Universidad ORT Uruguay
Facultad de Ingeniería

Ariadne's Thread

Gestor organizacional para Qualabs

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

Valentina Chalela – 212725

Keshet Hertz – 209074

Evgeny Sidagis – 222144

Patrick Silveira – 213180

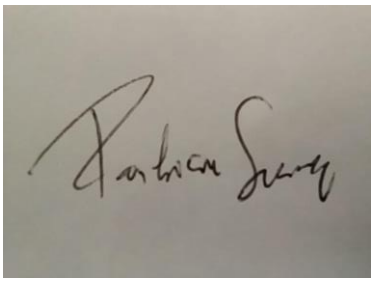
Tutor: Darío Macchi

2022

Declaración de autoría

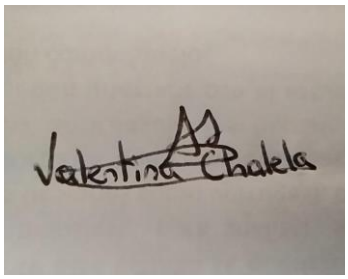
Nosotros, Valentina Chalela, Keshet Hertz, Evgeny Sidagis, Patrick Silveira, declaramos que el trabajo que se presenta en esa obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos el Proyecto final de la carrera 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.



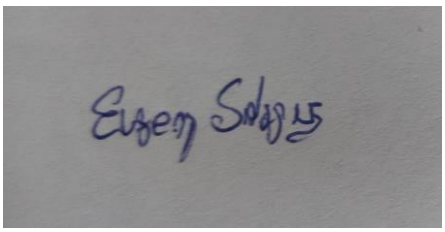
Patrick Silveira

28 de Setiembre de 2022



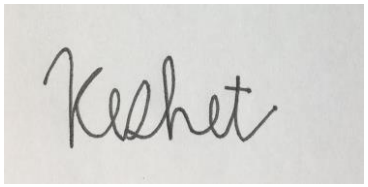
Valentina Chalela

28 de Setiembre de 2022



Evgeny Sidagis

28 de Setiembre de 2022



Keshet Hertz

28 de Setiembre de 2022

Agradecimientos

Para comenzar, queremos agradecer a nuestras familias, amigos y a todas las personas que nos apoyaron en la carrera y este proyecto, alentándonos a seguir adelante y dar lo mejor de nosotros.

El siguiente agradecimiento se lo dedicamos a nuestro tutor, Darío Macchi, quien nos apoyó durante el proyecto de forma dedicada y comprometida, guiando al equipo y dando consejos a lo largo del mismo, y por sus enseñanzas que nos ayudaron a mejorar tanto a nivel de proyecto como personal, y también sus aportes para resolver debates y conflictos.

Además, agradecemos muy especialmente a nuestro cliente, Qualabs, por darnos la posibilidad de trabajar en este proyecto, por la confianza depositada en nosotros y por sus invaluables aportes, brindando retroalimentaciones valiosas de forma constante y continua, dando sugerencias y la posibilidad de debatir con ellos, interactuando siempre de forma cálida y cercana.

Por otro lado, queremos agradecer a la Universidad ORT Uruguay y a todos los que nos acompañaron, sean catedráticos, docentes, ayudantes o funcionarios, y nos ayudaron a formarnos a nivel académico, profesional y personal a lo largo de estos años.

Finalmente, agradecemos a los profesores de la Cátedra de Ingeniería de Software que actuaron como revisores de este proyecto, Álvaro Ortas, Amalia Álvarez y Martín Solari, que nos brindaron retroalimentación clave, que nos ayudó a mejorar nuestro proceso y proyecto.

A todas estas personas les damos nuestras más sinceras gracias, nada de esto podría haber sido posible sin su apoyo.

¡Muchas gracias!

Abstract

La empresa Qualabs se encontraba en un período de alto crecimiento, lo cual les implicó nuevos desafíos. Uno de ellos era poder realizar una gestión eficiente de la información de la empresa, lo que era una tarea compleja. Esto era así porque la información de proyectos, equipos y personas, se encontraba distribuida en diferentes plataformas, como Microsoft Office, Google Drive, Asana y Mural.

Es por esto que surge Ariadne's Thread, sistema a medida para Qualabs, que fue desarrollado con una alta participación del cliente, manteniendo una comunicación fluida y respetando todas sus solicitudes, con el fin de que el producto pudiera solucionar las problemáticas que este estaba enfrentando. Para esto, se utilizaron metodologías ágiles, específicamente la de *Dual Track Scrum* y *Design Sprint*.

El sistema resultante consta de una aplicación web, construida utilizando la librería React para *frontend*, y un servidor *backend*, implementado con NodeJs y Typescript como lenguaje de programación. La aplicación fue desplegada en la nube en base a los servicios de *Amazon Web Services*.

Como resultado final, el cliente recibió un sistema que, desde la primera versión del software, ya lo utilizó de manera activa con datos reales (en su ambiente de producción). Esto hizo que el mismo tuviese la posibilidad de brindar *feedback* en base al uso generado, lo que permitió al equipo detectar posibles mejoras y adaptar el sistema a los procesos y preferencias del cliente.

Para concluir, en base a encuestas y comentarios recibidos se constató que el sistema logró satisfacer las necesidades y expectativas del cliente. Esto fue así, gracias a la interacción continua que se mantuvo con el mismo, siendo esta un punto fuerte del proyecto.

Palabras clave

Qualabs; People & Culture; *Dashboard*; *Drag & Drop*; *Dual Track Scrum*; *Design Sprint*; Prototipado; *Feedback*; React; Typescript; AWS; *Amazon Web Services*; NodeJs.

Glosario

- **Asana**: Plataforma web y móvil para ayudar a los equipos a organizar, seguir y gestionar su trabajo.
- ***Backend***: El *backend* es la parte lógica de un sistema, también conocida como la capa de acceso a datos.
- ***Backlog***: Lista priorizada de funcionalidades que debe tener un producto. Se realiza en conjunto con el cliente.
- ***Backlog refinement***: Reunión utilizada para refinar, seleccionar y agregar detalles como descripción o tamaño a las historias que podrían ser realizadas en un *sprint*.
- **Bitbucket**: Sistema de gestión de repositorios basado en git con funcionalidades de CI / CD y conexión con JIRA.
- ***Brainstorming***: *Brainstorming* o "tormenta de ideas" es un proceso de ideación basado en generar muchas ideas en un ambiente creativo con el equipo.
- ***Bucket* S3**: Servicio de almacenamiento seguro en la nube provisto por *Amazon Web Services*.
- ***Bug***: Se le considera *bug* a todo aquel error o problema que causa un comportamiento inesperado o incorrecto en un producto de software.
- ***Cache***: Capa de almacenamiento de alta velocidad que almacena datos temporalmente.
- ***Code review***: Ceremonia de Scrum en la que el cliente y el equipo participan y muestran los avances de una iteración.
- **Confluence**: Parte del ecosistema de Atlassian dedicado a la documentación, permitiendo conectarla con JIRA. Permite crear documentos formales.
- ***Daily***: Reunión diaria de Scrum, con el fin de conocer qué está haciendo cada integrante, si precisa ayuda y que hará luego de terminar la tarea en la que está trabajando.

- ***Dashboard*:** Pantalla principal del sistema que contiene la diagramación actual de equipos y personas de la empresa.

- ***Deployment***: Proceso por el cual un sistema se proporciona a los usuarios para su uso. En el caso del proyecto, los *deploys* fueron realizados en *Amazon Web Services*.

- ***Downtime***: *Downtime* refiere a periodos de tiempo en el cual el sistema no puede ser utilizado, ya sea por reparación, *deployment* de una nueva versión, o por haber tenido una falla.

- ***Drag & Drop***: Es una *feature* basada en "agarrar" un objeto, y "arrastrarlo" a otro lugar en la pantalla.

- ***Dual Track Scrum***: Ampliación de Scrum que además de contener una etapa de desarrollo, también tiene una etapa de diseño. Estas etapas quedan definidas en dos *tracks* o carriles, llamados etapas de descubrimiento y etapa de desarrollo, para el diseño y para el desarrollo respectivamente.

- ***Endpoint***: Un *endpoint* es un punto en el que un sistema recibe pedidos desde otro sistema.

- ***Feedback***: *Feedback* o retroalimentación es el proceso por el cual un evento de salida es usado como elemento de entrada del sistema. En el contexto del proyecto, el *feedback* proporcionado por el cliente es utilizado como entrada para nuevas funcionalidades o cambios al sistema con el fin de proporcionar una mejor experiencia al usuario.

- **Figma**: Figma es una herramienta de generación de gráficos vectoriales y elaboración de prototipos.

- ***Frontend***: La capa de presentación es la capa del sistema que compone lo que "ve" el usuario, y la puerta de entrada con la cual interactuar con el sistema.

- **Git:** Sistema de control de versiones de código abierto.

- ***Git Flow***: Modelo de ramificado de Git, basado en tener dos ramas principales (*master* y *develop*) y el resto del trabajo en ramas particulares, con el fin de tener un flujo ordenado y controlado de trabajo.

- ***Hosting***: Servicio que provee a los usuarios un lugar de almacenamiento en línea.
- ***Insights***: Un *insight* se puede considerar como un entendimiento profundo de un problema o situación.
- **JIRA**: *Software* de gestión de proyectos del ecosistema Atlassian. Presenta interconectividad con Bitbucket y Confluence, y funciones como creación de *tickets* de trabajo y *boards* de varios formatos, entre ellos Kanban y Scrum.
- **Kanban**: Metodología de control de flujo de trabajo con el fin de mejorar los procesos de una empresa.
- **Mural**: Mural es un sistema que permite crear "pizarras" interactivas y trabajar en conjunto con un equipo, colocando pegatinas u otras figuras con el fin de armar un "mural" de información.
- ***Overhead***: Se considera como *overhead* en computación a un sobrecosto en el tiempo de computación o recursos del sistema.
- ***Pain***: Un *pain* o dolor es considerado como una de las causas que provocan problemas al cliente, generando de esta forma el problema completo que lo afecta.
- **People & Culture**: También escrita como P&C, es el área de la empresa Qualabs encargada del manejo de las personas y su bienestar en la empresa.
- ***Performance***: *Perfomance* mide qué tan efectivo es un producto de software en realizar una tarea en un periodo corto de tiempo con un número de recursos disponibles.
- ***Planning***: Ceremonia de Scrum en la que se planea un *sprint* y las tareas que se realizarán para el mismo ya sean de diseño/investigación o de desarrollo, tomando en cuenta los riesgos y una velocidad objetiva. Cuenta con la presencia del *Proxy Product Owner*.
- ***Product Owner***: El *Product Owner* es el dueño del producto, y fuente principal de funcionalidades para resolver el problema.
- **Prototipo:** Se entiende como prototipo a una representación bosquejada de una interfaz, intentando mostrar su esqueleto y el flujo a otras secciones.

- ***Proxy Product Owner***: El *Proxy Product Owner* (PPO) es el intermediario entre el equipo y el *Product Owner* en caso de que este no se encuentre disponible.

- **Qualabs**: Empresa cliente del proyecto.

- **RAM (memoria RAM)**: La memoria de acceso aleatorio o RAM es la memoria de la computadora que almacena información que un programa necesita para su ejecución.

- **Ramas**: Las ramas en git se utilizan para trabajar en paralelo en varias funcionalidades sin incidir en otras.

- ***Release***: Conjunto de funcionalidades que conforman una versión del *software* que es liberada al cliente.

- ***Retrospective***: La ceremonia de *sprint retrospective* es utilizada para validar el proceso y los aprendizajes al final de un sprint.

- ***Rollback***: Un *rollback* o regresión es un proceso por el cual una base de datos vuelve a un estado previo. Útil en el caso de que suceda una falla al actualizar o realizar un cambio en la base de datos.

- ***Script***: Guion utilizado para guiar una ceremonia de *sprint revie*w, revisión o prueba de usuario.

- **Scrum**: Marco de trabajo para la metodología ágil adoptado para este proyecto.

- ***Sprint***: Intervalo de tiempo en el que el equipo trabaja con el fin de finalizar un número establecido de trabajo.

- ***Story point***: Los puntos de historia o *story points* permiten expresar en una medida el "esfuerzo" estimado de una tarea en base a su complejidad y tamaño.

- ***Tradeoff**:* Compromiso entre mejorar un aspecto a coste de otro.

- **UI**: *User Interface* o Interfaz de Usuario es la portada que tiene el usuario para interactuar con el sistema.

- ***User Stories***: Una historia de usuario es una explicación informal de una funcionalidad, escrita desde la perspectiva del usuario final del sistema.

- **UX**: *User Experience* o experiencia de usuario es la forma en la que el usuario interactúa y experimenta el producto de Software.

Índice

1 Introducción 17
1.1 Descripción del equipo 17
1.2 Elección del proyecto 17
1.3 Descripción del cliente 18
1.4 Objetivos 18
1.4.1 Objetivos académicos 19
1.4.2 Objetivos del producto 20
1.4.3 Objetivos del proyecto 21
1.5 Estructura del documento 21
2 Problema 23
2.1 Contexto 23
2.2 Proceso para definir el problema 23
2.3 El problema 24
3 Solución 27
3.1 Descubrimiento de la solución 27
3.2 La solución 27
3.2.1 ¿Cómo surgió el nombre? 28
3.2.2 Funcionalidades principales 28
4 Marco de trabajo 34
4.1 Características del proyecto 34
4.2 Características del equipo 35
4.3 Metodología de trabajo 36
4.3.1 *Dual Track Scrum* 37
4.3.2 Ciclo de vida 38
4.3.3 Reuniones 39

4.3.4 Artefactos 43
4.4 Roles del Equipo 43
4.5 Roles externos 44
4.6 Conclusiones y lecciones aprendidas 45
5 Ingeniería de requerimientos 47
5.1 Proceso 47
5.1.1 Relevamiento 48
5.1.2 Especificación 53
5.1.3 Validación 53
5.2 Requerimientos funcionales 56
5.3 Requerimientos no funcionales 58
5.3.1 Usabilidad 58
5.3.2 Mantenibilidad 59
5.3.3 Seguridad 59
5.3.4 Disponibilidad 60
5.3.5 Restricciones 60
5.4 Entregables adicionales 60
5.5 Conclusiones y lecciones aprendidas 61
6 Arquitectura y Diseño 63
6.1 Descripción general de la arquitectura *Cloud* 63
6.1.1 Frontend 64
6.1.2 Backend Server 65
6.1.3 Base de datos 67
6.1.4 Bucket de imágenes 67
6.2 Decisiones de arquitectura 68
6.2.1 Frontend 68
6.2.2 Backend 71

6.3 Descripción general 74
6.3.1 Frontend 75
6.3.2 Backend 75
6.4 Atributos de calidad 77
6.4.1 Modificabilidad 77
6.4.2 Seguridad 81
6.4.3 Disponibilidad 84
6.4.4 Testeabilidad 87
6.4.5 Usabilidad 88
6.5 Conclusiones y lecciones aprendidas 92
7 Gestión del proyecto 94
7.1 Proceso 94
7.2 Gestión de *releases* 95
7.2.1 Proceso de liberación de *releases* 96
7.2.2 Plan de *release* 97
7.3 Feedback y su importancia en el proyecto 98
7.3.1 Gestión del *feedback* recibido 100
7.4 Principales hitos del proyecto 102
7.5 Gestión del esfuerzo 105
7.6 Métricas de gestión 108
7.7 Gestión de la comunicación 113
7.7.1 Comunicación interna 114
7.7.2 Comunicación con el cliente 114
7.7.3 Comunicación con el tutor 115
7.8 Gestión de riesgos 115
7.8.1 Identificación de los riesgos 116
7.8.2 Análisis 117

7.8.3 Respuesta 118
7.8.4 Seguimiento de los riesgos 118
7.9 Conclusiones y lecciones aprendidas 122
8 Gestión de la calidad 124
8.1 ¿Qué es la calidad para Ariadne's Thread? 124
8.2 Prácticas de aseguramiento de la calidad 125
8.2.1 Aplicación de estándares 125
8.2.2 Capacitaciones 129
8.2.3 Revisiones 129
8.2.4 Pruebas 131
8.3 Métricas 133
8.3.1 Mantenibilidad 133
8.3.2 Usabilidad 142
8.4 Conclusiones y lecciones aprendidas 146
9 Gestión de la configuración 148
9.1 Elementos de la configuración 148
9.1.1 Código Fuente 148
9.1.2 Documentación 155
9.2 Flujo de Trabajo 156
9.2.1 Ciclo de Vida de una tarea de investigación 156
9.2.2 Ciclo de Vida de una tarea de desarrollo 157
9.3 Conclusiones y lecciones aprendidas 157
10 Conclusiones 159
10.1 Conclusiones generales 159
10.1.1 Objetivos académicos 159
10.1.2 Objetivos del producto 161
10.1.3 Objetivos del proyecto 162

10.2 Lecciones aprendidas 162
10.3 Próximos pasos 164
11 Referencias bibliográficas 165
12 Anexos 174
12.1 *Design Sprint* 174
12.2 Investigación de ingeniería inversa 177
12.3 *Backlog* del Proyecto 180
12.3.1 *Release* 1 180
12.3.2 *Release* 2 183
12.3.3 *Release* 3 186
12.4 Evolución del plan de *release* 191
12.4.1 Primera versión 191
12.4.2 Segunda versión 192
12.4.3 Tercera versión 193
12.4.4 Cuarta versión 194
12.4.5 Quinta versión 195
12.5 Registro de riesgos 197
12.5.1 Riesgos por *sprint* 197
12.5.2 Evolución de los riesgos con seguimiento en todos los *sprints* 210
12.6 Heurísticas de Nielsen 216
12.7 *Tests* de usuarios 224
12.7.1 *Script* 224
12.7.2 Ejecución de pruebas 227
12.8 Encuesta de satisfacción 231
12.8.1 Resultados de la encuesta 231
12.8.2 Cálculo de métricas 233
12.9 Comentarios del cliente 236

12.10 Diferentes incidencias por *sprint* 239
12.10.1 Bugs 239
12.10.2 Feedback 242
12.11 Evolución del Sistema 244
12.11.1 Principales funcionalidades en la versión 1.0.0 244
12.11.2 Principales funcionalidades en la versión 2.0.0 252
12.11.3 Principales funcionalidades en la versión 3.0.0 263

1 Introducción

1.1 Descripción del equipo

El equipo está compuesto por cuatro estudiantes de la carrera Ingeniería en Sistemas:

- **Valentina Chalela:** estudiante de Ingeniería en Sistemas, actualmente trabajando como *Software developer* en Qualabs.
- **Keshet Hertz:** estudiante de Ingeniería en Sistemas.
- **Evgeny Sidagis:** estudiante de Ingeniería en Sistemas.
- **Patrick Silveira:** estudiante de Ingeniería en Sistemas, actualmente trabajando como *Software developer* en Qualabs.

Los integrantes nunca habían trabajado todos en conjunto, lo que presenta un desafío en cuanto a la confianza y coordinación de tareas a lo largo del proyecto. Sin embargo, dos de ellos venían trabajando en equipo a lo largo de la carrera y los otros dos habían comenzado a realizar trabajos obligatorios en conjunto desde hace un tiempo.

1.2 Elección del proyecto

Desde el comienzo el equipo tenía claro que quería un proyecto que lo desafiara y mantuviera motivado a lo largo del proceso, por lo que decidieron concurrir a la feria de proyectos de la Universidad ORT para poder conocer las propuestas que estaban ofreciendo diferentes empresas. Si bien en la feria había varias propuestas interesantes y desafiantes, el equipo no lograba coincidir en una en la que todos se sintieran motivados para realizarla, o estas no se ajustaban a la carrera o dimensiones del grupo.

Debido a esto, los integrantes que se encontraban trabajando decidieron consultar en sus empresas (que en ese momento eran distintas) si contaban con algún proyecto disponible para fin de grado, a lo que Qualabs respondió afirmativamente. La empresa ya había actuado como cliente de proyecto de fin de carrera para sus empleados en otras ocasiones y actualmente contaba con dos opciones para ofrecer, por lo que el equipo decidió reunirse con ellos para escuchar sus propuestas.

Lamentablemente estas no lograron interesar a todos los miembros del equipo dando lugar a posibles problemas de motivación en el transcurso del proyecto y por tanto decidieron evitarse.

Al escuchar esto, la empresa empezó a buscar si tenía algún otro proyecto para ofrecer. En una de las reuniones, el sector de People & Culture (área encargada del manejo de las personas y su bienestar en la empresa) planteó una problemática que estaban enfrentando en ese momento, la cual podía llegar a ser resuelta mediante un proyecto de *software*. Esto cautivó el interés del equipo dado que presentaba un gran desafío en varios aspectos.

En primer lugar, a diferencia de las otras opciones, este posible proyecto no se encontraba definido en absoluto y debía realizarse el proceso de definición desde cero, implicando un desafío a nivel de gestión.

