfacción del cliente con producto - Release 3

Release 4		
Área de enfoque	**Respuesta**	**Comentario**
Rendimiento y velocidad	5	*Perfecto*
Facilidad de uso	5	*Excelente*
Cumplimiento de requerimientos	5	*Cumple con todo la pactado*
Evaluación general	5	*Sin ninguna duda muy buen trabajo*

Resultados encuesta satisfacción del cliente con producto - Release 4

A.34. Encuesta satisfacción del cliente con el equipo - 5 temáticas

	Trato profesional y de respeto	Puntualidad	Claridad en la comunicación	Compromiso del equipo	Adaptacion a los cambios
Release 1	5	5	5	4	3
Release 2	5	5	4	4	5
Release 3	5	5	5	5	5
Release 4	5	5	5	5	5

Resumen resultados totales encuesta satisfacción del cliente con equipo

Release 1		
Área de enfoque	**Respuesta**	**Comentario**
Trato profesional y de respeto	5	*Excelente el trato personal*
Puntualidad	5	*Nunca tuve problema con la puntualidad con el equipo*
Claridad en la comunicación	5	*Perfecta siempre la comunicación*
Compromiso del equipo	4	*Siempre atentos*
Adaptación a los cambios	3	*Requiere varias reuniones transmitir ideas nuevas*

Resultados encuesta satisfacción del cliente con equipo - Release 1

Release 2		
Área de enfoque	**Respuesta**	**Comentario**
Trato profesional y de respeto	5	*Excelente el trato personal*
Puntualidad	5	*Siempre en hora*
Claridad en la comunicación	4	*Casi siempre nos comunicamos sin problema*

Compromiso del equipo	4	*Muy bueno el compromiso*
Adaptación a los cambios	5	*Reaccion perfecta nuevas ideas*

Resultados encuesta satisfacción del cliente con equipo - Release 2

Release 3		
Área de enfoque	**Respuesta**	**Comentario**
Trato profesional y de respeto	5	*Excelente el trato personal*
Puntualidad	5	*Siempre puntales*
Claridad en la comunicación	5	*Ningún problema para comunicarnos*
Compromiso del equipo	5	*Impresionante el compromiso*
Adaptación a los cambios	5	*Se adaptan perfecto a los cambios*

Resultados encuesta satisfacción del cliente con equipo - Release 3

Release 4		
Área de enfoque	**Respuesta**	**Comentario**
Trato profesional y de respeto	5	*Muy profesional el trato*
Puntualidad	5	*Perfecta puntualidad*
Claridad en la comunicación	5	*Hermosa la comunicación que se maneja*
Compromiso del equipo	5	*Impresionante el compromiso de ellos*
Adaptación a los cambios	5	*Totalmente adaptables a los cambios planteados*

Resultados encuesta satisfacción del cliente con equipo - Release 4

A.35. Cálculo key result cumplimiento heurísticas de nielsen

	Heuristica 1	Heuristica 2	Heuristica 3	Heuristica 4	Heuristica 5	Heuristica 6	Heuristica 7	Heuristica 8	Heuristica 9	Heuristica 10
Release 1	No	Si	Si	Si	No	No	Si	Si	Si	Si
Release 2	No	Si	Si	Si	Si	No	Si	Si	Si	Si
Release 3	Si	Si	Si	Si	Si	No	Si	Si	Si	Si
Release 4	Si	Si	Si	Si	Si	Si	Si	Si	Si	Si