Por otra parte, debían enfrentarse en el desarrollo de un sistema para un sector de la empresa, que no sólo no contaba con conocimientos técnicos, sino que además, nunca había tomado el rol del cliente en un proyecto de *software* (dado que en los casos anteriores era el área de ingeniería que proponía un proyecto y actuaba como cliente). Por lo tanto, el equipo iba a tener que trabajar en sus habilidades blandas para lograr comprender al cliente, sus ideas extremas, bajarlas a tierra, que este lo comprendiera y llevar a cabo negociaciones a lo largo del proyecto.

Finalmente, dado que el sistema iba a ser utilizado y mantenido por Qualabs, este definió las tecnologías de desarrollo y *hosting*, las cuales los integrantes tenían poca o ninguna experiencia y debían aprender sobre el transcurso del proyecto.

1.3 Descripción del cliente

Qualabs es una empresa uruguaya basada en la industria de *software* de video con 5 años de presencia en el mercado nacional e internacional. Se encarga de ayudar a distintas compañías a realizar integraciones de *software* con video, o escalar su capacidad de desarrollo, enfocándose en la velocidad de salida al mercado. Cuenta con múltiples clientes, principalmente de Estados Unidos, y actualmente con un total de 12 equipos de desarrollo, pero planean seguir creciendo a largo y corto plazo.

1.4 Objetivos

En esta sección se detallarán los principales objetivos que el equipo se propuso alcanzar en el proyecto. Estos se dividieron en diferentes secciones que son: académicos, del producto y del proyecto. Con el fin de que se pudieran evaluar y mantener la motivación para alcanzarlos, se

decidió utilizar la técnica SMART [1] para su definición, para que sean específicos, medibles, alcanzables, realistas y de duración limitada.

1.4.1 Objetivos académicos

Aplicar conocimientos de la carrera

- **Descripción:** Poner en práctica de manera exitosa los conocimientos adquiridos a lo largo de la carrera.
- **Medición de éxito:** Haber aplicado al menos el conocimiento de tres áreas diferentes de la Ingeniería de Software a lo largo del proyecto.
- **Propósito:** Poner en práctica los conocimientos generados a lo largo de la carrera en un proyecto real.

Puntaje de excelencia

- **Descripción:** Finalizar el proyecto con un puntaje de aprobación de excelencia.
- **Medición de éxito:** Lograr un puntaje de aprobación del proyecto igual o mayor a 95.
- **Propósito:** Generar un objetivo de aprobación alto para motivar a los miembros del equipo a generar un producto de calidad y un proceso acorde.

Aplicación del proceso de Ingeniería

- **Descripción:** Lograr adaptar el proceso de ingeniería seleccionado de forma que permita generar un proyecto que cumpla con las necesidades del cliente.
- **Medición de éxito:** Completar como mínimo el 85% de los requerimientos de máxima prioridad planteados por el cliente.
- **Propósito:** Ganar experiencia en los procesos seleccionados intentando potenciarlos para cumplir y de ser posible superar las expectativas del cliente.

Uso de nuevas tecnologías

- **Descripción:** Que el equipo aprenda tecnologías nuevas.
- **Medición de éxito:** Que cada integrante del equipo utilice al menos una tecnología que no haya utilizado anteriormente durante el desarrollo del proyecto.
- **Propósito:** Apuntar al crecimiento técnico de los integrantes del equipo en base a la utilización de tecnologías muy usadas en el mercado laboral.

1.4.2 Objetivos del producto

Cumplir con las expectativas del cliente

- **Descripción:** Que el cliente sienta que el equipo cumplió con sus expectativas y que el producto cumpla con sus necesidades.
- **Medición de éxito:** Obtener un mínimo de 70% de usuarios satisfechos en la encuesta de satisfacción entregada al cliente.
- **Propósito:** Generar un producto de calidad que resuelva los dolores del cliente y que cumpla con sus necesidades.

Buena usabilidad de la interfaz de usuario

- **Descripción:** Que el sistema final permita a los usuarios realizar sus tareas fluidamente.
- **Medición de éxito:** El 80% de los usuarios deberá poder utilizar el sitio web sin ningún tipo de capacitación ni ayuda.
- **Propósito:** Lograr generar una experiencia de calidad para el cliente final, generando así un producto útil para el cliente y a su vez experiencia valiosa para la carrera profesional de los integrantes del equipo.

Generar un producto de calidad

- **Descripción:** Generar un producto que funcione como el cliente espera.
- **Medición de éxito:** No tener o tener menos de 10 peticiones de arreglos (peticiones por *feedback)* en la versión final liberada.
- **Propósito:** Poder generar un producto que se adapte a su cultura, permita realizar sus procesos y cumpla, en lo posible, con la mayoría de sus expectativas.

Entregar un producto sin errores críticos

- **Descripción:** Que el sistema no posea errores críticos detectados al entregar, y un mínimo de errores no críticos. Se entiende por errores críticos aquellos que imposibilitan el uso del sistema.
- **Medición de éxito:** No tener errores críticos detectados al entregar, y un máximo de diez errores no críticos.

- **Propósito:** Generar un producto que el cliente pueda utilizar con confianza, teniendo el mínimo o ningún problema de funcionamiento.

1.4.3 Objetivos del proyecto

Asegurar la entrega continua al cliente

- **Descripción:** Los integrantes del equipo deben mantener un nivel de compromiso y esfuerzo alto a lo largo del proyecto.
- **Medición de éxito:** Llegar a un total de al menos 60 horas de trabajo realizadas en el área de desarrollo, por parte del equipo, en los *sprints* del proyecto.
- **Propósito:** Generar una medida de esfuerzo la cual mantenga al equipo siempre activo y generando valor para el cliente.

1.5 Estructura del documento

Problema

En este capítulo se muestra el contexto del problema, junto con el proceso para definirlo, y el problema en sí.

Solución

Este capítulo se enfoca en la solución, su descubrimiento, el origen del nombre del proyecto y las funcionalidades principales definidas para solucionar el problema.

Marco de trabajo

Este capítulo se enfoca en presentar el marco de trabajo del proyecto, presentando las principales decisiones relacionadas a la metodología de trabajo, y las características del equipo y del proyecto que justifican estas decisiones.

Ingeniería de requerimientos

En este capítulo se describe el proceso utilizado para la definición de los requerimientos funcionales y no funcionales del proyecto.

Arquitectura y diseño

En este capítulo se encuentran detalladas las principales decisiones arquitectónicas y de diseño tomadas para la construcción del sistema, algunas de las consideraciones que se tuvieron para realizarlas y cómo éstas satisfacen los atributos de calidad definidos para el proyecto.

Gestión del proyecto

Este capítulo muestra las actividades de gestión y planificación realizadas en el proyecto, mencionando cómo fueron algunos de estos procesos, también se describen los principales hitos y cómo estos afectaron en el proyecto. Finalmente se menciona cómo se realizó la gestión de la comunicación y la de riesgos.

Gestión de la calidad

En este capítulo se definen las acciones tomadas con el fin de asegurar la calidad del sistema, tomando como base lo establecido en ciertos objetivos del producto y algunos requerimientos no funcionales. También se detallan las métricas de calidad obtenidas a lo largo del tiempo y las conclusiones obtenidas de las mismas.

Gestión de la configuración

En este capítulo se identifican los diferentes elementos que componen la configuración del proyecto, tomando en cuenta el control de versiones y ramificado. Agregando, se muestra el flujo de trabajo utilizado para los tipos de tareas del proyecto.

Conclusiones

En este capítulo se muestran las conclusiones y aprendizajes obtenidos respecto a todo el proyecto en sí, tomando en cuenta los objetivos definidos previamente.

2 Problema

2.1 Contexto

Antes de describir el problema, es necesario conocer brevemente cómo funciona Qualabs y sus valores, para poder valorar la trascendencia que tiene para la empresa, y la necesidad de obtener una solución.

Qualabs es una *software factory* especializada en video, cuyo enfoque se basa en equipos dedicados de desarrolladores, apoyados en un proceso de mejora continua, en el que intentan llevar el desarrollo a otro nivel, con la finalidad de dar un servicio de calidad y brindar valor a sus clientes. Es importante destacar, que no solamente buscan dar valor al cliente, sino también impulsar el desarrollo de sus empleados, siendo esta la base de su cultura, para lo cual, tienen como objetivos la satisfacción y el crecimiento de las personas dentro de la empresa. Para lograrlo, crearon un área de soporte que se encarga de acompañar el desarrollo de las personas llamada People & Culture (en adelante P&C). Si bien podría pensarse como la típica área de "Recursos Humanos", intentan que sea más que solo un manejo del personal, incorporando en su proceso también el crecimiento de las personas y la cultura.

Se basan en establecer un plan de carrera para sus empleados en dónde estos puedan definir objetivos para su desarrollo, y tengan instancias de *feedback* con su líder o acompañante cada cuatro meses. En estas, reciben comentarios sobre la evolución y desempeño que se ha percibido hasta el momento y se evalúan los objetivos establecidos, para que puedan refinarlos o definir nuevos, en base a lo que quieren lograr. Se sigue este proceso con el fin de que la persona pueda impulsarse, crecer y lograr sus metas. Un punto importante a destacar, es que este crecimiento puede ser tanto en el área profesional, tecnológica o personal, ya que creen fuertemente en el bienestar personal para el mejor desarrollo de las personas y de la empresa; siendo la comunicación, colaboración y transparencia puntos claves para lograrlo.

2.2 Proceso para definir el problema

Llegar a la definición del problema llevó su trabajo, dado que el cliente se había percatado del mismo poco antes de las primeras reuniones con el equipo y recién se lo estaba planteando. Como se mencionó en la sección "Elección del proyecto" del capítulo anterior, todo surgió en base a un comentario de un integrante de P&C en una de las reuniones iniciales del equipo con

el cliente, que resultó en la generación del proyecto. Sin embargo, debido a que todo esto surgió sin un anterior planteamiento y pienso del cliente respecto al problema, este debía analizar sus procesos para detectar sus falencias. Por tanto, el equipo debió llevar a cabo una serie de reuniones con el cliente para descubrir y entender las causas de sus problemas, comprender sus procesos y su cultura, e intentar definir soluciones para resolver el problema correctamente.

Por lo expuesto, el hecho de que el cliente no tuviera completamente definido el problema, presentó un desafío adicional a los ya mencionados en el capítulo anterior.

2.3 El problema

En 2021 Qualabs se propuso como objetivo aumentar su personal, planeando duplicar su tamaño para finales del año. Sin embargo, estando en pleno crecimiento y con planes de seguir creciendo, detectaron que el proceso de gestión de personas, equipos y proyectos de P&C no era escalable, generando grandes problemas y pérdidas de tiempo, al aumentar el número de empleados. De no encontrar una solución, sería muy difícil organizar eficientemente el gran volumen de información para proporcionar un crecimiento controlado, exitoso y sostenido en el futuro, y la cultura de la empresa ya no podría prosperar.

En ese momento, la información de los empleados y equipos se encontraba dispersa en diferentes lugares, herramientas o en la cabeza de las personas. Los diferentes datos de sus empleados, ya fueran personales o referentes a la empresa, se encontraban en múltiples plataformas tales como Microsoft Office y Asana. Para visualizar los equipos dentro de la organización utilizaban la aplicación Mural en la cual realizaban manualmente un esquema de la estructura de la empresa en ese momento, situando a los diferentes empleados dentro de sus equipos de trabajo. Sin embargo, esta herramienta solo les brindaba una ayuda visual con figuras y títulos, no podían acceder rápidamente o almacenar información relevante sobre los elementos en pantalla, que sería de utilidad para las operaciones de la empresa. Además, este diagrama era propiedad del sector de P&C, y no se encontraba accesible para otros sectores de la organización, que hubieran podido beneficiarse utilizándolo en sus propios procesos.

Otra parte de la información se encontraba en la cabeza del personal de P&C, generando una dependencia hacia personas particulares o hacia dicho sector, a la hora de querer obtener información específica sobre empleados o equipos.

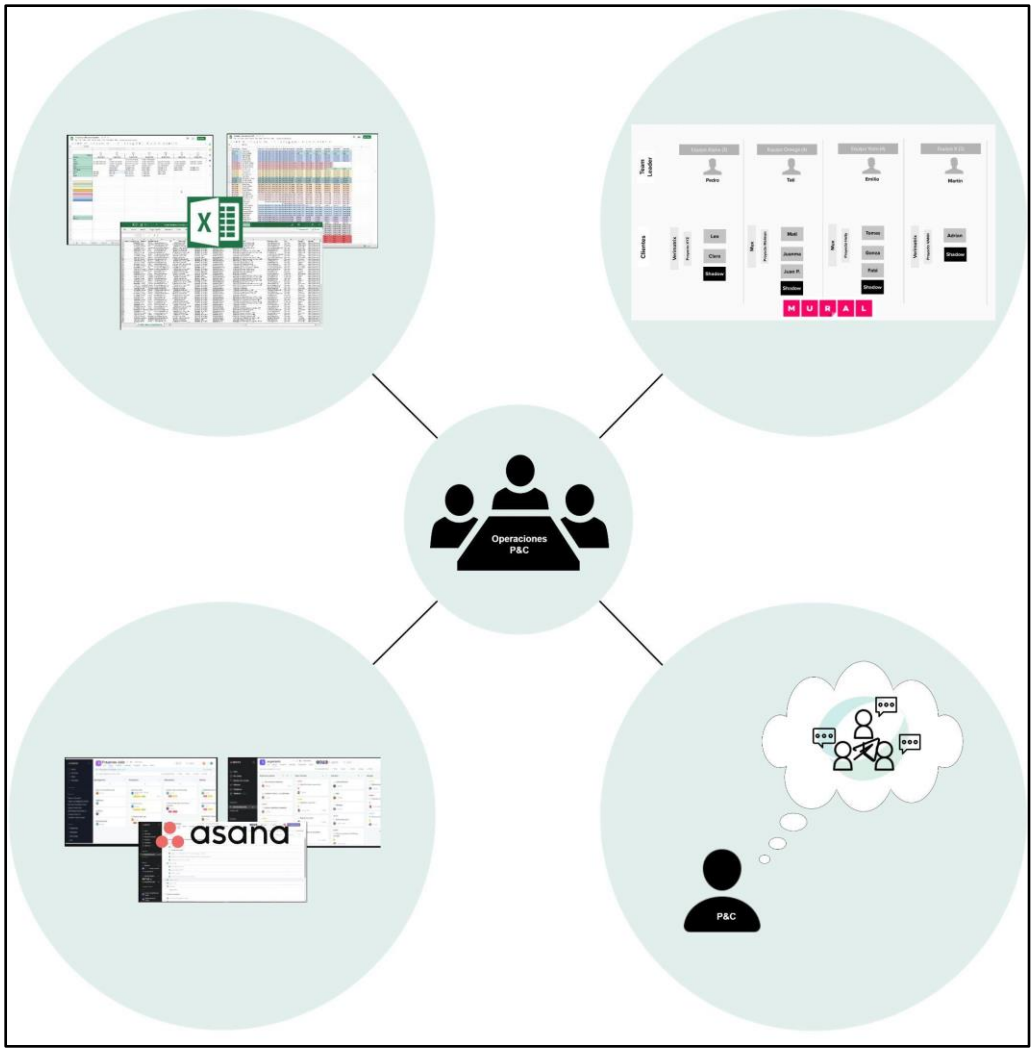


Figura 2.1: Representación de la distribución de la información para los procesos de P&C

Con el crecimiento de la empresa vieron que este mecanismo se estaba volviendo cada vez más lento, engorroso y cada vez más **difícil visualizar la estructura y cambios de la empresa**, generando **grandes pérdidas de tiempo**, **datos** y **frustraciones**. Además, comenzaron a notar que ya no era tan sencillo recordar todos los detalles para el desarrollo de las personas o requerimientos del negocio, haciendo que algunos **datos importantes** se **terminaran perdiendo** u olvidando, y con ello, se perdía la marca diferencial de la empresa.

Por lo tanto, para lograr crecer de la mejor manera, sin perder el foco en la cultura y la singularidad de las personas, necesitaban un sistema que les permitiera:

- Visualizar y comprender fácilmente la estructura actual de la organización.
- Tener la información organizada y almacenada correctamente de manera que pueda ser rápidamente accesible para la toma de decisiones estratégicas en la organización.

- Visualizar el estado actual de la organización con sus requerimientos de negocio y necesidades para dar lugar a discusiones de acciones de mejora y cambios a realizar para cumplir con estos y entregar valor.
- Compartir la información de la estructura de equipos y personas, de manera que los diferentes sectores de la organización puedan visualizarla, comprenderla y utilizarla sin depender de un área en específico para esto.

3 Solución

En este capítulo se narrará brevemente el proceso seguido para el descubrimiento y definición de la solución propuesta, finalizando con una descripción de sus principales funcionalidades.

3.1 Descubrimiento de la solución

En el transcurso para la definición del problema, el equipo empezó a comprender algunos aspectos de los procesos y de la cultura de la empresa, que fueron tomados como base para la creación de la solución. Lograr una primera versión de la solución llevó varias iteraciones de encuentros con el cliente, negociaciones y redefiniciones (este proceso se encuentra en mayor detalle en el capítulo de “Ingeniería de requerimientos”).

Al inicio, el equipo se había enfocado en el problema de escalado y gestión de los datos, inclinando la solución hacia sistemas de gestión de recursos humanos, y realizando ingeniería inversa sobre sistemas de este tipo. Sin embargo, avanzado el proceso, y habiendo conocido más al cliente luego de varios intercambios, el equipo se percató de que eso no era lo que estaban buscando. El cliente quería lograr un crecimiento empresarial, preservando la cultura, e impulsando a las personas para que crezcan dentro de la empresa, y por tanto el sistema debía poder permitirlo. Con el fin de poder lograrlo, el sistema se fue construyendo en conjunto, haciendo que el *software* fuese evolucionando en base a las devoluciones recibidas, para adaptarse a su visión y necesidades.

De esta forma, se fue generando un sistema que permite preservar la integridad de las personas y potenciar el desarrollo individual, junto con el crecimiento empresarial, sin caer en el manejo de las personas como si fueran números. Otro punto por el cual fue vital este proceso de co-construcción, es porque el cliente no quería que el sistema implicara un esfuerzo de aprendizaje elevado. Por tanto, era importante intentar realizar flujos sencillos y respetar la visual a la que estaban acostumbrados -en la medida de lo posible-, para que su uso no generase un sobreesfuerzo cognitivo.

3.2 La solución

Se construyó una aplicación web la cuál, es el centro para las operaciones de P&C, en donde pueden acceder a la información de la empresa o dirigirse a información relacionada, y es

posible visualizar claramente la estructura organizacional con sus datos. También pueden compartir el conocimiento con otras áreas (para que se pueda generar un entendimiento común), y se pueden crear vistas con cambios hipotéticos en los equipos, permitiéndoles tomar decisiones más asertivas de negocio. Ello lleva a que el proceso sea más rápido, ordenado y pueda ser compartido dentro de la empresa, eliminando la dependencia de personas o áreas (además de eliminar la sobrecarga mental por almacenar estos datos en la cabeza).

3.2.1 ¿Cómo surgió el nombre?

Luego de aprobado el proyecto por la Universidad (con su bosquejo de requerimientos iniciales), pero todavía sin un nombre definido, el equipo le mencionó al cliente que debían establecer un nombre para el sistema. Luego de múltiples intercambios, a Sofía (cabeza de P&C) se le ocurrió el nombre *Ariadne's Thread* (el hilo de Ariadna), porque así como ocurre en el mito griego [2], ella sentía que estaba atrapada en un problema sin solución, y el equipo mediante el sistema en desarrollo le estaba brindando el "hilo" y el camino para salir de allí. Tanto el equipo, como el resto del grupo del cliente estuvieron de acuerdo, luego de entender el significado que reflejaba el mismo, y así nació el nombre del sistema.

3.2.2 Funcionalidades principales

En esta sección se mencionan algunas de las funcionalidades que aportan mayor valor a la solución, con una breve descripción para que el lector se pueda hacer una idea del sistema y contrastar con los problemas antes mencionados. La lista de los requerimientos funcionales se encuentra en su respectiva sección dentro del capítulo de "Ingeniería de requerimientos".

Nota: Estas funcionalidades mencionadas son parte de la solución final, sin embargo no todas se encontraban en la primera versión de la solución y fueron surgiendo en el desarrollo.

Diagrama organizacional interactivo

Es la pantalla principal del sistema (también llamado *Dashboard*), muestra la diagramación de todos los equipos y personas, que están activos, en forma de columnas y tarjetas (respetando la visual del diagrama de Mural que utilizaba antes P&C para este objetivo). Los elementos en pantalla se pueden clickear para acceder a información acerca de los mismos. Se adicionó la capacidad de edición que permite mover a las personas de equipo, ver personas sin equipos y

agregarlas a uno mediante movimientos de *Drag & Drop*, y cambiar el rol de una persona dentro de un equipo, para poder actualizar el estado de la empresa en tiempo real.