	Cantidad	Porcentaje
	7	70
	8	80
	9	90
	10	100

	Cumplimiento
Release 1	70
Release 2	80
Release 3	90
Release 4	100

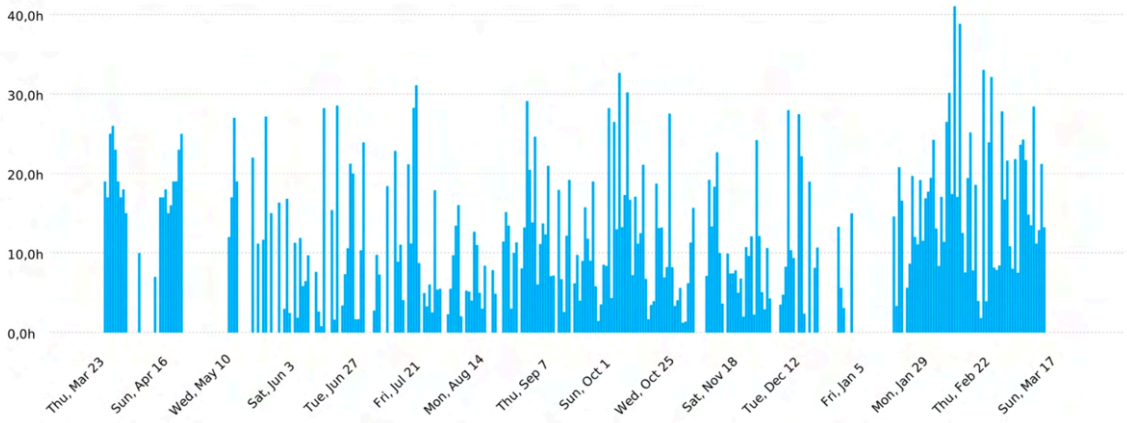
Resumen heurísticas de nielsen cumplidas por cada release

A.36. Trackeo de horas mensuales

Summary report

23/03/2023 - 22/03/2024

Total: 3361:21:11 Billable: 3361:21:11 Amount: 0,00 USD



Month



Mar 23 - Mar 31, 2023	179:02:02	5,33%
Apr 2023	186:00:25	5,53%
May 2023	198:10:35	5,90%
Jun 2023	234:28:04	6,98%
Jul 2023	233:26:20	6,94%
Aug 2023	249:11:07	7,41%
Sep 2023	315:59:00	9,40%
Oct 2023	367:51:20	10,94%
Nov 2023	250:18:43	7,45%
Dec 2023	191:04:58	5,68%
Jan 2024	221:22:16	6,59%
Feb 2024	512:14:50	15,24%
Mar 1 - Mar 22, 2024	222:11:31	6,61%

A.37. Encuestas de grado de satisfacción del equipo con metodología

	Sanguinetti	Rostan	Charbonnier	Torres
Release 1	3	3	3	2
Release 2	3	3	3	3
Release 3	3	2	3	3
Release 4	3	3	3	3

A.38. Cálculo de retrabajo

GRAFICA RETRABAJO	
Total Desarrollo	Retrabajo
1069,39	262,16

A.39. Cálculo de la velocidad

GRAFICA VELOCIDAD	
Velocidad: cantidad de StoryPoints por sprint	Valor
Sprint 1	49
Sprint 2	49
Sprint 3	61
Sprint 4	16
Sprint 5	68
Sprint 6	65
Sprint 7	67
Sprint 8	68
Sprint 9	52
Sprint 10	63
Sprint 11	64
Sprint 12	62
Sprint 13	26

A.40. Cálculo de la productividad

GRAFICA PRODUCTIVIDAD	
Releases	Valor
Release 1	
Release 2	1,697545455
Release 3	1,717738693
Release 4	1,41951049

A.41. Evidencia de cobertura de código



Evidencia satisfacción de tests

All files modules/administrators/repositories

89.28% Statements 75/84 100% Branches 12/12 87.5% Functions 7/8 89.65% Lines 78/87

Press n or j to go to the next uncovered block, b, p or k for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
admins.repo.ts	89.28%	75/84	100%	12/12	87.5%	7/8	89.65%	78/87

Evidencia de cobertura de código de módulo administrators/repositories

All files modules/administrators/useCases

93.59% Statements 73/78 **91.67%** Branches 11/12 **100%** Functions 18/18 **91.67%** Lines 88/96

Press n or j to go to the next uncovered block, b, p or k for the previous block.