Posiciones vacantes

Cuando un cliente solicita un proyecto, también establece la cantidad de personas que desea que trabajen en el mismo. Luego, en base a esto, la empresa calcula y establece el número de personas que va a asignar a ese equipo (para permitir licencias del personal, sin dejar de cumplir con el número requerido por el cliente). El sistema permite registrar el número contratado por el cliente en los proyectos, y el establecido por la empresa en los equipos (ya que cada equipo se dedica a un proyecto). Estos números son utilizados en el *Dashboard* para generar automáticamente tarjetas de posiciones vacantes, en base a la diferencia con la cantidad de personas del equipo. Estas tarjetas tienen un código de color dependiendo de su importancia, para destacar a simple vista si la posición que se está necesitando es para cubrir los requerimientos del cliente, o para llenar los puestos que la empresa desea tener. Esto le permite al usuario detectar rápidamente la situación y pensar en estrategias para el negocio.

Categorías customizables

Se pueden crear y actualizar categorías, las cuales tienen nombre, color y una lista de etiquetas, y pueden ser asignadas a las diferentes entidades. A través de estas, se puede añadir información adicional a los empleados, equipos y proyectos según se necesite, dando una mayor flexibilidad al sistema. Se encuentran visibles en los perfiles de las entidades y en el *Dashboard*, pudiendo además en este último filtrar los elementos en base a estas para visualizar diferentes aspectos del negocio rápidamente.

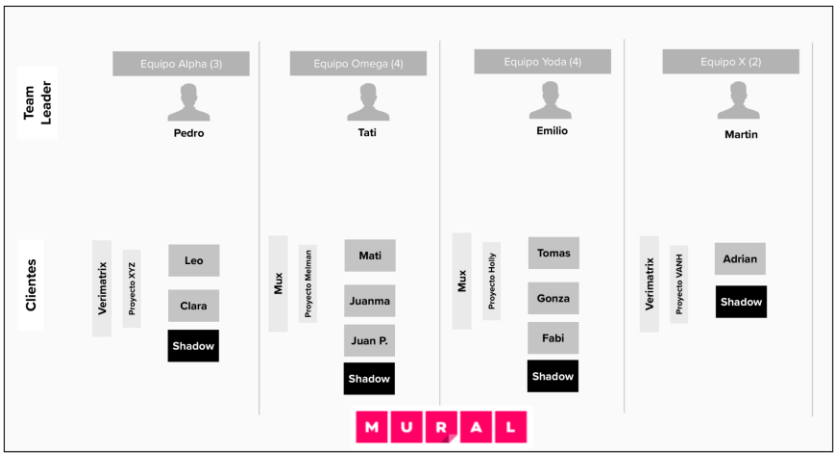


Figura 3.1: Recreación del diagrama de equipos y personas que utilizaba P&C

En la Figura 3.1 se muestra el Mural que utilizaba P&C para visualizar los equipos y empleados de la organización, teniendo solamente tarjetas con nombres (o apodos) de los desarrolladores, el nombre del cliente y el proyecto, y un título con el nombre del equipo y cantidad de personas contratadas por el cliente. Los *shadows* que se observan son los recursos extra que la empresa desea incluir en el equipo, para cubrir con los números del cliente y contemplar las licencias. Todo esto se realizaba mediante un armado manual y solo les brindaba lo que se ve en el diagrama, no cuenta con información extra. En caso de quererla debían recordarla o buscarla en los otros sistemas.

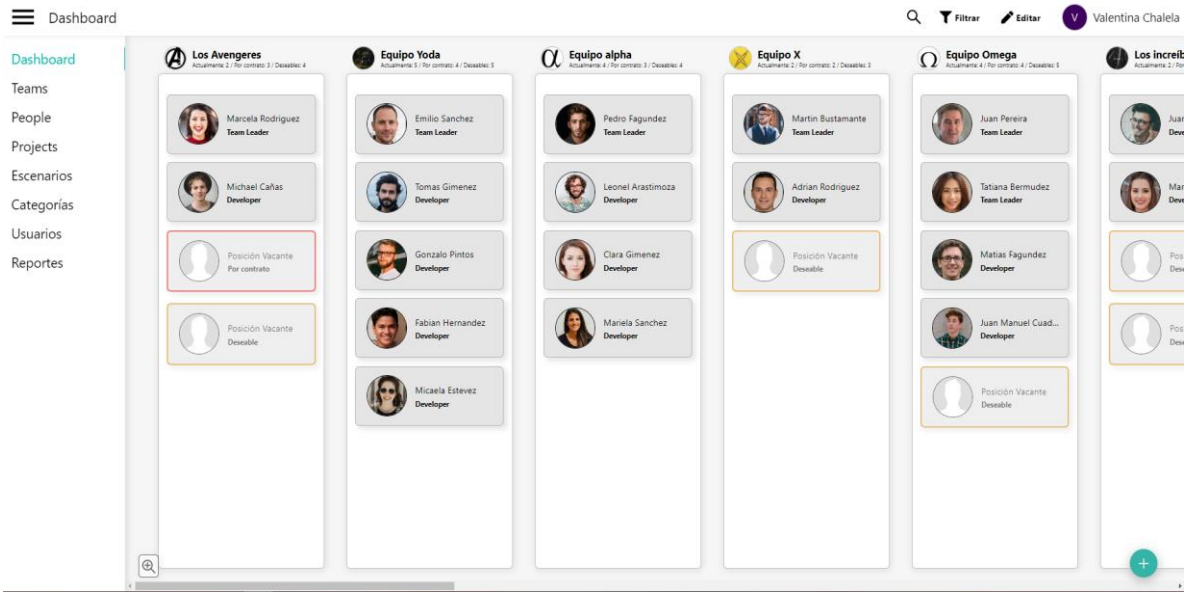


Figura 3.2: Diagrama organizacional interactivo (pantalla principal del sistema)

La Figura 3.2 muestra la versión final de la pantalla principal del sistema, si bien se conserva la estructura de columnas y tarjetas (conforme a lo solicitado) tiene mayor cantidad de información. El título de las columnas se conforma por el nombre del equipo y su logo, junto con la cantidad de empleados actuales, cantidad contratada por el cliente y la deseada por la empresa (solicitado así por Qualabs). Dentro de cada equipo se pueden ver las tarjetas de los desarrolladores teniendo su nombre, apellido, rol y foto, y además las posiciones vacantes diferenciadas por color en base a las necesidades del proyecto y la empresa. También se puede cambiar el orden de los equipos según se desee para personalizar la vista.

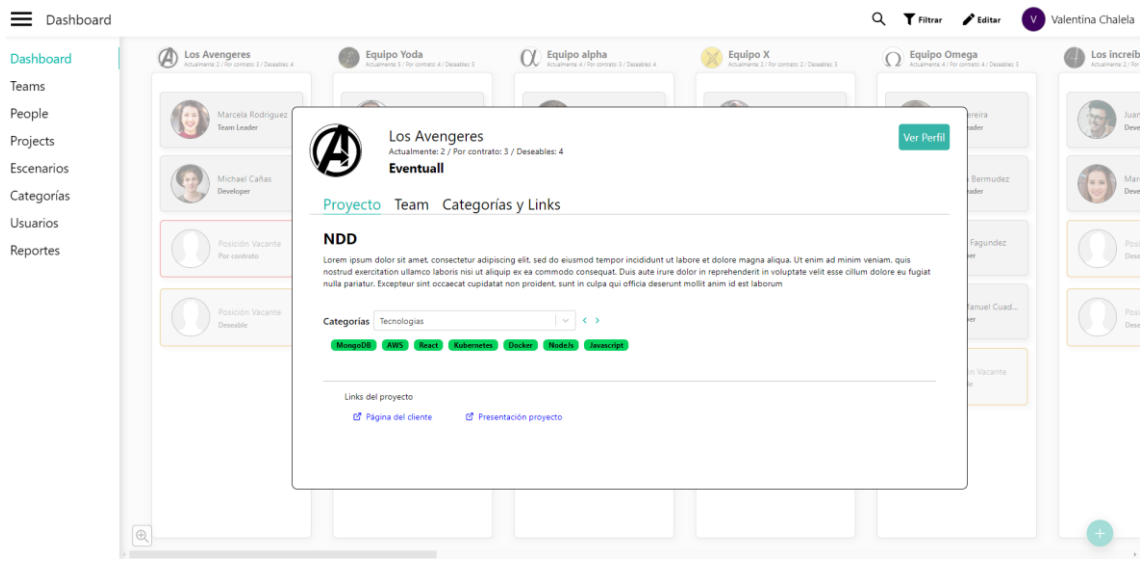


Figura 3.3: Tarjeta desplegable de información del equipo

En la figura 3.3 se muestra como ejemplo la tarjeta de información de un equipo. Al clickear en las personas o equipos se abre una tarjeta con breve información de la entidad, permitiendo además acceder al perfil completo de estos en caso de querer ver más información.

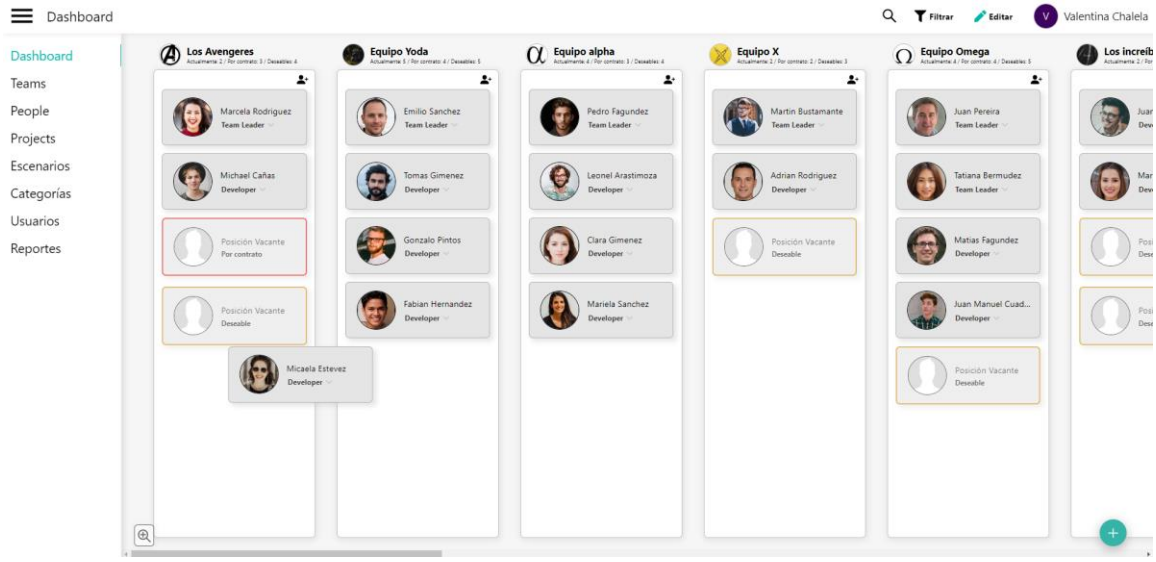


Figura 3.4: Movimiento de persona entre equipos

La figura 3.4 muestra el modo edición (al cual se accede haciendo *click* en el ícono de editar), en este se pueden actualizar los elementos del tablero. En el ejemplo se observa el cambio de equipo de una persona mediante un movimiento de *Drag & Drop*. Adicionalmente, las personas pueden ser cambiadas de equipos mediante sus tarjetas de información, cambiando en el selector de equipo el equipo asignado.

Escenarios interactivos

Es un espacio en donde se pueden crear diferentes diagramaciones de empleados y equipos en base a la información actual de la empresa (tomando el *Dashboard* como lienzo), al que se le puede poner nombre y guardar, para poder luego volver a ver estas ideas generadas. Además, permite ver los cambios que presenta el escenario generado respecto al modelo inicial de partida, desglosados por equipos, y explicitando si fueron adiciones, eliminaciones o modificaciones de sus integrantes.

Sistema de roles

Dado que personas de diferentes posiciones dentro de la organización pueden utilizar el sistema, se crearon distintos roles para permitir el acceso organizado y seguro a la información. Se crearon específicamente 3 roles, puesto que así fue solicitado por el cliente, en el entendido de que una mayor diversidad de roles podría perjudicar la transparencia que buscan en este sistema.

Estos son:

Administrador: Se encarga de administrar todo el sistema, puede gestionar usuarios y categorías además de realizar todas las funcionalidades del sistema.

Asistente: Pueden realizar todas las funcionalidades del sistema menos las referentes a manejo de categorías y usuarios. Este rol será utilizado por el personal de P&C y otras áreas (como operaciones) que ayudan a gestionar los movimientos de personas y equipos, y toman decisiones estratégicas. Dado que la empresa está creciendo en todas sus áreas y no solo en desarrolladores, este rol fue necesario para que todos puedan gestionar los equipos y personas, dejando el manejo del sistema en sí, al personal de mayor experiencia dentro de la organización.

Empleado: Son los desarrolladores de la empresa, que pueden ingresar al sitio para actualizar su información, y ver la diagramación actual de la misma (con menor cantidad de información que los otros usuarios). Esto hace que sea más fácil mantener el sistema actualizado, libera al personal de P&C del trabajo de pasar y actualizar estos datos, además de permitir que los desarrolladores puedan ver a los otros equipos existentes en la empresa. Esta funcionalidad fue agregada hacia el final del proyecto, y fue valorada muy positivamente por la empresa.

4 Marco de trabajo

En el siguiente capítulo se verán detallados todos los elementos utilizados a lo largo del proyecto para conformar el marco de trabajo.

4.1 Características del proyecto

Para seleccionar la metodología de trabajo a utilizar se analizaron las diferentes características del proyecto, a continuación, se detallan los aspectos más relevantes para la toma de esta decisión.

Cliente cuenta con un área de ingeniería

Si bien se tenía el desafío de que el área de P&C, interesado principal e integrante en las negociaciones, no posee conocimientos técnicos, la empresa cuenta con un área de ingeniería la cual se encarga de gestionar los aspectos técnicos de la empresa. Esta área fue la encargada de establecer las tecnologías a utilizar en el proyecto, mostrándose dispuesta a brindar apoyo en decisiones técnicas importantes que impacten en el producto, dado que serían los encargados del mantenimiento del sistema luego de finalizado el proyecto.

Disponibilidad del cliente

La empresa se mostró disponible a lo largo del proyecto, siendo de gran apoyo para este. Se interesó en seguir el proceso de construcción, ser parte del mismo, e ir probando y dando *feedback* sobre el estado del sistema. Si bien el cliente se encontraba muy ocupado con sus obligaciones laborales, trataban de que al menos un recurso estuviera disponible para interactuar con el equipo.

Requerimientos abiertos

Si bien había algunos requerimientos que estaban inicialmente definidos, ya que establecían las bases del sistema, no se tenía una definición total de estos ni la certeza de cuál iba a ser el conjunto total de los mismos. Además, la empresa se encontraba en plena evolución, haciendo cambios también en sus procesos y por tanto era muy probable que los requerimientos “ya establecidos” fueran mutando, aparecieran nuevos e incluso algunos fueran eliminados. Sin olvidar que el sistema tiene una fuerte orientación al usuario, y por tanto se debe ser flexible

para poder contemplar y realizar los posibles cambios solicitados. Todo esto hace que se tenga que tener una gran apertura al cambio, organización, coordinación y comunicación para poder lograr estos puntos.

Conocimiento del dominio

El equipo no contaba con conocimientos del dominio o procesos del cliente, y si bien el cliente tenía este conocimiento, no tenía claros sus problemas o las magnitudes de estos. Esto hizo que el equipo comenzará un proceso para entender el contexto del cliente, descubrir sus *pains*, investigar sobre el problema, comprender cómo realizaban sus procesos, para poder generar posibles opciones para resolverlos. Luego se intentaban transmitir de forma eficiente estas soluciones, dando lugar a instancias de negociación con el cliente para mostrarles lo generado, escuchar *feedback* de su parte y llegar a un conjunto de funcionalidades que les aportaran valor.

Contexto

El proyecto empezó bajo un marco de virtualidad debido a la pandemia de COVID-19, lo cual presentó tanto un desafío como una ventaja. Como desafío, representó una desconexión dentro del grupo y del grupo con la empresa, por lo que el equipo tuvo que intensificar la comunicación constante. Al mismo tiempo, presentó una ventaja dado que se pudo ahorrar tiempo de traslado ya sea desde la universidad o trabajo, y usarlo para la comunicación o para realizar el proyecto.

4.2 Características del equipo

También se tuvieron en cuenta, a la hora de tomar estas decisiones, las siguientes características del equipo.

Tamaño del equipo

El equipo cuenta con cuatro integrantes, los que, al inicio en función de sus conocimientos, se dividieron a la mitad entre *backend* y *frontend,* con el fin de promover la eficiencia del desarrollo aprovechando los puntos fuertes de cada uno. Sin embargo, fue necesario coordinarse en las tareas, y en algunos casos, “cambiar de posición” para cumplir con lo propuesto. Esto se debió principalmente a que el desarrollo era incierto y no siempre eran equitativas las cargas de trabajo de estas áreas.

Desconocimiento de las tecnologías necesarias

Dado que las tecnologías requeridas para el proyecto eran establecidas por el cliente y los miembros del equipo no contaban con conocimientos ni experiencia trabajando con ellas, se vio necesario investigar y dedicar tiempo a aprender las tecnologías desde cero (exceptuando por un integrante que presentaba experiencia laboral respecto a la tecnología usada en el *frontend* del sistema).

Capacidad de trabajo

Todos los miembros del equipo poseían obligaciones por fuera del proyecto, tanto por trabajo o estudio, por lo que se veía necesario mantener una buena comunicación, manteniéndose organizados a la hora de realizar tareas y buscando un ritmo de trabajo que fuese lo más consistente posible.

4.3 Metodología de trabajo

Debido a los factores descritos previamente, se eligieron las metodologías ágiles, dado que permiten la flexibilidad necesaria para lidiar con el cambio constante de los requerimientos, dando la posibilidad de priorizar el trabajo según lo que se necesite, y permitiendo negociar los requerimientos con el cliente en las diversas reuniones establecidas.

No obstante, quedan un par de factores a tomar en cuenta; la investigación, y el diseño. El equipo no contaba con conocimientos sobre las tecnologías a utilizar y por lo tanto debían aprenderlas, haciendo necesaria la investigación para el desarrollo del proyecto. Además, dado que el proyecto está orientado al usuario y pretende ser utilizado casi diariamente por la empresa, se vio necesario poner foco en el diseño, crear prototipos y validarlos para que el sistema se adapte y cumpla con las expectativas y necesidades del cliente.

Por lo tanto, con esto, y que el producto no estaba del todo definido, se denotó que el proyecto iba a tener gran potencial de investigación de descubrimiento de producto y diseño. Por estas razones, es que se decidió utilizar la adaptación de Scrum llamada *Dual Track Scrum* [3], [4].

4.3.1 *Dual Track Scrum*

Dual Track Scrum es una ampliación de Scrum en la que se le agrega al desarrollo una etapa de diseño, quedando definidos dos *tracks* o carriles en paralelo llamados *Discovery* (descubrimiento) y *Delivery* (en español denominado como desarrollo) [5]. Esto permite realizar una gestión eficiente de ambos procesos, otorgando gran valor para el usuario final ya que se dedica tiempo al armado y validación de diseños a la vez que se realiza el desarrollo.

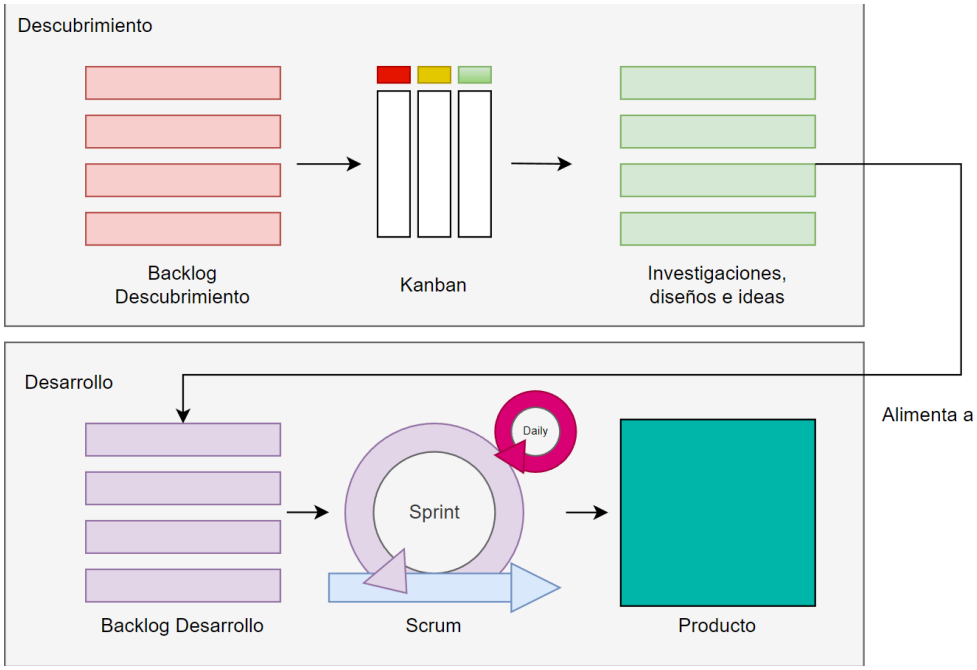


Figura 4.1: *Dual Track Scrum*

En el carril de descubrimiento se diseña, realizan prototipos, testean soluciones, y se validan con el cliente. También se realizan investigaciones. En el de desarrollo se toman los aprendizajes obtenidos y se utilizan para el desarrollo del producto final. Este proceso permitió que el equipo se sintiera más seguro en el desarrollo, ya que se contaba con un entendimiento de las necesidades del cliente y lo que se debía desarrollar, maximizando el aporte de valor al cliente, y posibilitando que el resultado final sea igual o lo más aproximado a lo esperado, además de reducir el retrabajo de correcciones.

Para llevar a cabo la organización de las tareas del proyecto se utilizó la plataforma de JIRA, en dónde se crearon dos proyectos, uno de investigación y otro de desarrollo para llevar de forma ordenada las diferentes tareas. Sin embargo, por la naturaleza de los diferentes carriles se utilizaron diferentes formatos para cada uno de ellos. El carril de descubrimiento contó con

un formato Kanban, dado que es un escenario más volátil y es probable que entren o salgan tareas en el proceso. Para el otro carril, se definió un tablero de Scrum ya que es más organizado, se priorizan tareas, se lleva un conjunto definido de tareas a realizar en el *sprint* y el *sprint* siguiente y se debe entregar un incremento de valor al cliente al finalizar.

Design Sprint

Dado que se pretendía lograr una interfaz que le resulte cómoda y atractiva al cliente se decidió utilizar *Design Sprint* [6], que es una metodología ágil diseñada por Google Ventures. El proceso original consta de cinco días, en donde se va construyendo mediante la investigación, innovación y bocetos la interfaz, hasta que finalmente se prueba con los usuarios. Por cuestiones de tiempo, el equipo adaptó el proceso a tres reuniones. Esto llevó a que cada diseño fuera planeado, prototipado y validado por el cliente antes de empezar a programar, obteniendo *feedback* para realizar cambios de ser necesario, y ahorrando tiempo de retrabajo, además de incluir al cliente en el proceso de diseño.

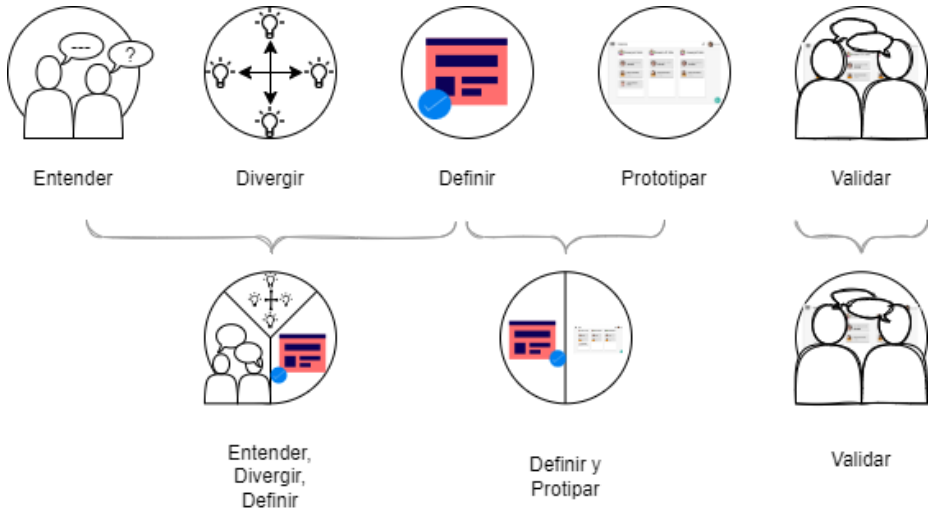


Figura 4.2: Diagrama de la adaptación de *Design Sprint*

4.3.2 Ciclo de vida

El ciclo de vida seleccionado fue el iterativo incremental, dado que es el ciclo de vida con el que trabaja Scrum [7]. Al inicio se realizó una fase de descubrimiento de producto y configuraciones, en donde el equipo se centró en realizar diversas investigaciones (tanto metodológicas como de tecnologías) para sentar las bases del proyecto, tomar los *insights* de la fase inicial de ingeniería de requerimientos y profundizarlos con más conocimientos del

negocio, para generar una base inicial sobre las necesidades del cliente. Esto resultó en una primera versión de la solución con requerimientos y posible arquitectura.

Establecido esto, se comenzó la fase de desarrollo, iniciando el funcionamiento de ambos *tracks* en paralelo. En el *track* de descubrimiento se realizaban investigaciones, diseños de las pantallas y validaciones con el cliente. Los resultados, en caso de ser exitosos, se utilizaban como *input* para el desarrollo. En el *track* de desarrollo, se tomaban los aprendizajes del descubrimiento y se los utilizaba para el desarrollo, en el que se trabajaba en iteraciones con el fin de generar incrementos de valor al cliente.

Figura 4.3: Aplicación de *Dual Track Scrum*

4.3.3 Reuniones

Como base del proceso, se tomaron las ceremonias de Scrum *(planning, daily, retrospective* y *review*), adicionando luego una intra-equipo para mejorar la organización y eficiencia (*backlog refinement*).

También se mantenían reuniones para el *Design Sprint* (diseñado, validado) y otras con el cliente y parte del equipo para la gestión del proyecto (por ejemplo: refinamiento, priorización o descubrimiento de requerimientos).

Backlog Refinement

En esta reunión [8] se juntan el *Proxy Product Owner* y la *Project Manager*, para seleccionar, refinar y agregar detalles como una descripción o tamaño a las historias que pueden ser

agregadas al *sprint*. Estas reuniones fueron realizadas previas a un *sprint*, o durante un *sprint* en el caso de que surjan nuevos requerimientos.

Sprint Planning Meeting

Previo a la reunión, el *Proxy Product Owner* y la *Project Manager* generaban el *backlog* con Qualabs, seleccionando posteriormente tareas del *backlog* para el *sprint*, por medio de *Backlog Refinement*. En la reunión de *sprint planning*, se discutían los posibles riesgos que podrían afectar el *sprint*, se establece una velocidad objetivo y luego se estimaba cada tarea por medio de la técnica de *Planning Poker* [9] utilizando la escala de Fibonacci. Finalmente, se generaba un objetivo para el *sprint*, y se daba comienzo al mismo.

Daily Meetings

Por temas de tiempo y obligaciones de los miembros del equipo, las *Daily Meetings* se realizaban los Martes y Jueves, con la posibilidad de realizarse otros días en base a las necesidades del proyecto. Asimismo, el equipo se mantuvo actualizado por medio de herramientas como Whatsapp. Esta reunión ayudaba a mantener la cohesión del equipo, y conocer lo que cada miembro estaba haciendo, planeaba hacer, y si precisaban ayuda o si tenían algún comentario respecto al proceso que se estaba llevado. Los resultados de estas reuniones eran mantenidos en un canal de Discord con el fin de que quedara un registro para realizar un seguimiento, o también para que un integrante que no pudiera asistir pudiera ver lo sucedido.

Sprint Review

La *Sprint Review* era una reunión realizada al final de cada iteración de Scrum. La misma era llevada a cabo en conjunto con el cliente (siempre que este tuviera disponibilidad), en la que participaban varios integrantes/interesados de diferentes áreas, cómo el *Product Owner*, miembros de P&C, el representante de ingeniería, entre otros; con el fin de que pudieran ver el avance del proyecto.

Previo a la reunión, el equipo revisaba lo que fue hecho en el *sprint* ya sea historias o *feedback*, y organizaba lo que iba a mostrar y decir por medio de un *script* de demo. Dentro del *script* se separaba en secciones los cambios provenientes de *feedback* y los que no, con el fin de enfatizar el origen de estos en la presentación, y para que el cliente sienta que sus propuestas fueron escuchadas. Con esto se logró mantener un flujo ordenado durante la s*print review*, evitando

dejar funcionalidades afuera. Respecto al *feedback*, se muestran los cambios hechos en base a estos, para que el cliente pueda comentar y ver cómo quedaron los cambios que fueron solicitados.

Estas reuniones resultaron extremadamente valiosas para el equipo, dado que mediante estas se logró mantener una instancia para recibir *feedback* de manera constante, logrando un sistema más útil, y por medio de esta reunión el cliente se podía adentrar en la construcción del sistema.

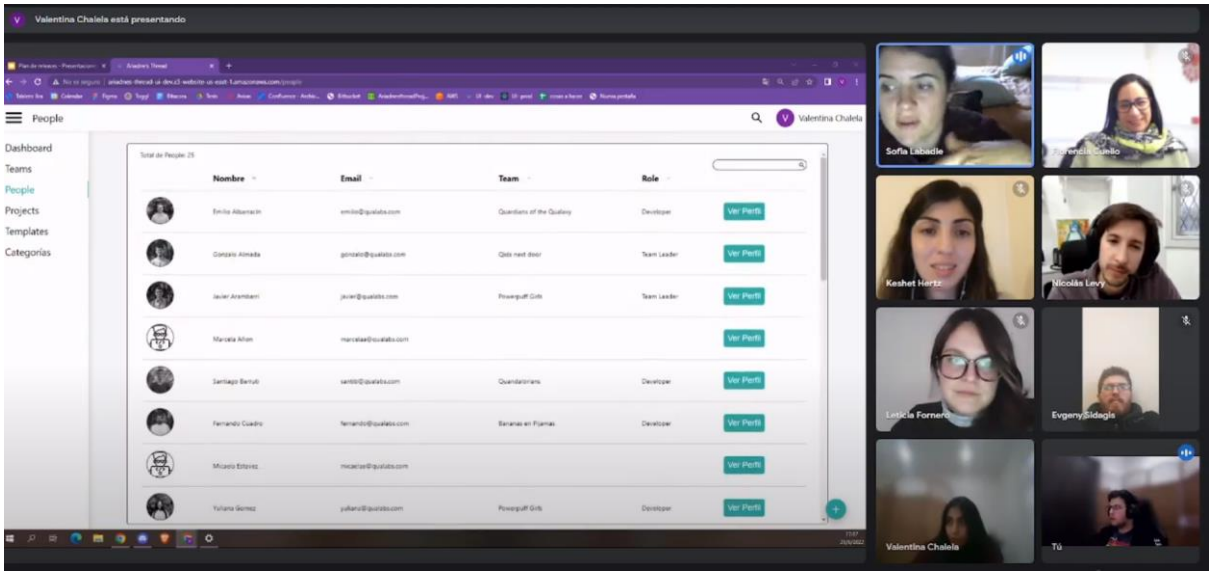


Figura 4.4: Imagen de una *sprint review* con el cliente

Sprint Retrospective

Las retrospectivas se utilizaban para validar el proceso y los aprendizajes obtenidos al final del *sprint*, anotando posibles cambios, mejoras, aspectos que el equipo debería mantener, y una evaluación de cada miembro en base a su desempeño y el desempeño del equipo. Al mismo tiempo, el equipo analizaba si llegó al objetivo de trabajo para el *sprint*, evaluando también el resultado obtenido en cuanto a las historias realizadas y el esfuerzo, luego el equipo discutía y anotaba posibles acciones a tomar con el fin de generar mejoras a futuro.

Para esto, se empezó a usar un *board* en Miro, dividiendo en distintas columnas y en base a tarjetas lo que se podría mantener, mejorar y agregar.

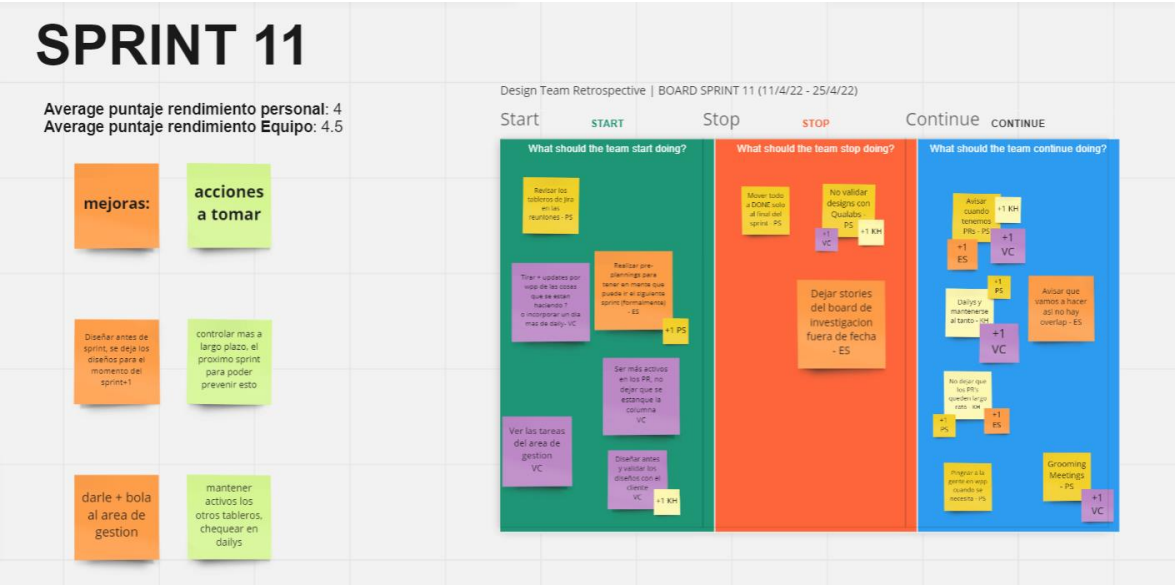


Figura 4.5: Ejemplo de *retrospective* en Miró

Design Meeting

Las *meetings* de diseño eran utilizadas por el equipo para generar las interfaces de usuario en base a las necesidades del cliente. Como se mencionó en la sección de "*Design Sprint*" de este capítulo, debido al tiempo acotado del proyecto, se adaptó el proceso original de cinco días de esta metodología a tres reuniones. En las dos primeras reuniones se realizaba el proceso establecido originalmente a realizar en los primeros cuatro días, y la tercera reunión se llevaba a cabo con el cliente para validar y testear el prototipo (correspondiente al último día del *Design Sprint*). Para más detalles sobre la transformación de las instancias de *Design Sprint* entre reuniones propias, ver la Figura 4.2 en este capítulo.

Los prototipos fueron generados utilizando la herramienta de diseño Figma. Para ver más información respecto a los prototipos, dirigirse al anexo "*Design Sprint*".

Validación de Interfaz con el Cliente

Como reunión final de *Design Sprint*, dos miembros del equipo se reunían con uno o más miembros del equipo de P&C, para validar si el prototipo de interfaz del cliente era de su agregado, y si no lo era, el prototipo era modificado, y luego validado nuevamente. En caso de no poder realizarse esta reunión, se tomaba parte del final de la reunión de *sprint review* para realizar esta validación con todo el equipo presente.

4.3.4 Artefactos

Product Backlog

El *Product Backlog* contiene las historias con un número de identificación. Se utilizó la herramienta JIRA para gestionarlo y actualizarlo conforme surgían nuevas tareas. Las tareas eran identificadas en base a épicas a las que pertenecían.

Sprint Backlog

El *sprint backlog* contiene las historias seleccionadas para un *sprint*, estimadas en *Story Points*. En cada *ticket* de JIRA se detallan casos de uso y/o criterios de aceptación, junto con una descripción. Consideramos a un *ticket* como una *user story* o parte de una.

En el *board,* los *tickets* tienen 5 estados:

- *To Do*: *Tickets* prontos para ser empezados. Cuando un integrante toma el *ticket* le agrega su identificación.
- *In Progress*: *Ticket* en progreso.
- *Code Review*: El *ticket* está abierto a *peer review* por medio de *Pull Requests* en *Bitbucket.*
- *Testing:* El *ticket* se encuentra en el ambiente de desarrollo, listo para ser probado en este ambiente.
- *Done:* El *ticket* se encuentra completado y funcionando en el ambiente.

4.4 Roles del Equipo

La asignación de roles tomó como base las preferencias y fortalezas de cada miembro del equipo. A su vez, los roles se dividieron entre roles de Scrum y roles por Afinidad.

Valentina Chalela

- *Scrum Master*: Se asegura que la utilización de Scrum y sus adaptaciones sean correctas.
- *Project Manager:* Gestión del proyecto y proceso de trabajo, se reúne con el *Proxy Product Owner* para refinar el *backlog*.

- *Tech Lead:* Encargada de tomar las decisiones importantes del sistema, desarrollar y supervisar el desarrollo.

Patrick Silveira

- *Proxy Product Owner*: Media entre el cliente y el equipo, se encarga de elicitar requerimientos y genera un canal de comunicación entre estos.
- *Frontend Lead:* Liderazgo en la toma de decisiones respecto al *frontend.*
- *Cloud Architect*: Se encarga de mantener los repositorios, ambientes, *deployments* y *pipelines*.

Keshet Hertz

- *User Experience/User Interface Designer*: Encargada de diseñar la experiencia de usuario y las interfaces, incluyendo la creación y mantenimiento de prototipos.

Evgeny Sidagis

- *Backend Lead*: Encargado de las investigaciones y el desarrollo de *backend.*
- *Quality Assurance Engineer*: Encargado de mantener las pruebas, calidad y criterios de aceptación.

Roles transversales

En adición a los roles mencionados anteriormente, los cuatro miembros del equipo tomaron los roles de Desarrolladores y de *Testers*.

4.5 Roles externos

Como roles externos, se asignaron roles a los diferentes actores principales de Qualabs, en base a sus puestos y afinidad, siendo estos: Juan Pablo Saibene como *Product Owner,* Nicolás Levy como Asistente de Ingeniería; y Sofía Labadie como principal cliente. Dado que Juan Pablo Saibene es el *Chief Executive Officer* de Qualabs, él no podía estar presente todo el tiempo, por lo que se decidió crear el rol de *Proxy Product Owner.* Lo mismo aplicaba a Sofía Labadie, por lo que fue reemplazada por Lucia Lizarazu (integrante del área de P&C) en las ocasiones en las

que no podía asistir. Finalmente, si bien las principales clientes eran estas últimas, todo el área de P&C interactuó con el equipo, dándole *feedback* valioso desde varios puntos de vista.

En el siguiente organigrama, se muestran los roles y jerarquías dentro de Qualabs. Las líneas negras marcan a los miembros que estuvieron involucrados desde el inicio del proyecto, y las líneas verdes marcan los miembros que se fueron involucrando a lo largo del proyecto.

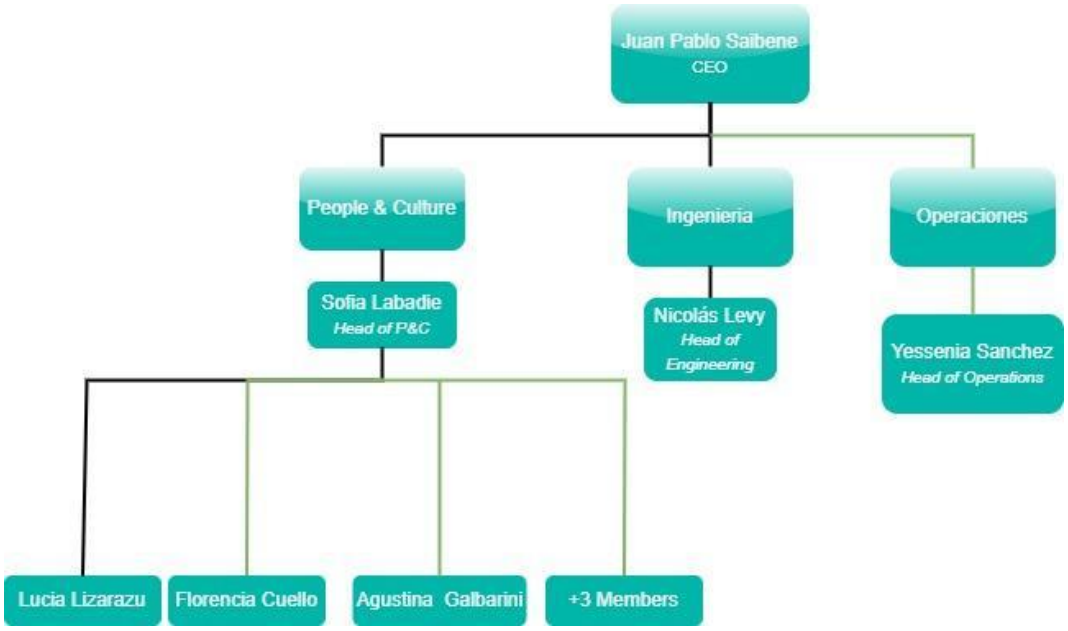


Figura 4.6 Organigrama de Qualabs

4.6 Conclusiones y lecciones aprendidas

El primer aprendizaje que se lleva el equipo es la utilización de *Dual Track Agile*, lo cual supuso un proceso de aprender a cómo utilizarlo, y también cómo balancear los dos *tracks* de trabajo, de forma tal en que no se sobrecargue de investigaciones y diseños el *track* de descubrimiento, pero tampoco desestimarlo totalmente. El uso de esta adaptación de Scrum fue una gran oportunidad para aprender algo nuevo, y utilizar algo útil para un proyecto con un fuerte componente de investigación y diseño, junto con desarrollo.