Filter:

File	Statements	Branches	Functions	Lines
loginAdmin.ts	100% 13/13	100% 2/2	100% 1/1	84.61% 11/13
signupAdmin.ts	91.67% 11/12	100% 1/1	100% 1/1	90.90% 10/11
addAdmin.ts	100% 10/10	100% 1/1	100% 1/1	100% 9/9
getActiveParam.ts	100% 3/3	100% 0/0	100% 1/1	100% 3/3
getAdmins.ts	71.42% 5/7	100% 1/1	100% 1/1	71.42% 5/7
removeAdmin.ts	100% 3/3	100% 1/1	100% 1/1	100% 3/3
createLevels.ts	100% 6/6	50% 1/2	100% 1/1	87.5% 7/8
createParam.ts	85.7% 6/7	100% 2/2	100% 1/1	100% 13/13
getAdminInfo.ts	87.5% 7/8	100% 0/0	100% 1/1	100% 11/11
updateAdminById.ts	100% 9/9	100% 2/2	100% 1/1	88.89% 16/18

Evidencia de cobertura de código de módulo *administrators/useCases*

A.42. Evidencia de prueba unitaria creada

```
describe('addAdmin', () => {
    afterEach(() => {
        jest.clearAllMocks();
    });

    it('should add admin successfully', async () => {
        const signupData: SignupDto = {
            email: 'test@example.com',
            password: encryptPassword('password123'),
            name: 'Test User',
            phone: '1234567890',
        };

        const expectedResponse: SignInResponse = {
            admin: {
                email: signupData.email,
                name: signupData.name,
                phone: signupData.phone,
            },
        };

        getAdmin.mockResolvedValue(null);

        createAdministrator.mockResolvedValue(signupData); // Mock the createAdministrator function

        const result: SignInResponse = await addAdmin(signupData);

        expect(getAdmin).toHaveBeenCalledWith('email', signupData.email.toLowerCase());
        expect(createAdministrator).toHaveBeenCalledWith({
            password: signupData.password,
            name: signupData.name,
            email: signupData.email,
            phone: signupData.phone,
        });
        expect(result).toEqual(expectedResponse);
    });

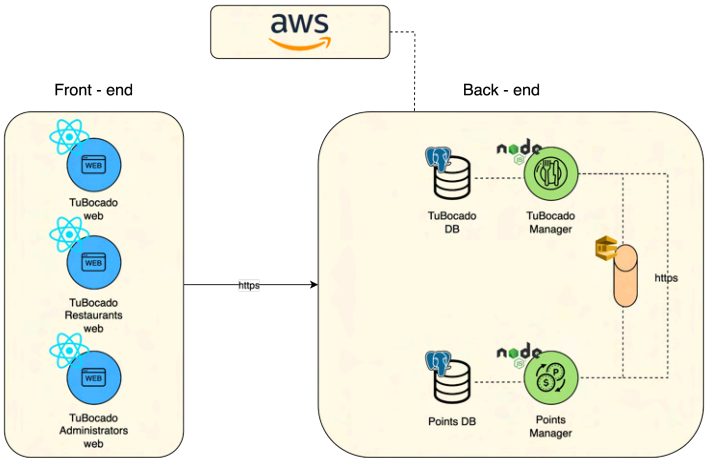
    it('should throw UserError if admin already exists', async () => {
        const signupData: SignupDto = {
            email: 'test@example.com',
            password: encryptPassword('password123'),
            name: 'Test User',
            phone: '1234567890',
        };

        getAdmin.mockResolvedValue({ email: signupData.email });

        await expect(addAdmin(signupData)).rejects.toThrow(UserError);
    });
});
```

Ejemplo de prueba unitaria de creación de un administrador

A.43. Diagrama con un resumen de tecnologías usadas



Resumen de tecnologías utilizadas