Otro punto importante que fue aprendido es la interacción con el cliente, y cómo incluirlo, y escucharlo durante el proceso. Las interacciones con el cliente fueron ricas en lo que respecta al proyecto, y también como un aprendizaje para el equipo en sí, pudiendo “sacarse el miedo” inherente que una persona puede tener de discutir con el cliente los distintos aspectos del proyecto.

Para finalizar, la comunicación entre el equipo, tanto en las *dailies* como en las *sprint retrospectives* fue un factor decisivo para mantener el ritmo e implementar mejoras de forma continua al proceso. Por medio de estas últimas, el equipo detectó falencias, procedimientos para continuar realizando, y mejoras. Como ejemplo, gracias a estas reuniones se pudo agilizar el proceso de *Code Review*, o el agregado de la reunión de *Backlog Refinement*, con el fin de tener listos un set de *tickets* para hacer en un *sprint* en particular.

5 Ingeniería de requerimientos

En este capítulo se detallará el proceso de Ingeniería de Requerimientos que el equipo siguió a lo largo de todo el proyecto. Se expondrá el proceso utilizado para la especificación y validación, en conjunto con las herramientas utilizadas, finalizando con los requerimientos funcionales, los no funcionales y las restricciones.

5.1 Proceso

Debido a que gran parte del equipo no conocía al cliente, y los que sí lo hacían no contaban con conocimiento sobre sus procesos de negocio ni del funcionamiento del departamento de P&C, este proceso comenzó con una etapa de reuniones informales. Estas primeras reuniones tenían como objetivo que ambas partes se conocieran y que el cliente comenzara a explicarle al equipo los diferentes procesos de P&C, cómo se realizaban y qué impedimentos tenían. Las mismas fueron claves para el desarrollo del producto dado que hasta ese punto no se tenía claridad sobre qué sistema realmente necesitaban, ni cuáles eran sus mayores *pains*. Esta fue una etapa que necesitó de numerosas reuniones para recabar toda la información posible, convertirla en necesidades y problemas, para bosquejar un conjunto inicial de funcionalidades a realizar.

Luego de establecido el primer conjunto de funcionalidades, se comenzó una etapa de múltiples iteraciones en las que se fue generando y refinando el plan de desarrollo del sistema. En dicha etapa el equipo se centró en realizar intercambios con el cliente en los cuales se mostraban los requerimientos generados, para que este pudiese validarlos o pedir cambios para los mismos. A su vez, estas interacciones con el cliente se utilizaban para priorizar requerimientos, permitiéndole así al equipo generar un plan de acción priorizado.

Una vez acordado el primer conjunto de funcionalidades, el equipo comenzó el proceso de desarrollo del sistema. A lo largo de dicho proceso se mantuvieron diversas reuniones con el cliente para validar los resultados generados (en su mayoría como *reviews* de Scrum[1]), recibir *feedback* de los mismos, y especificar más detalladamente los requerimientos que serían implementados en el siguiente ciclo de desarrollo.

[1] Para más información sobre el proceso de *reviews* de Scrum dirigirse a la sección de “Metodología de trabajo” del capítulo “Marco de trabajo”.

Debido a que el cliente quería que el sistema estuviese fuertemente adaptado a su cultura y a la forma en la que desempeñaban sus procesos, se comprometió a concurrir a diversas reuniones para poder aportar su visión y conocimiento del negocio. Esta presencia tan activa del cliente potenció la validación de requerimientos permitiéndole al equipo comprender las necesidades y prioridades de P&C, generando así que los incrementos realizados tuviesen gran valor para el negocio.

La participación del cliente a lo largo del proceso fue clave para el desarrollo del producto, ya que fue aportando su conocimiento y experiencia, resultando en una co-creación del sistema en conjunto con el equipo. El proceso de desarrollo llevado a cabo consistía en generar pequeños incrementos de valor en cada iteración[2]. Estos eran evaluados por el cliente en las reuniones de *sprint review* al final de cada ciclo, permitiendo así que este pudiese solicitar cambios o brindar ideas de cómo mejorar el producto. Dichos cambios o mejoras eran tomados en cuenta por el equipo, tanto para realizar mejoras a las funcionalidades que ya se encontraban implementadas, como para mejorar las que serían desarrolladas en futuras iteraciones. Este continuo proceso de *feedback*[3] generó que poco a poco los integrantes del equipo pudiesen comprender cada vez mejor el negocio, provocando que estos mismos comenzaran a proponer mejoras al cliente que podrían aportarles valor.

El flujo continuo de retroalimentación llevado a cabo con el cliente hizo que las reuniones fueran cada vez más provechosas en cada nueva iteración, generando negociaciones mucho más ricas, e incluso, en numerosos casos, propuestas de funcionalidades por parte del equipo que otorgaron gran valor al cliente.

5.1.1 Relevamiento

El proceso de relevamiento de los requerimientos del sistema fue realizado principalmente por los integrantes Valentina Chalela (*Project Manager*) y Patrick Silveira (*Proxy Product Owner*). Estos fueron los responsables de reunirse con el cliente con el objetivo de recabar información sobre la parte del negocio que necesitaba apoyo, escuchar los deseos e ideas que este tenía, e intentar comprender los verdaderos *pains* que afectaban al cliente.

[2] Este proceso se ve detallado en la sección de "Dual Track Scrum" dentro del capítulo de "Marco de trabajo".

[3] Para más detalle sobre el proceso de *feedback* llevado a cabo en el proyecto dirigirse a la sección "*Feedback* y su importancia en el proyecto" del capítulo "Gestión del proyecto".

Debido a que el tamaño del equipo es reducido, se tomó la decisión de realizar esta división con el fin de promover la productividad, haciendo así que se pudieran relevar los requerimientos al mismo tiempo que se mantenía el desarrollo del sistema. Esto permitió que el equipo se mantuviera continuamente aportando valor al cliente, para que este pudiese tener una sensación de que su sistema se encontraba en constante crecimiento.

Durante el transcurso del proyecto el relevamiento de los requerimientos fue evolucionando, notándose una diferencia en la manera en la que se realizaba, los objetivos de cada reunión y los resultados de estas. Estos cambios pueden agruparse dentro de tres etapas, las cuales fueron: “Descubrimiento del Producto”, “Desarrollo y Evolución” y “Cierre del Proyecto”.

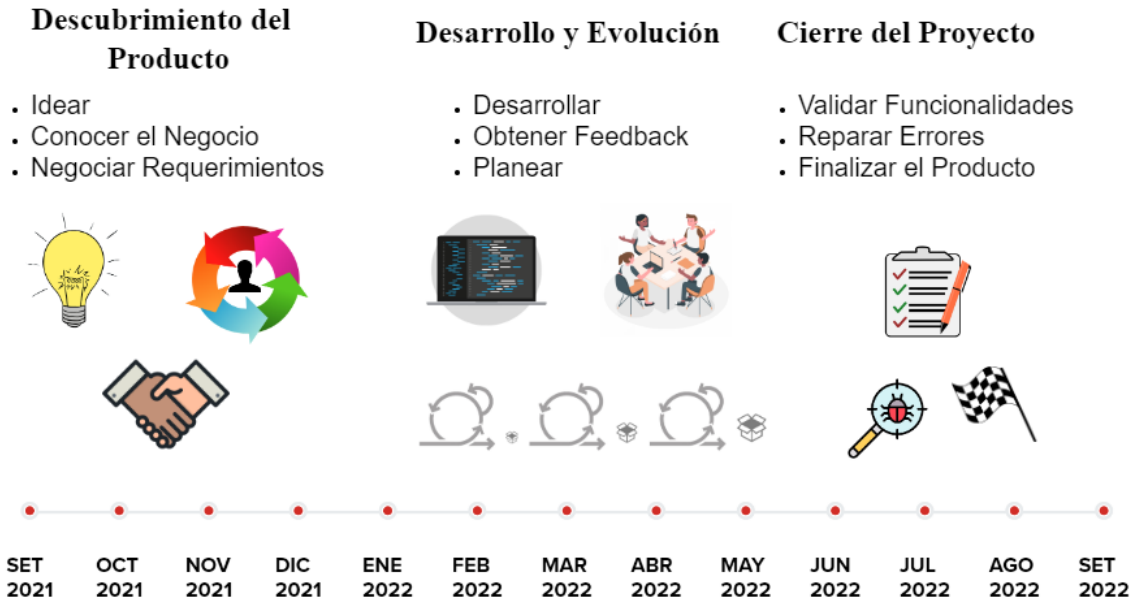


Figura 5.1: Etapas del proceso de relevamiento de Ingeniería de Requerimientos

Etapa de Descubrimiento del Producto

Esta fue la etapa inicial del proyecto en la cual el equipo todavía no había tenido mucho contacto con el cliente, el producto no se encontraba totalmente definido por el mismo, y se buscaba llegar a un mutuo entendimiento de cuál era el problema que se iría a atacar y qué funcionalidades se crearían para solucionarlo. Esta etapa se caracteriza por haber tenido un fuerte foco en el descubrimiento, ya que el objetivo era lograr definir un plan para el desarrollo del sistema.

Al inicio del proyecto se utilizó la técnica de entrevistas, comenzando con estas sin una estructura marcada, con el fin de facilitar el diálogo y la obtención de conocimientos. Estas eran llevadas a cabo en reuniones en las cuales participaban diversas personas del lado del cliente. Entre los participantes de las mismas se encontraban Juan Pablo Saibene (CEO), Nicolás Levy (Director de Ingeniería), Sofía Labadie (Directora de P&C) y Lucía Lizarazu (integrante de P&C). El objetivo principal de dichas entrevistas era que los diferentes integrantes de Qualabs le contaran al equipo los impedimentos con los que se habían encontrado en el departamento de P&C, ideas que pudiesen tener para solucionarlos, y cómo eran los procesos que estaban llevando a cabo. Todo esto con el fin de que el equipo pudiera comprender los procesos de negocio, sus detalles clave y los posibles puntos de mejora para lograr construir un *software* personalizado para ellos que resolviera estos problemas.

Otra técnica utilizada por el equipo fue la Ingeniería Inversa[4], ya que se consideró necesario realizar una investigación sobre sistemas existentes, para conocer productos similares en el mercado y qué funcionalidades tenían los mismos. El objetivo de esto era comprender dónde estaría posicionada la solución, ver qué aspectos de otros sistemas podrían ser imitados para solucionar los problemas del cliente, y entender los aspectos que diferenciarían el producto a desarrollar de las soluciones ya existentes.

En conjunto con las técnicas anteriormente mencionadas se realizaron lluvias de ideas para crear posibles funcionalidades que podrían ser agregadas al proyecto, comenzando a diseñar con estas una primera versión de la solución. El uso de esta técnica fue muy útil en etapas tempranas del proyecto dado que permitió al equipo divergir con los conocimientos adquiridos del uso de las otras técnicas, generando así múltiples ideas las cuales luego se convertirían en posibles funcionalidades para el sistema.

Finalmente, en esta etapa inicial se utilizó el prototipado (ver anexo "*Design Sprint*") luego de generado un conjunto de posibles requerimientos para la solución, con el objetivo de darle mayor visibilidad al cliente de cómo se verían -a grandes rasgos- algunas de las funcionalidades clave, y a su vez, dar lugar a posibles cambios y negociaciones sobre estas, para que pudieran adaptarse correctamente a los procesos del departamento de P&C.

[4] Se puede ver un resumen de esta investigación en el anexo "Investigación de ingeniería inversa".

Etapa de Desarrollo y Evolución

Luego de finalizada la etapa inicial del proyecto se comenzó con el proceso de desarrollo, siguiendo igualmente con el descubrimiento en paralelo. En esta etapa el equipo ya contaba con un plan inicial de épicas que conformarían el sistema, sin embargo las mismas no contaban con demasiado detalle. Esto provocó que los integrantes del equipo debieran reunirse periódicamente con el cliente para refinar las historias a realizar en iteraciones cercanas del ciclo de desarrollo, y a su vez, validar nuevamente con este si dichas historias seguían siendo válidas (ya que la empresa estaba en constante evolución). El objetivo de esta etapa era asegurar el valor a entregar en cada incremento del desarrollo e ir preparando el *backlog* para las futuras iteraciones.

Esta etapa volvió a contar con entrevistas, sin embargo, las mismas se realizaban en su mayoría al final de las iteraciones de desarrollo en la reunión de *sprint review*. A estas concurrían normalmente Juan Pablo Saibene, Nicolas Levy e integrantes del departamento de P&C (dependiendo de su disponibilidad). Dicha reunión tenía como objetivo mostrar al cliente el incremento de desarrollo realizado, obtener *feedback* sobre el mismo y crear discusiones o negociaciones sobre futuras funcionalidades a desarrollar en las próximas iteraciones. Al comenzar la reunión, el equipo mostraba el plan de *release*, haciendo énfasis en lo que se había completado hasta el momento y qué cosas se desarrollarían en el futuro cercano. Esto era clave para darle visibilidad al cliente sobre el avance del proyecto dado a que los participantes de esta reunión oscilaban con frecuencia. Luego, el equipo realizaba una demostración de las funcionalidades implementadas en la última iteración de desarrollo, que al finalizarse, el cliente proseguía a brindar *feedback*, el cual era anotado para su posterior análisis. Para finalizar dicha reunión, el equipo realizaba una entrevista semi estructurada en la cual se le realizaban un conjunto de preguntas al cliente para aclarar puntos sobre funcionalidades a realizar en futuros *sprints*. Estas eran muy útiles para recabar información sobre cómo se podrían mejorar los procesos, entender la visión que tenía el cliente sobre las funcionalidades futuras y verificar que el equipo hubiese comprendido correctamente su objetivo.

En el caso de que el equipo tuviese dudas sobre lo que se iba a desarrollar, y que las mismas no pudiesen ser solucionadas en la reunión de *sprint review* por falta de tiempo o necesidad de mayor preparación, se generaba una reunión extra a la que concurrían solamente los participantes necesarios. Estos eran, del lado del equipo, el PPO y la PM, y del lado del cliente, los integrantes que contaban con conocimiento de lo que se quería atacar. Cabe destacar que

estas entrevistas comenzaron sin tener una estructura definida para su realización, pero a lo largo del tiempo se fue generando una en base al *feedback* del cliente, y al deseo del equipo de optimizar el uso del tiempo. El proceso generado consistía en crear un archivo previo a la realización de la reunión, en el que se describían los puntos a tratar en la misma junto a preguntas claves. Este era enviado con un par de días de antelación al cliente para que el mismo pudiese comprender lo que se trataría en la entrevista y pudiera prepararse.

Tanto para ayudar al cliente a visualizar fácilmente algunas de las funcionalidades clave para el producto, como para que el equipo pudiese obtener *feedback* sobre las mismas previo a su desarrollo, se utilizó la técnica de prototipado. Esta permitió al equipo hacer cambios tanto de usabilidad como de alcance en base a las opiniones recibidas del cliente. Esta técnica fue especialmente eficaz cuando el cliente no se encontraba del todo seguro sobre cómo debería funcionar el sistema en ciertos casos. El cliente se podía apoyar en estos para comprender lo que se estaría desarrollando, podría imaginarse los casos de uso que haría con la funcionalidad, y entender así qué ventajas y desventajas tendría para su uso el diseño generado.

Etapa de Cierre del proyecto

Al finalizar el segundo de los tres *releases* prometidos al cliente el equipo comenzó la etapa de cierre del proyecto. En esta, el equipo se centró en desarrollar las últimas funcionalidades que entrarían dentro del alcance del proyecto, y a su vez realizar la mayor cantidad de cambios provenientes de *feedbacks* del cliente que no se hubiesen solucionado hasta el momento. También se comenzó a analizar las funcionalidades que se habían dado por finalizadas, en búsqueda de encontrar cambios sencillos de realizar que pudiesen aportarle valor en términos de usabilidad o funcionalidad al cliente.

En esta etapa se utilizaron nuevamente las entrevistas, siendo estas en su mayoría llevadas a cabo dentro de la reunión de *sprint review*, y en menor medida en reuniones de refinamiento. Las mismas tenían como principal objetivo obtener y analizar *feedback* de parte del cliente. Esto fue dado a que era una forma muy sencilla de encontrar errores o mejoras en las funcionalidades desarrolladas, y a su vez generaba en el cliente un sentimiento de pertenencia y co-creación del producto. Al entrevistarlo, se pudieron encontrar muchos detalles en las funcionalidades ya finalizadas que podrían ser mejorados y que eran sencillos de realizar. Muchos de estos surgieron gracias al uso periódico de la aplicación que estaba realizando el cliente. Este uso del sistema por parte del cliente hizo que fuera más sencillo para el equipo

determinar qué flujos de trabajos estaban bien pulidos y cuales podrían necesitar mejoras, que siempre que no fuesen muy complejas, se atacaban al poco tiempo de ser detectadas.

Otra técnica que fue utilizada fueron las lluvias de ideas. Estas se utilizaron con el principal propósito de generar posibles cambios que fuesen fáciles de realizar, pero que pudiese aportar valor al cliente. Las lluvias de ideas eran realizadas tras conseguir *feedback* del cliente, analizando el mismo y generando en base a este, ideas que pudiesen ser de ayuda. Muchas de las funcionalidades que se agregaron en el cierre del proyecto fueron provenientes del uso de esta técnica, o una consecuencia indirecta del uso de la misma.

5.1.2 Especificación

Los requerimientos del proyecto eran especificados en forma de *user stories* [10], escribiéndose desde la perspectiva del usuario para que fuese más claro el valor que aportaba. Cada una de estas describía una funcionalidad específica que se desarrollaría dentro del sistema, pudiendo separarse las mismas en múltiples tareas que luego serían ejecutadas por el equipo.

Para definir cada una de las historias de usuario el equipo establecía un título, una descripción y criterios de aceptación. El título de la historia debía ser descriptivo de la funcionalidad que se estaría desarrollando, la descripción de la misma se definió utilizando el formato de **Como** <Rol> **Quiero** <Objetivo> **Para** <Beneficio>, y para los criterios de aceptación el equipo decidió utilizar lenguaje natural para definirlos con claridad.

5.1.3 Validación

Como se mencionó en la sección de “Relevamiento” de este capítulo, la validación estuvo presente en todas las etapas del proyecto. Esto fue en gran parte gracias a la estrecha relación formada con el cliente y las múltiples reuniones mantenidas con el mismo. En estas se obtuvieron *feedbacks* e *insights* del negocio, se logró que el producto final cumpliese con las expectativas del cliente, e incluso superándolas en algunos casos.

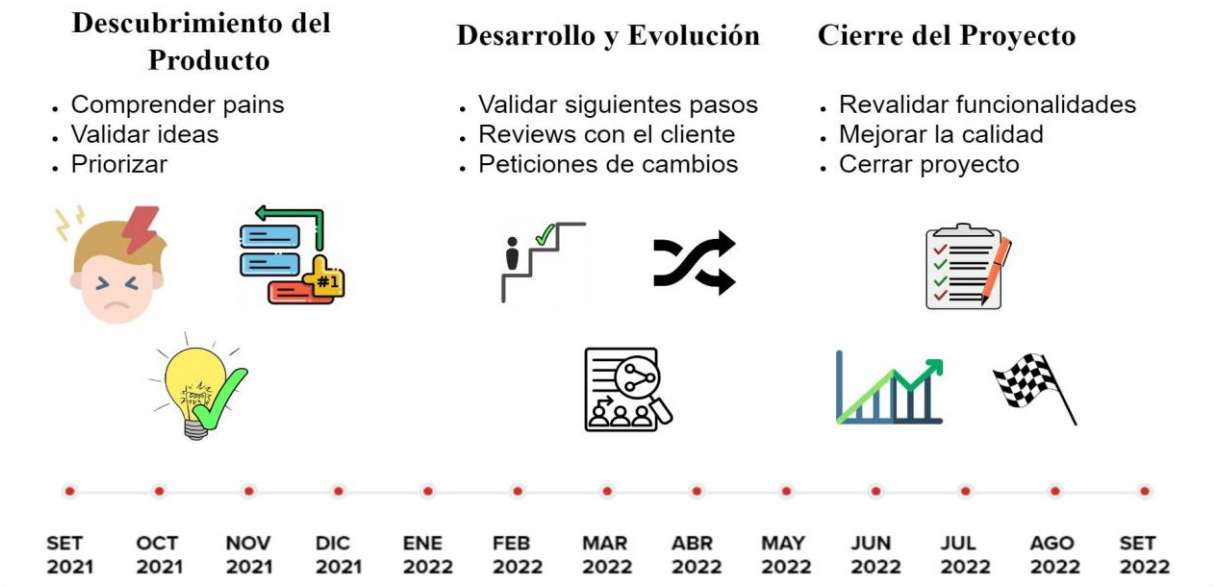


Figura 5.2: Etapas del proceso de validación de Ingeniería de Requerimientos

Etapa de Descubrimiento del Producto

En esta etapa del proyecto el equipo se encargó de validar los requerimientos que se fueron formando tras hablar con el cliente y comprender sus *pains*. El hecho de que el área de P&C no tuviera experiencia real siendo parte de la creación de un *software*, dificultó en parte la validación, ya que carecían de conocimientos sobre el proceso que se llevaría a cabo en la creación del producto. Esto hizo que muchas veces en la fase inicial del proyecto se aprobaran requerimientos sin realmente estar seguros de si estos serían útiles para el producto. Por ello, el equipo necesitó como primer paso generar una priorización en los requerimientos relevados, para poder hacer énfasis en la validación de aquellos requerimientos que serían desarrollados en las primeras iteraciones del producto. Además, se trató de siempre serle transparente al cliente sobre en qué momentos serían desarrollados cada uno de los requerimientos, para que este estuviese tranquilo y comprendiese fácilmente las diferentes etapas que tendría el producto.

Con los requerimientos que fueron validados por el cliente en esta primera etapa, el equipo logró generar un primer plan de *release* a seguir en el desarrollo del producto. Estas validaciones a su vez fueron clave para que tanto el equipo como el cliente pudiesen visualizar por primera vez qué funcionalidades tendría el producto final.

Etapa de Desarrollo y Evolución

Al comenzar el desarrollo, la validación del cliente fue crucial para el equipo, ya que le permitía saber si el proyecto se estaba construyendo alineado a sus necesidades, dando lugar a posibles cambios. Para asegurarse de esto, se realizaban *reviews* con el cliente al final de cada iteración de desarrollo en las que se le mostraba los incrementos generados, diferenciando los provenientes del desarrollo planeado, de los que provenían del *feedback*. Este brindaba una opinión de si las funcionalidades estaban cumpliendo con sus expectativas, y a su vez peticiones de cambios para que las mismas cumplieran con sus necesidades. Dado a que no era sencillo encontrar tiempos en los que poder reunirse con el cliente, estas reuniones eran también utilizadas para volver a validar los requerimientos que serían implementados en la siguiente iteración de desarrollo, asegurándose así el equipo de minimizar la cantidad de retrabajo a realizar y maximizar el valor otorgado al cliente en cada iteración.

Finalizadas las bases del proyecto, el equipo generó dos instancias en la nube del producto las cuales el cliente podía acceder y utilizar libremente. Una de estas era para el desarrollo (utilizándose como ambiente en las demos), y la otra era de uso exclusivo del cliente. Con esta en funcionamiento se logró que el cliente la comenzara a utilizar, percatándose (mediante el uso), qué requerimientos le serían de mayor utilidad, y qué cambios podrían ser realizados sobre las funcionalidades ya desarrolladas.

Etapa de Cierre del proyecto

Al comenzar el último de los *releases* del sistema, el equipo seguía con las validaciones periódicas de las funcionalidades que se estaban construyendo, pero en esta etapa a su vez se comenzó a revalidar las funcionalidades ya finalizadas del sistema. El objetivo de esto era asegurarse que el sistema entregado cumpliese con todas las expectativas del cliente, y en caso de ser necesario y posible, mejorar su calidad con cambios que no requiriesen demasiado esfuerzo. En el caso de los cambios que no pudiesen ser realizados dentro del alcance del proyecto, se los registraba dentro de la lista de mejoras a futuro. Las validaciones continuaron siendo, en su mayoría, en la reunión bisemanal (*review*) que se tenía agendada con el cliente. A su vez se les pidió a los usuarios que comenzaran a utilizar el sistema con una vista un poco más crítica, para que se pudiesen detectar posibles mejoras o flujos que podrían ser optimizados. Este uso que le dieron a la aplicación con un foco consciente en la búsqueda de errores o mejoras

generó ideas de cambios que beneficiaban en gran medida, siendo en su mayoría cambios sencillos de desarrollar, pero que marcaban la diferencia al usar la aplicación.

5.2 Requerimientos funcionales

Los requerimientos funcionales fueron cambiando a lo largo del proyecto, ya que la empresa estaba en constante evolución y también sus procesos. Es por este motivo que la lista fue sufriendo varios cambios, teniendo eliminaciones e inserciones en el camino. En esta sección se listarán las épicas (implementadas), ordenadas por *releases*, que conforman el sistema. En el anexo "*Backlog* del Proyecto" se encuentra el listado de todas las historias implementadas desglosadas por épicas y *releases*. Los *releases* fueron divididos y priorizados con el cliente, se encuentran ordenados de manera que el primero conforma las funcionalidades básicas del proyecto y componentes iniciales, es decir, el conjunto mínimo de funcionalidades con las que el cliente podría empezar a trabajar. Y los otros dos brindan de forma incremental el resto de las funcionalidades que componen a la totalidad del sistema.

A continuación, se muestra la lista de épicas ordenadas por *release* del sistema, posicionando las mismas en el *release* que tuvo mayor peso en su desarrollo:

***Release* 1**

Épica E-1: Gestión de entidades empresariales

Como usuario de Ariadne's Thread
Quiero gestionar las entidades del sistema
Para tener su información actualizada y poder visualizarla fácilmente al gestionar la empresa

Épica E-2: Visualización organizacional

Como usuario de Ariadne's Thread
Quiero ver, mover y cambiar las entidades del sistema
Para poder visualizar, mover o reorganizar estos elementos, en base a los cambios de la organización

Release 2

Épica E-9: Cambios de *Feedback*

Como usuario de Ariadne's Thread
Quiero que se realicen ciertos cambios solicitados
Para que el sistema se adapte a las necesidades del negocio

Épica E-4: Gestión de Categorías

Como usuario de Ariadne's Thread
Quiero poder crear y gestionar categorías, y asignarlas a las diferentes entidades del sistema
Para poder representar ciertos aspectos de interés para la organización, pudiendo etiquetar las diferentes entidades con estas y así tener esta información extra disponible para tomar mejores decisiones de negocio

Épica E-5: Filtros

Como usuario de Ariadne's Thread
Quiero poder filtrar en base a las diferentes categorías y sobre algunas entidades del sistema
Para poder acceder o visualizar rápidamente a información de mi interés, para la toma de decisiones de negocio

Release 3

Épica E-8: Links personalizables

Como usuario de Ariadne's Thread
Quiero agregar y gestionar links en las entidades del sistema
Para poder almacenar información de referencias a otros sistemas o documentos

Épica E-6: Reportes

Como usuario de Ariadne's Thread
Quiero poder ver visualmente la historia de equipos, empleados y cambios en estos
Para poder analizar los diferentes cambios que sucedieron a lo largo del tiempo y tomar decisiones estratégicas en base a estos.

Épica E-3: Gestión de usuario

Como usuario de Ariadne's Thread
Quiero poder gestionar usuarios y permitir que estos ingresen al sistema
Para poder tener un registro de los usuarios y que estos puedan utilizar el sistema

Épica E-7: Escenarios

Como usuario de Ariadne's Thread
Quiero gestionar escenarios
Para trabajar sobre la estructura actual de la organización, realizando cambios en la misma sin impactar en la estructura real de la empresa, y poder generar posibles cambios estratégicos.

5.3 Requerimientos no funcionales

En la siguiente sección se detallarán los requerimientos no funcionales que se tomaron en consideración para la realización del proyecto. Los requerimientos no funcionales [11] representan las características y limitaciones generales del producto y estos fueron escritos en forma medible para luego poder verificar si se cumplieron.

5.3.1 Usabilidad

Este punto fue esencial para el proyecto dado que el sistema tendrá un elevado uso dentro la empresa. Por esto, se estableció que el producto debe ser agradable y permitir la ejecución eficiente de las tareas. Para evaluar el cumplimiento de estos, se crearon una serie de métricas de uso y satisfacción del sistema. Estas se pueden ver en la sección "Usabilidad" en el capítulo "Gestión de la calidad".

RNF 1 - El sistema debe ser fácil de usar. El 80% de los usuarios deberá poder utilizar el sitio web sin ningún tipo de capacitación ni ayuda.

RNF 2 - Los usuarios nuevos podrán crear un nuevo equipo, agregar integrantes al equipo y asignar un proyecto en menos de 20 minutos. Los usuarios experimentados deberán hacerlo en 10 minutos.

RNF 3 - Al menos el 70% de los usuarios encuestados deberían estar satisfechos con la usabilidad del producto.

RNF 4 - El sistema debe ser actualizado regularmente según el *feedback* del cliente con el fin de que se adapte a su cultura y permita realizar sus operaciones.

5.3.2 Mantenibilidad

Dado que el sistema será mantenido por la empresa del cliente, en particular por el área de ingeniería, se estableció junto con estos un conjunto de prácticas de *Clean Code* [2] y principios de diseño que el sistema debe seguir. Además, estuvieron presenten en decisiones importantes de implementación que pudieran afectar la mantenibilidad del sistema.

RNF 5 - El sistema debe obtener buenos índices en herramientas de análisis de código estático (Rating A en *Sonarqube*) con menos de 5% de repetición de líneas, y un número menor a 350 *code smells* cada 10000 líneas.

RNF 6 - El cliente debe recibir una documentación de la arquitectura, decisiones de diseño y diagramas.

RNF 7 - Se busca que el 90% o más de los *bugs* encontrados sean encontrados por el equipo, contra un 10% o menos encontrados por el cliente.

RNF 8 - Se busca un tiempo medio de solución entre todos los errores que sea bajo, tomando en cuenta su severidad, siendo estos:

- Severidad mínima y baja: 2 *sprint*s o menos.
- Severidad media: 1 *sprint* o menos
- Severidad alta y máxima: 0 *sprints*.

Siendo “0 *sprints*” si se soluciona en el *sprint* que fue reportado y “1 *sprint*” si se soluciona el *sprint* siguiente.

RNF 9 - Se busca que el tiempo medio entre todos los *feedbacks* implementados sea de un *sprint* o menos, siendo que un *sprint* representa el *sprint* siguiente al reporte.

5.3.3 Seguridad

La seguridad es un factor importante ya que se almacena la información personal de los empleados y la empresa. El producto debe ser utilizado solamente por el personal de Qualabs, y personas no autorizadas no deben poder acceder a los datos.

RNF 10 - Se debe permitir el inicio de sesión solamente a los usuarios registrados y la *API* debe estar autenticada mediante token de identificación.

RNF 11 - La comunicación entre cliente y servidor debe estar basada en el protocolo HTTPS.

5.3.4 Disponibilidad

El sistema está pensado para ser utilizado frecuentemente para los procesos y operaciones del cliente y por tanto es importante que se encuentre en funcionamiento cuando se quiera utilizar. Además, el hecho de no estarlo puede afectar negativamente a la usabilidad, generando frustraciones y malas experiencia de usuario si el sistema está fuera de servicio cuando se lo quiere utilizar o entra en este estado mientras se lo está utilizando. Por esto, se tomaron decisiones y consideraciones en el desarrollo para promover este atributo.

RNF 12 - El sistema debe estar disponible para el uso del cliente, se intentará proporcionar una disponibilidad del 90% dentro del horario laboral de trabajo los días de semana (pudiendo exceptuar dicha regla los fines de semana).

5.3.5 Restricciones

Res1 - Debido a que el cliente va a mantener el producto en el futuro, el sistema debe ser una página web utilizando las tecnologías React para *frontend* y NodeJs en *backend*. Debe ser alojada en AWS.

5.4 Entregables adicionales

Además del sistema a desarrollar, el área de Ingeniería del cliente (que es la que mantendrá el sistema una vez entregado), solicitó un conjunto de entregables que quiere recibir además del producto principal. Dentro de esta lista se encuentran los documentos requeridos para el cumplimiento del RNF 6. Estos son:

- Ambos elementos del sistema (*frontend* y *backend*) deben tener un archivo explicativo (*readme*) que tenga las instrucciones para levantar correctamente ambos sistemas de manera local.
- Ambiente de Docker local, para el caso del *backend*, el cliente pidió que sea posible levantar este sistema mediante la creación de un Docker Compose, con el fin de que

pueda levantar el sistema sin necesidad de instalar en su computador los servicios asociados. Se deben entregar los archivos de Docker correspondientes para este fin.

- Documentación de la arquitectura del sistema que contenga una descripción de los diferentes componentes que conforman la solución, y cómo se realiza la comunicación entre sus diferentes partes. Así como también los servicios de la nube utilizados y cómo se encuentran estos configurados.
- Instrucciones para configurar y levantar con sus debidas configuraciones los servicios de la nube de manera correcta (con el fin de poder seguir el mismo procedimiento que realizó el equipo, en caso de que tengan que crear un nuevo ambiente).
- Documentación de mejoras e incidencias abiertas, este documento tiene como fin dejar constancia de las posibles deudas técnicas que hayan quedado pendientes de resolver en el sistema, es decir partes o elementos del sistema cuya implementación el equipo entiende que tiene lugar a mejoras. También debe contener una lista de *bugs* conocidos que no han sido solucionados.

En adición a estos elementos se hará el traspaso de credenciales de los sistemas, y acceso a las herramientas utilizadas para la gestión del proyecto.

5.5 Conclusiones y lecciones aprendidas

Una gran lección aprendida que tuvo el equipo en lo que respecta a la ingeniería de requerimientos, fue la importancia de tener un buen proceso de relevamiento y validación de requerimientos, y mantenerlo constante a lo largo del proyecto. Gracias al proceso seguido, el equipo pudo comprender que la ingeniería de requerimientos es algo más que llegar a un listado de funciones a desarrollar. Como se utilizaron las metodologías ágiles para el desarrollo del proyecto, y gracias a la buena disposición de Qualabs, el equipo pudo experimentar de primera mano las ventajas que otorga el tener un contacto cercano y constante con el cliente. A su vez, se comprendió la importancia de mantener al cliente involucrado en el proceso durante todo el proyecto, enfocándose en ir más allá del problema, pudiendo entender al cliente y sus procesos con el fin de lograr un buen producto.

Otro punto a destacar es el uso del *Design Sprint*, que proporcionó grandes ventajas sobre el proceso de relevamiento y validación de requerimientos realizado. El hecho de tomarse un tiempo para pensar, diseñar las pantallas y poder validarlas con el cliente antes de

implementarlas fue muy beneficioso. Además de ser otra instancia que permitía entender y conectar más con el cliente, en muchas ocasiones dio lugar a cambios o permitió detectar mejoras en los prototipos en base a como ellos lo iban a utilizar. A su vez, permitió potenciar el sentimiento de involucramiento del cliente en la construcción del sistema y aumentar la confianza con el equipo.

Junto al valor que genera mantener al cliente informado y adentrado en el proceso, un punto a destacar es la importancia de las negociaciones y estar abierto a los cambios. Esto no solo permitió poder ir adaptando el sistema, sino también ir adaptando el proceso y la metodología elegida conforme iba avanzando el proyecto. Como resultado, hizo que se pudiera ir balanceando las cargas de trabajo sobre las diferentes secciones, y las formalidades de las reuniones. Esto permitió ir adaptando y mejorando los procesos, además de permitir sacar el mayor provecho de las instancias.

Finalmente, si bien el hecho de tener un cliente no técnico tuvo sus desafíos, el equipo cree que esto sirvió como un gran aprendizaje para ambas partes. Por parte del equipo, este aspecto “no técnico” permitió enfocar los procesos con un mayor pienso en el diseño, tomar tiempos para idear, negociar y entender las necesidades por detrás de las peticiones. Y por parte del cliente, expresaron que esto les sirvió para tener una mirada más en detalle de lo que es un proyecto de *software*, y lograr entender un poco más el mundo técnico y sus procesos.

6 Arquitectura y Diseño

El presente capítulo contendrá las decisiones tomadas por parte del equipo en lo referente a la arquitectura del sistema, el diseño de código y los desafíos que se presentaron durante el transcurso del desarrollo del proyecto.

6.1 Descripción general de la arquitectura *Cloud*

Dado a que el cliente solicitó que se usara la plataforma *Amazon Web Services* (AWS) [12] para realizar el *hosting* en la nube, el equipo tuvo que seleccionar cuáles de los servicios de Amazon utilizaría para el *deploy* del sistema. A la hora de seleccionar los servicios, se tomaron múltiples factores en cuenta.

Dentro de los más importantes estaban los costos del servicio, debido a que se desplegarían dos ambientes, uno de ellos utilizado para el desarrollo y el otro utilizado por el cliente. Este factor fue crucial a la hora de seleccionar qué servicios se utilizarían para la arquitectura, tanto por los costos generados a la empresa durante la etapa de desarrollo, como por los costos de mantener la aplicación posterior a la finalización del proyecto. Aunque mantener los costos bajos era importante, siempre se tomaron en cuenta todos los factores antes de tomar una decisión de qué servicios utilizar.

Otro punto fuerte a la hora de seleccionar los servicios de AWS a ser utilizados fue la mantenibilidad. Esto se debió a que se deseaba minimizar el esfuerzo necesario de mantenimiento de la arquitectura *cloud*, permitiendo así que el equipo pudiera dedicarse plenamente al desarrollo del producto. A su vez, como Qualabs se encargaría de mantener el sistema luego de finalizado el proyecto, se trató de que la arquitectura seleccionada fuese sencilla de mantener, fácil de comprender y que no requiriese conocimientos técnicos muy avanzados para poder modificarla a su gusto.

Dado que la empresa se encontraba en pleno crecimiento durante el desarrollo del proyecto, y que se sabía que se continuaría creciendo luego de finalizado este, se decidió tomar en cuenta también la escalabilidad del sistema. Para ello fue necesario analizar si los servicios utilizados tenían habilidades de escalado, y en el caso de no tenerlo, si el equipo podría generar dichas habilidades a través del uso de otros medios u otras herramientas.

Como puntos finales a la hora de tomar la decisión se tomaron en cuenta el *performance* y la seguridad, sin embargo, muchas veces estos mismos se generaban como *tradeoff* del uso de AWS como servicio de *hosting*. Esto debido a que los diferentes servicios que se utilizaron se encontraban todos dentro de esta plataforma, y la misma cuenta con una red interna la cual hace que los diferentes componentes tengan muy poca latencia entre ellos (siendo especialmente importante para el servidor de *backend* y la base de datos). A su vez, por parte de la seguridad, AWS brinda la posibilidad de utilizar *Security Groups* [13] para conectar los diferentes servicios, con lo cual se puede hacer que un servicio solamente se pueda conectar con otro servicio específico. Eso mejora en gran medida la seguridad de la arquitectura dado que hay menos salidas al mundo por las cuales podría beneficiarse un atacante.

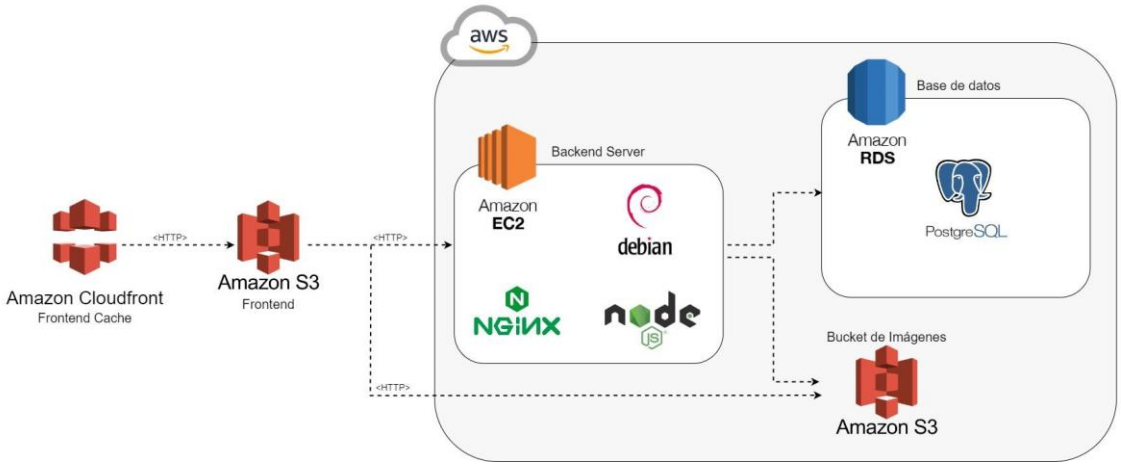


Figura 6.1: Arquitectura de *deployment* en la nube

6.1.1 Frontend

El código de *frontend* de la aplicación fue desplegado en un *bucket S3* [14], el cual es un servicio que permite almacenar archivos en la nube. Este fue seleccionado gracias a su practicidad para alojar páginas *web*, su bajo costo (ya que el cliente paga solamente lo que usa), y a que es el método más utilizado para alojar *webs* dentro del ecosistema AWS, por lo que tiene una gran fuente de documentación y tutoriales al respecto. Para agregarle una capa de seguridad a la *web*, se decidió hacer que la conexión fuese a través de HTTPS, esto para que todos los datos de comunicación entre el cliente y el servidor viajasen cifrados. Para realizarlo se decidió utilizar el servicio CloudFront [15], que es un *cache* que se encarga de hacer que los usuarios de la *web* reciban mucho más rápido los archivos estáticos de la página, haciendo que estos tengan un menor tiempo de espera hasta que la web se cargue. A este se le anexó un certificado SSL [16]

para conseguir que las comunicaciones que se realicen a través de dicho servicio se encuentren cifradas, con lo que nos brinda beneficios tanto por el lado de la seguridad como por el lado de la *performance*.

6.1.2 Backend Server

Para la arquitectura que se tendría en el *backend* del sistema se analizaron principalmente los servicios de EC2 [17] y Elastic Beanstalk [18]. EC2 es un servicio de *Infrastructure as a Service* (IaaS) [19] que le permite al desarrollador crear una instancia de una máquina con el sistema operativo que quiera, y modificarla a gusto. Por otra parte, Elastic Beanstalk es un servicio de *Platform as a Service* (PaaS) [20] que permite que un desarrollador despliegue una aplicación sin tener que preocuparse por su infraestructura. Ambos servicios son similares en cuanto a las capacidades que le permiten al desarrollador, sin embargo, tomando en cuenta la experiencia que tiene Qualabs con el servicio EC2, y las posibilidades que este brinda de poder personalizar el servidor a gusto, se optó finalmente por dicho servicio.

En cuanto a lo que refiere a costos, en la instancia de EC2 utilizada se pudo optar por una instancia de bajos recursos, dado que el servidor de *backend* no necesita grandes cantidades de memoria RAM ni de habilidad de procesamiento. A la hora de seleccionar el sistema operativo a utilizar, se analizó cuáles eran los que utilizaba Qualabs en su día a día, por lo que el equipo notó que muchas personas de la empresa optaban por el uso de sistemas operativos Linux, siendo Debian [21] la distribución más utilizada dentro de la empresa. De utilizarla, el equipo estaría tranquilo de que cuando Qualabs necesitase realizar mantenimiento en el servidor, este tendría personal capacitado para realizarlo. Por esta razón, y por el hecho de que el integrante del equipo que se encargaría de configurar toda la arquitectura *cloud* también contaba con experiencia en dicho sistema operativo, se decidió que sería la mejor opción.

Como se mencionó anteriormente en el capítulo, dado que la empresa se encontraba en pleno crecimiento de personal, y que la tendencia era a que continuase creciendo, se necesitó tomar en cuenta la escalabilidad al tomar las decisiones de los servicios a utilizar. Para facilitar la escalabilidad de la instancia de EC2 a futuro se decidió utilizar NGINX [22], el cual es un software de código abierto que puede ser utilizado como *reverse proxy* [23], balanceador de carga [24], entre otras cosas. El equipo instaló y configuró este *software* en el servidor utilizándolo solamente como *reverse proxy* para poder mapear el puerto de salida del servidor con el puerto interno de la aplicación de *backend* en ejecución. Igualmente, en caso de que el

cliente quisiese escalar la cantidad de servidores de la aplicación debería solamente cambiar la configuración de NGINX, transformándolo en un balanceador de carga que distribuya la carga de la aplicación a diferentes servidores. Esto permite que la aplicación funcione correctamente en su estado actual, y genera la posibilidad de que en el futuro se pueda escalar su uso fácilmente.

En el caso de que la empresa decida que no quiere escalar la aplicación horizontalmente pero sí verticalmente [25], el servicio de EC2 tiene la posibilidad de aumentar el *hardware* configurado en la máquina. Esto le permitirá al cliente poder escalar en la cantidad de recursos, permitiendo que el servidor de *backend* esté preparado para recibir más tráfico *web* sin crecer la cantidad de instancias a mantener. Dado que la decisión de si escalar horizontal o verticalmente no recae sobre el equipo, se decidió generarle al cliente ambas posibilidades para que este pueda decidir a futuro como desea seguir creciendo el sistema.

En virtud de que el *backend* del sistema tenía como restricción ser implementado en NodeJs (Res1), y que el equipo decidió utilizar Typescript como lenguaje de programación, el equipo instaló las dependencias en el servidor para poder ejecutar estas herramientas. En un inicio, se analizó la posibilidad de utilizar Docker [26] dentro del servidor, pero la misma se descartó rápidamente dado que complejizaba demasiado el mantenimiento del sistema, agregándole una herramienta más que se tendría que tener en cuenta al realizar cambios en el servidor. Igualmente, se generó un archivo Dockerfile (archivo de configuración de Docker) configurado para correr el *backend* y un archivo de configuración de Docker Compose [27] con instrucciones de cómo ejecutarlo[5], para que el cliente pudiese probar el servidor fácilmente en un ambiente local, y en caso de quererlo, poder utilizar Docker en el servidor a futuro.

Ya que el equipo decidió utilizar un sistema operativo Linux en el servidor, y para poder ayudar al sistema a recuperarse luego de fallas, se decidió configurar la aplicación de *backend* como un servicio dentro del servidor de EC2. Para esto se utilizó el comando *systemctl* [28] y una configuración en los archivos del sistema, con el objetivo de volver a la aplicación parte de los servicios clave del sistema. Dicha acción permitió al equipo configurar que en caso de que la aplicación de *backend* fallase, esta se volvería a iniciar, permitiendo así mantener una alta disponibilidad del sistema.

[5] Para más información sobre los entregables que se le brindaron al cliente luego de finalizado el proyecto ver sección “Entregables adicionales” dentro del capítulo de “Ingeniería de requerimientos”.

6.1.3 Base de datos

Antes de seleccionar el servicio que se utilizaría para albergar a la base de datos, el equipo se preocupó por seleccionar qué base de datos iba a utilizar. Para ello se tomaron en cuenta diversos factores tales como la mantenibilidad, la versatilidad, y qué tan fácil sería utilizarla dentro del contexto del sistema a ser desarrollado. Con esto en mente, el equipo analizó la posibilidad de utilizar diversas bases de datos hasta que llegó a la decisión de utilizar PostgreSQL [29]. La decisión fue tomada gracias a que dicho motor de base de datos es muy versátil, debido a que es una base de datos relacional, pero al mismo tiempo tiene la posibilidad de manejar tipos de datos JSON [30], guardándolos en esta con el tipo de datos JSONB [31] (lo que sería muy útil para el dominio del proyecto). A su vez, este motor existe hace bastante tiempo en la industria y es bastante utilizado, además de tener una buena documentación y una gran comunidad por detrás. Esto asegura al equipo que cualquier problema que tuviesen al desarrollar con este motor se podría encontrar medianamente fácil la solución en internet.

En cuanto al servicio que se utilizaría para alojar la base de datos del sistema, el equipo analizó principalmente los servicios EC2 y RDS [32]. El primero de ellos, como ya se ha mencionado anteriormente, es un servicio de IaaS que le permite al usuario generar un servidor con el sistema operativo que quiera y alojar cualquier tipo de aplicación dentro de este. Algunos desarrolladores lo seleccionan para alojar su base de datos dentro de un servidor en el cual tengan completo control, pero el lado negativo de esto es que genera que el equipo tenga otro servidor más por el cual preocuparse y mantener. Por otro lado, RDS es un servicio que permite instanciar servidores de base de datos relacionales sin tener que preocuparse por su infraestructura. Sumado a esto, tiene ventajas como la posibilidad de agregarle autoescalado, habilidades de replicación de datos (lo cual podría serle útil a Qualabs en el futuro), una simple configuración para generar *backups* [33] periódicos, entre otras. Con todas las ventajas que le generaba al equipo el uso de RDS, y sabiendo que la base de datos seleccionada era relacional, se optó por utilizar este servicio para albergar la base de datos del sistema.

6.1.4 Bucket de imágenes

Con el objetivo de poder guardar las fotos de los diferentes empleados, equipos y proyectos del sistema, se instanció otro *bucket* S3. Este se utilizó dado que no se quería reutilizar el que contenía los archivos del *frontend* del sistema, para no mezclar las responsabilidades de las

unidades de almacenamiento. A su vez, dado que los almacenajes de S3 solo cobran al usuario lo que este usa realmente, no se le encontró problema de llevar a cabo esta decisión.

6.2 Decisiones de arquitectura

Durante el proceso de desarrollo del proyecto, se realizaron múltiples decisiones en lo que respecta a la arquitectura del sistema. Las mismas fueron tomadas tanto para el *frontend* como para el *backend* del producto.

6.2.1 Frontend

La implementación del *frontend* del sistema fue realizada utilizando el enfoque Monolítico [34], haciendo que sus componentes tuviesen bajo acoplamiento, y siempre que fuese posible, creándolos, pensando en el reuso de los mismos. Estos puntos fueron claves para el desarrollo del sistema dado que el cliente no tenía una clara visión de cómo debería verse el sistema final, por lo que la aplicación *web* podría sufrir múltiples cambios en el proceso de creación.

6.2.1.1 Monolito Vs. Micro Frontends

Al pensar en la arquitectura que se llevaría a cabo para el *frontend* del sistema, el equipo analizó principalmente las posibilidades de utilizar el patrón de arquitectura Monolito ó el patrón de *Micro Frontends* [35]. Esta decisión fue analizada por todo el equipo, pero tras deliberar, se tomó la decisión de utilizar el enfoque Monolítico. Dicha decisión surgió de que la UI del sistema no tenía, en un principio, claridad de cuáles serían todos los diferentes servicios que tendría. A su vez, el patrón de *Micro Frontends* está pensado principalmente para el trabajo de múltiples equipos en diferentes partes de un mismo sistema, los cuales quieren desacoplar sus soluciones para que estas no interfieran con el desarrollo de las otras partes. Se comprendió que el uso de este patrón en el sistema no habría brindado mejoras claras al mismo, sino que solamente causaría mayor complejidad a la hora de desarrollar, y también a la hora de realizar los despliegues de cada uno de los servicios. Por estos motivos fue que el equipo decidió no utilizar dicho patrón para el desarrollo del sistema.

6.2.1.2 Acoplamiento de estilos CSS

Debido a que el desarrollo *web* ha evolucionado a pasos agigantados, muchos de los elementos que fueron diseñados para ser utilizados en sus comienzos causan posibles problemas a la hora

de generar código hoy en día. Un ejemplo claro de esto es el funcionamiento de CSS [36] (hojas de estilo en cascada), el que en su momento permitió comenzar a reutilizar estilos en diferentes partes de la *web*, pero hoy en día genera problemas en el desarrollo de sistemas medianos o grandes, como el acoplamiento de estilos. Este problema refiere a que un desarrollador puede generar un estilo para un componente en específico y colocarle un nombre "X" al estilo, y luego otro desarrollador genera otro estilo que desea sea diferente del primero pero le coloca nuevamente "X" como nombre. Al hacer esto, causa que a todos los elementos que se les asigne el estilo "X" se les aplique ambos cambios de estilos, el que generó el primer desarrollador, y a su vez, el que generó el segundo desarrollador. Esto causa problemas a la hora de desarrollar, dado a que cada persona que vaya a generar código de estilos debería crear un nombre único para su elemento, para que el mismo no se superponga con uno ya existente. El crear cada vez un nombre único para un nuevo estilo, pensando en que el mismo sea nemotécnico, y que no esté repetido, no es para nada sencillo.

Dicho problema existe ya hace un tiempo en el ámbito de desarrollo *web*, por lo que se han generado tanto soluciones que simplemente generan un estándar para el nombrado de estilos, como Atomic CSS [37] o BEM [38], como librerías que ayudan a eliminarlo. Para el desarrollo del sistema, el equipo analizó las librerías más utilizadas en React que se encargasen de solucionarlo, encontrando que las dos más utilizadas eran *styled-components* [39] y CSS *modules* [40]. Dado que la segunda opción mencionada tenía beneficios como utilizar archivos CSS (que podrían ser cacheados para mejor *performance*), permitía el uso de SASS [41] y no generaba un gran cambio de sintaxis con respecto a lo que el equipo estaba acostumbrado, se optó por utilizar dicha librería. Con su uso se pudieron generar estilos sin tener que estar preocupados por el acoplamiento que se podría generar entre componentes, y a su vez, se pudo reutilizar código de estilos para diferentes partes del proyecto que eran visualmente similares.

6.2.1.3 Reuso de componentes

Con los objetivos de agilizar el desarrollo, estandarizar la estética del sistema y minimizar sus puntos de falla, el equipo se enfocó en que todo componente que pudiese ser reutilizado en múltiples partes del sistema, se implementase pensando en su reuso. Con esto, se logró que múltiples secciones en el sistema se desarrollaran más ágilmente, dado que algunos elementos necesarios para dichas secciones ya habían sido generados con anticipación. A su vez, este enfoque logró que fuese más sencillo reparar muchos errores del sistema en poco tiempo, ya

que el equipo sólo debía repararlos en un componente que estaba siendo reutilizado, en lugar de tener que replicar los cambios en las diferentes partes del sistema.

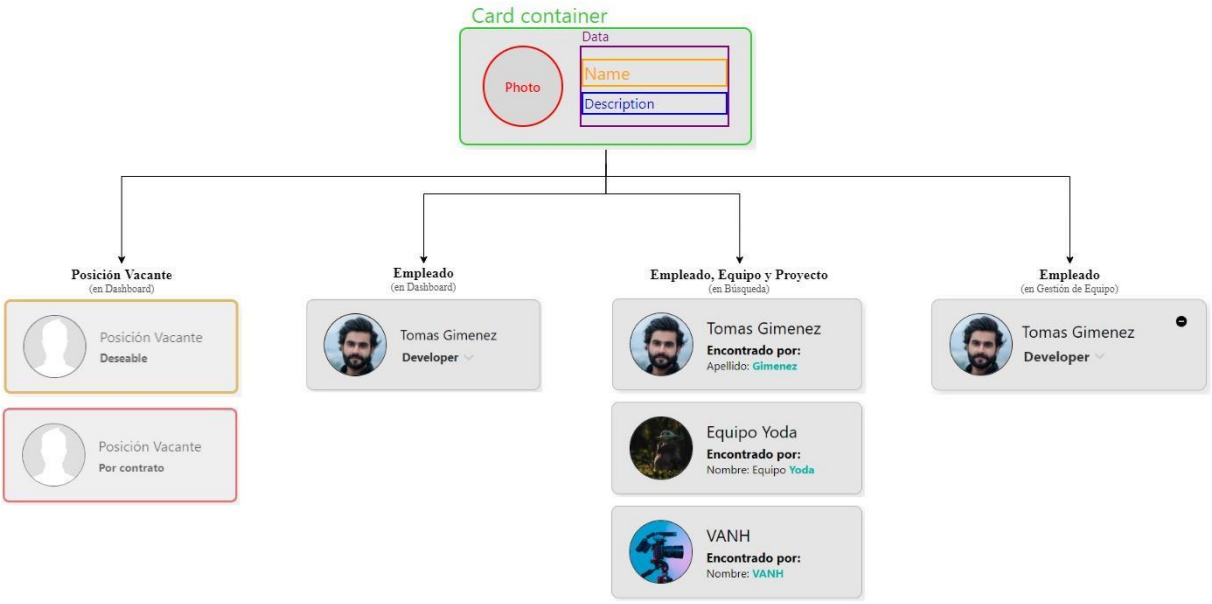


Figura 6.2: Reutilización del componente *Card*

Uno de los componentes en los que se ve más claramente el reuso realizado en la aplicación, fue el componente de *card* (como se ve en la Figura 6.2). Este contaba con un diseño sencillo, en el cual se podía agregar una imagen, un nombre y una descripción, y a su vez se le podían adicionar diferentes acciones que sucederían de ocurrir diferentes eventos, como por ejemplo un *click*. Dicho componente fue reutilizado en múltiples partes del sistema, haciendo que el mismo fuese más ágil de desarrollarse y permitiendo mantener una misma estética a lo largo de todo el producto.

Junto a este componente de *card* también se desarrollaron múltiples otros componentes. Estos otros llegaban a ser componentes más sencillos como por ejemplo campos de texto, botones o *links*, y también componentes con cierta complejidad por detrás como modales, tablas, etc. El haber desarrollado tantos componentes que tuviesen un uso genérico en la aplicación produjo un ahorro de tiempo a la hora de desarrollar el sistema en general, y permitió mantener una consistencia estética. Con estos, el equipo pudo centrar sus esfuerzos en generar nuevas pantallas, o cambiar las existentes en base al *feedback* recibido, brindándole utilidad al usuario sin necesitar cada vez crear de cero estos elementos.

6.2.2 Backend

Para la implementación del backend del sistema, el equipo decidió utilizar el patrón de arquitectura Monolito. Este propone que todos los componentes del sistema se encuentren dentro de una misma aplicación, siendo la misma independiente de otros servicios. Dado que el uso del ambiente de desarrollo NodeJs era una restricción en el proyecto, y que no existía restricción en el lenguaje de programación a utilizar, el equipo optó por utilizar Typescript.

6.2.2.1 Monolito Vs. Microservicios

Cuando el equipo estaba decidiendo qué arquitectura sería la indicada para el sistema a ser desarrollado, surgió el planteamiento de si utilizar Monolito o Microservicios [42]. Para poder tomar la decisión, el equipo tuvo que analizar las ventajas y desventajas de ambos enfoques, analizar las necesidades del cliente y contrastar con la utilidad que cada uno de estos tendría en el sistema.

Modificabilidad

Si se analizan ambos enfoques enfrentados con respecto a la modificabilidad del producto final, se denota fácilmente que es mejor utilizar el patrón de Microservicios. Esto dado que el uso de este patrón hace que las unidades de código se encuentren separadas por responsabilidades. Este detalle genera que el sistema termine siendo más simple de comprender, y por ende, más simple de modificar. Por otra parte, a medida que va creciendo la cantidad de código de un Monolito, la complejidad a la hora de realizar cambios sobre el mismo va aumentando, disminuyendo así su modificabilidad.

Complejidad al desarrollar

Con respecto a la dificultad que presenta desarrollar en los diferentes enfoques, se puede notar que el más sencillo de ellos es el Monolítico. Esto se debe a que todo el código se encuentra en una sola aplicación, pudiendo cualquier parte del sistema conectarse a otra directamente desde el código. A su vez, es notable el aumento de complejidad que requiere el uso y desarrollo de Microservicios. Esto se da principalmente dado que se complejizan factores como la interconexión entre servicios, cada nueva funcionalidad puede requerir cambios en múltiples servicios, o también factores como el despliegue y mantenimiento de diferentes servidores.

Performance

Para los Microservicios la *performance* claramente se ve afectada con respecto a la del enfoque Monolítico. Esto debido a que la interconexión entre los diferentes servicios se realiza mediante comunicación de red. Otro factor a tener en cuenta es que, al tener diferentes servicios independientes distribuidos, también se tienen bases de datos independientes para cada uno de ellos. Esto causa que algunas búsquedas de datos requieran realizar múltiples llamadas a diferentes bases de datos, y luego unirlas en memoria, causando *overhead*. Si bien este último factor podría ser solucionado en parte utilizando técnicas como la replicación de datos, esto llevaría a incurrir en nuevas problemáticas como la consistencia eventual, necesidad de uso de técnicas de sincronización, entre otras.

Escalabilidad

Si tomamos en cuenta la escalabilidad del sistema, es claro que el enfoque que brinda mayor beneficio es el de Microservicios. Dicho enfoque permite escalar cada servicio del sistema independientemente, permitiendo un mejor uso de recursos. Por otro lado, si se desea escalar un Monolito, el mismo debe escalar todas las diferentes partes del sistema en simultáneo.

Seguridad

Por el simple hecho de que los Microservicios generan múltiples sistemas para albergar a los diferentes servicios, y que los mismos deben intercomunicarse entre sí, es claro que este enfoque es el que resulta más complejo a la hora de asegurar su seguridad. Estos múltiples servicios generados deben todos contar con la seguridad apropiada, tener maneras de intercomunicarse de forma segura entre ellos, entre otras cosas. Por otro lado, en caso de tener un sistema Monolítico, esto implicaría tener un solo servidor, el que tendría dentro todos sus servicios. Al ser así, el equipo tendría que preocuparse por proteger una sola aplicación, siendo una tarea más sencilla desde el punto de vista de la seguridad.

Costos

A la hora de analizar cuál de ellos es más eficiente con respecto a costos a los que incurrirá la empresa, se puede notar que el uso de una arquitectura Monolítica genera menores costos que el uso de Microservicios. Esto se da principalmente por la gran cantidad de elementos que se tienen que desplegar para el correcto funcionamiento de los Microservicios (múltiples

servidores de *backend*, múltiples bases de datos, etc), en cambio, en el caso del Monolito solamente se requiere una instancia de cada tipo de servicio (*backend*, base de datos, etc) inicialmente. Si bien el hecho de que el Monolito es más barato en lo que respecta a costos dentro del contexto del proyecto, esto podría llegar a no ser verdad en el caso de que la aplicación crezca exponencialmente.

Debido a los puntos mencionados anteriormente, se tomó la decisión de utilizar el enfoque Monolítico. Con este enfoque, el equipo sintió que podría generar valor más rápidamente en la aplicación, y a su vez, al construir los servicios de la aplicación con bajo acoplamiento, se deja abierta la posibilidad de utilizar el enfoque de Microservicios en un futuro utilizando la técnica de *Monolith First* [43].

6.2.2.2 Typescript Vs. Javascript

Si bien el cliente solicitó como restricción que se utilizase el ambiente de desarrollo NodeJs para el desarrollo del proyecto (Res1), el equipo tenía libre elección de si utilizar Javascript o Typescript. Para seleccionar cuál de estas dos opciones sería el lenguaje de programación que se utilizaría en el *backend* del sistema, se decidió realizar una investigación. En esta, el equipo descubrió que sería mucho más sencillo mantener un código limpio y reusable utilizando Typescript, dado que es un lenguaje fuertemente tipado, orientado a objetos y que tiene todas las capacidades de Javascript, pero a su vez le agrega algunas herramientas extras.

Al ser Typescript un lenguaje orientado a objetos y fuertemente tipado, permite crear objetos con tipos definidos para utilizarlos en las diferentes capas de la aplicación, teniendo un chequeo de tipos en los mismos. Este chequeo permite evitar o notar errores de tipos al compilar el sistema, dando más seguridad al desarrollo que en caso de Javascript, puesto que al no tener estos chequeos, dichos errores podrían pasar desapercibidos. Typescript también cuenta con interfaces, con las que se puede realizar un desarrollo más limpio, permitiendo realizar dependencias en abstracciones y no en clases. Dichos elementos dan lugar a técnicas como la inyección de dependencias, que permiten evitar la creación de dependencias directas, favoreciendo la mantenibilidad. Por lo tanto, el uso de Typescript permitiría crear un sistema robusto, y que favorezca a la mantenibilidad (dado a la reducción de acoplamiento directo y posibilidad de cambiar fácilmente las implementaciones sin generar grandes problemas). Por estos motivos, se decidió utilizar esta tecnología para implementar el *backend* del sistema, a pesar del desconocimiento de los desarrolladores.

6.2.2.3 Estrategia de *Deployment*

A la hora de analizar qué estrategia sería la más eficaz de utilizar para los *deployments* de las distintas versiones del sistema, se analizaron diferentes factores que podrían influir en esta decisión. En primera instancia se encontró que al tener la aplicación un solo servidor corriendo por ambiente, no tendría mucho sentido utilizar una estrategia de despliegue muy compleja. Por otro lado, un punto importante fue que la aplicación, al ser desarrollada para una empresa que la quería utilizar en sus procesos laborales, no contaría con uso en los fines de semana. Por ello, el equipo decidió utilizar la estrategia de *Recreate* [44] (también llamada *All at once* [45]), la cual si bien genera un posible *downtime*, tiene la ventaja de que es la más simple de ejecutar. En el caso de que el equipo realizase los despliegues únicamente los fines de semana, que así lo tenían pensado, el *downtime* generado por esta estrategia no generaría problemas al usuario, dado que durante este período de tiempo el cliente no se encontraría utilizando la aplicación.

Si bien la implementación de la integración continua en los pipelines permitía hacer este proceso de despliegue de manera fácil y rápida, también había que actualizar la base de datos, proceso el cuál no se encontraba automatizado. Para hacer posible la actualización de los cambios en las bases de datos de las instancias de AWS, se utilizaron migraciones. Gracias a la funcionalidad de TypeORM, se configuró la generación de migraciones automáticas, que permitía detectar las diferencias entre los modelos y las tablas de base de datos, facilitando su generación. Estas luego podían ser exportadas y aplicadas manualmente en la base de datos, o aplicarse directamente desde la consola, mediante algunos comandos. Como algunas veces los cambios en la estructura de las tablas eran complejos y el cliente no quería perder sus datos, siempre antes de realizar la actualización de la nueva versión y aplicar las migraciones, se realizaba un respaldo (*backup*) de la base de datos con el fin de que, en el peor de los casos, se pudiera volver al estado anterior sin perder los datos del cliente.

6.3 Descripción general

La arquitectura general del sistema se divide principalmente en las soluciones de *frontend* y *backend*. Ambas partes fueron diseñadas pensando en las necesidades del cliente y el problema a resolver. Se desarrollaron intentando buscar para cada caso una solución eficiente y que cumpla con los requerimientos y atributos de calidad establecidos. Esto con el fin de que el resultado final pudiese lograr alcanzar o superar sus expectativas.

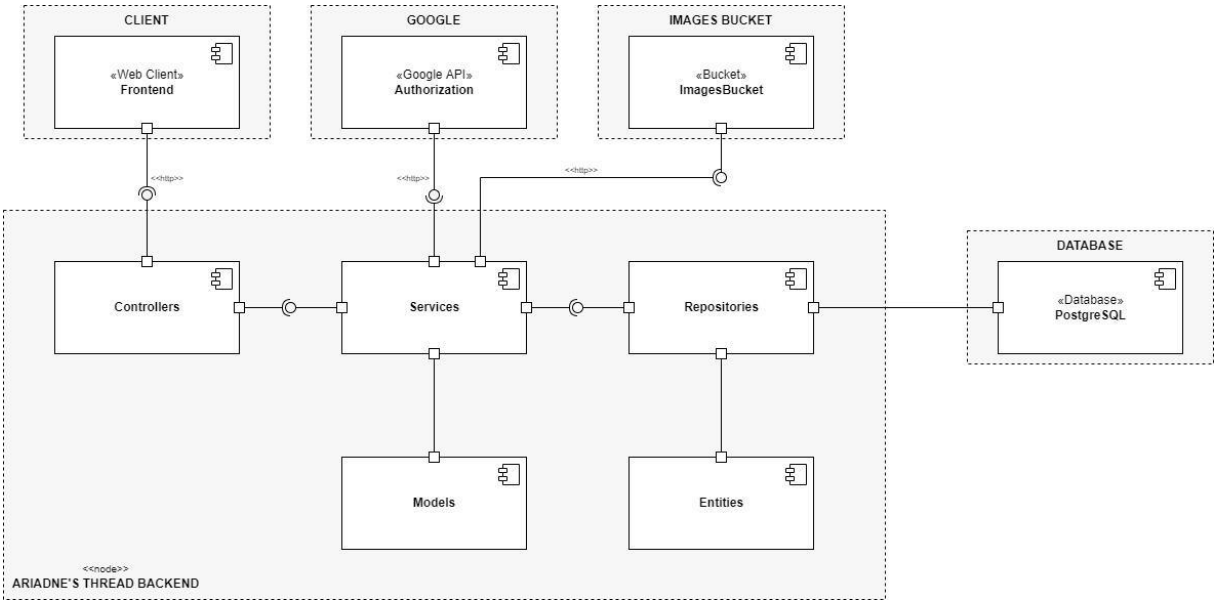


Figura 6.3: Arquitectura de la solución

6.3.1 Frontend

El *frontend* del sistema consiste en una página web generada con la librería React [46] y utilizando el lenguaje de programación Javascript. Como se mencionó anteriormente, esta se encuentra *deployada* en un *bucket* S3, y utiliza CloudFront como *cache* para mayor velocidad de cargado. Esta se conecta con el *backend* del sistema a través de una conexión encriptada HTTPS.

6.3.2 Backend

El *backend* del sistema consiste en un servicio de API que se encuentra implementado en el ambiente NodeJs con el lenguaje de programación Typescript (como mencionado anteriormente). Este se encuentra construido en base al patrón de arquitectura de software, *Layered Pattern* [45], de modo que los módulos se encuentran divididos definiendo diferentes niveles lógicos del sistema, teniendo responsabilidades específicas para las distintas clases, lo que permitió un desarrollo más ordenado y eficiente. Para realizar la comunicación entre las diferentes clases, se siguió el principio de inversión de dependencias (DIP) [47]. Para este fin, se crearon interfaces de modo que las clases dependan de estas y no de la clase concreta. Esto fue realizado con la ayuda de la librería typedi, que permite realizar la inyección de dependencias entre las clases del sistema. La misma permite que mediante configuraciones se puedan definir las implementaciones concretas para las interfaces, y luego estas sean

"inyectadas" en las clases que las necesiten, quitando la responsabilidad de las clases de instanciar estas dependencias. Esto permite disminuir el impacto en caso de posibles cambios de implementación, lo que favorece la mantenibilidad. Los componentes que conforman el *backend* son:

***Controllers* (API)**

Este componente es el encargado del manejo de la API, controladores, enrutadores y *middlewares*. Para la implementación de la API del sistema se decidió utilizar el *framework web Express* [48], ya que este es fácilmente configurable, tiene alta *performance* y es sencillo de utilizar. En la misma se utilizó REST [49] como estilo arquitectónico de diseño de *endpoints*, con el objetivo de lograr que los mismos fuesen fáciles de comprender, escalables e independientes.

Services

El componente de servicios de la aplicación es la que contiene toda la lógica del negocio. En él se encuentra todo lo relacionado a cómo debe funcionar la aplicación, la interacción entre las diferentes funcionalidades del sistema, y más. Este se encuentra conectado al componente de Modelos (*Models*) del sistema, dado que lo utiliza para realizar acciones con los elementos del dominio. También se encuentra conectado con el acceso a datos (*Repositories*), ya que, entre las operaciones del negocio, muchas veces se encuentra la habilidad de guardar, modificar, obtener o eliminar datos del sistema.

Models

En este componente se encuentran todos los elementos que conforman el dominio de la aplicación, como por ejemplo, las estructuras que mantienen los datos de los empleados, proyectos y equipos. Estos elementos son utilizados por la capa de servicios (como mencionado anteriormente) para realizar acciones de negocio.

Repositories

En el componente de *Repositories*, también conocido como acceso a datos, se mantiene toda la lógica referente a la conexión entre el sistema y la base de datos. El mismo utiliza el ORM [50] TypeORM [51] para todo lo relacionado a la conexión con la base de datos. El equipo decidió utilizar el mismo dado a la practicidad que este brinda, ya que facilita la realización de *queries*

a la base de datos, genera las diferentes tablas en base a entidades definidas en el código y tiene capacidad de generación automática de migraciones.

Entities

Es el componente encargado de manejar todos los elementos que se persisten en la base de datos del sistema. Este cuenta con entidades de TypeORM que hacen referencia a los elementos que serán guardados en las tablas de la base de datos. A su vez, cuenta con clases que convierten elementos del dominio a entidades, lo que facilita el trabajo de interacción entre la capa de acceso a datos y la capa de servicios.

6.4 Atributos de calidad

En esta sección se hablará de los principales atributos de calidad tomados en cuenta para la solución, y se describirán las tácticas utilizadas por el equipo para lograr satisfacer los mismos. Las tácticas utilizadas en esta sección se encuentran planteadas dentro del libro de arquitectura de software, *Software Architecture in Practice* [52].

6.4.1 Modificabilidad

La modificabilidad de un programa refiere a la capacidad del mismo de poder implementar cambios necesarios, minimizando el impacto que estos cambios generan, y los costos de realizarlos.

Split module

La separación de módulos es una táctica de arquitectura que refiere a que si un módulo del sistema comienza a crecer y se vuelve muy grande, el costo de modificación del mismo se vuelve muy alto. La solución a este problema sería dividir dicho módulo en múltiples módulos, lo cual haría que su costo de modificación baje significativamente.

Esta táctica se realizó tanto en el *frontend* del producto como en el *backend*. El desarrollo del sistema fue generado utilizando una metodología ágil, y con un conjunto de funcionalidades, que si bien estaba definido inicialmente, no estaba claro que todas formasen parte del producto final. Esto causó que el equipo debiera ir avanzando en el desarrollo del sistema sin total certeza de qué componentes se implementarían en futuras iteraciones. Para facilitar la introducción de nuevos cambios en las iteraciones próximas, los módulos que empezaban a ser muy grandes y

complejos se separaban, cuidando que cada uno de estos tuviese solamente una responsabilidad. Si bien esto llevaba una carga importante de trabajo cada vez que se realizaba, el resultado ayudaba a que se minimizara la carga de trabajo que implicaría realizar modificaciones dentro de los diferentes módulos del sistema.

Increase semantic coherence

La táctica de aumento de coherencia semántica describe que en el caso de que existan dos elementos que tienen responsabilidades diferentes, dichos elementos deben ser ubicados en módulos diferentes. Por ello, en el desarrollo del sistema se trató de mantener cada uno de los módulos altamente cohesivos, intentando mantener en estos, los elementos que cumpliesen una misma responsabilidad. Debido a esto, y junto al punto anterior, cada vez que el equipo detectaba que se estaban manteniendo múltiples responsabilidades en alguno de los módulos generados, se optaba por separar dichas responsabilidades en módulos distintos. Esto generaba que cada uno de los módulos tuviese sólo una responsabilidad, por lo que en caso de necesitar realizar cambios sobre dicha responsabilidad, el equipo sabía exactamente a qué módulo debía hacérsele cambios.

Encapsulamiento

Realizar encapsulamiento es la práctica de introducir interfaces explícitas a los diferentes módulos, para que en caso de que un elemento deba utilizar dicho módulo, este solamente deba conocer la interfaz y no la implementación específica de la misma. Esto permite una alta modificabilidad dado que los desarrolladores pueden cambiar la implementación de la interfaz, casi sin afectar al elemento que la utiliza.

El uso de interfaces en el *backend* del sistema fue una práctica constante, cada contacto entre los módulos de API y servicios, o entre los servicios y el acceso a datos se realizaba a través del uso de estas. Esto generaba el beneficio de que el equipo pudiese realizar cambios sobre la implementación de los diferentes módulos, o hasta cambiar totalmente la implementación de los mismos, sin afectar a los otros módulos. A su vez, el encapsular el uso de estos elementos detrás de una interfaz, permitía que las otras capas solo pudiesen utilizar las funciones que se exponían del módulo, haciendo que el desarrollo fuese más seguro.

Use an intermediary

El uso de intermediarios es una práctica común en la industria cuando se quiere romper dependencias entre elementos. Generalmente se realiza mediante la creación de una clase o elemento extra, a la cual se apunta la dependencia con el fin de evitar dependencias directas entre clases. Esto permite evitar el acoplamiento directo, bajando el impacto de posibles cambios de implementación en estas, y favoreciendo la modificabilidad del sistema.

Se utilizaron intermediarios en el código del sistema en muchas ocasiones. Estos fueron utilizados para romper dependencias directas, y a su vez, para lograr manejar la asincronía de algunos elementos (como llamadas al *backend*). Un claro caso del uso de esta táctica en el *frontend* del sistema es el uso de Redux [53], el cual es un manejador de estados de variables que permite separar la lógica de actualización de las variables, de la lógica de obtención de sus valores. Esta librería fue utilizada muchas veces como intermediario para eliminar el conocimiento entre productores de datos (como por ejemplo, llamadas asincrónicas al *backend*), y los consumidores de los mismos (componentes que utilizaban los datos retornados). Esta táctica fue muy útil dado que permitía que el productor se encargase de generar los datos, mientras que el (o los) consumidores podrían estar simplemente a la escucha del dato. Esto facilitó enormemente la interconexión entre las diferentes partes del sistema, y a su vez, evitó el acoplamiento entre las mismas.

En cuanto al *backend*, como ya se mencionó, se implementaron interfaces para que las dependencias sean sobre abstracciones y no clases concretas. También se configuró la inyección de dependencias, la cual se encarga de proporcionarle a las clases las instancias de sus dependencias, logrando evitar las dependencias directas entre ellas y favoreciendo la modificabilidad.

Refactor

El refactorizar el código es la técnica de detectar cuando un elemento de código es prácticamente una copia de otro elemento, y modificar el código para que se genere un nuevo elemento que pueda ser reutilizado para ambos casos. Para lograr que el código desarrollado por el equipo fuese de alta calidad, se utilizó esta táctica en múltiples ocasiones. Esto quiere decir que cada vez que el equipo se encontraba con código que podría ser reutilizado en la aplicación, dependiendo de la dificultad del cambio y del tiempo disponible con el cual se

contaba, se optaba entre rehacer el bloque de código para poder reutilizarse o crear un *ticket* de deuda técnica para hacer dicho cambio en un futuro. Esto causó que el código estuviese en constante proceso de revisión y cambios para asegurar que su calidad fuese la mejor posible. En múltiples ocasiones, cuando alguno de los integrantes tenía dudas de cómo volver reusable al código generado, o necesitaba ayuda para diseñar una mejor solución para lograrlo, el equipo se juntaba a discutir posibles cambios que se pudieran realizar para hacerlo posible.

Abstract common services

Esta fue una de las tácticas utilizadas para cumplir con el objetivo de promover la modificabilidad del sistema, y favorecer su mantenibilidad. La misma se basa en intentar, en los casos que sea posible, generalizar o abstraer funcionalidades con el fin de que se implemente el elemento una sola vez, y luego pueda ser utilizado en los distintos casos que se necesite. Esto no solo permite mejorar el código, ya que evita la creación de posible código repetido, también favorece a la modificabilidad ya que, en caso de un cambio, solo se debe cambiar un lugar (la abstracción) en vez de múltiples lugares.

Esta técnica se utilizó en el *frontend* del sistema cuando se realizaron los componentes reusables. Se puede ver un ejemplo de esto en la sección de “Reuso de componentes” dentro de este capítulo.

En el *backend*, también se utilizaron abstracciones en diferentes lugares. Un ejemplo de esto es el caso de las validaciones para asegurar la correctitud de las peticiones. Para estas, se implementaron funciones que permiten hacer estas validaciones de forma genérica, pudiendo usarse luego en sus lugares correspondientes. También, en el caso de las categorías (que son utilizadas por todas las entidades) se decidió centrar la implementación en base a estas, de modo de evitar código repetido entre las clases de entidades, dejando la implementación en un solo lugar.

Defer binding

Esta táctica se basa, en términos generales, en que es mejor para el sistema si el usuario puede modificar las variables de este mediante su interfaz visual, contra el caso de tener que modificar el código del mismo. Siempre es mejor poder modificar estos valores en etapas más tardías de la fase de desarrollo [54], dado que, en la mayoría de los casos, estos cambios suelen ser menos costosos.

El equipo utilizó esta táctica en algunas de las funcionalidades claves del sistema para permitirle al cliente realizar cambios sobre las mismas. Esto para que dichas funcionalidades puedan adaptarse más a la realidad de la empresa en todo momento. Una de las principales funcionalidades que hace uso de esta táctica es la épica "E-4, Gestión de Categorías". Esta se encarga de hacer posible que el usuario cree todas las categorías que quiera, con sus diferentes subcategorías, y pueda asignarlas a las diferentes entidades (Equipo, Persona y Proyecto). Por ende, el sistema deja en manos del usuario la posibilidad de configurar qué categorías aplican para qué entidades, y a su vez permite generar nuevas subcategorías o eliminar/modificar las existentes. Esto genera versatilidad para el cliente, ya que este es libre de decidir qué categorías siguen siendo vigentes para cada entidad, y deja abierta la posibilidad de que cree nuevas o modifique las existentes para adaptarse a la realidad de la empresa.

6.4.2 Seguridad

La seguridad es un punto crucial para la aplicación ya que esta maneja datos sensibles y confidenciales de la empresa. El tener una buena seguridad en la misma brindará al usuario la tranquilidad de que los datos no podrán ser modificados por personal no autorizado, y que la información solo podrá ser accedida por usuarios que cuenten con los permisos suficientes.

Identify actors

Esta táctica refiere a identificar el origen de las entradas externas sobre el sistema. Para este fin, los usuarios cuentan con un email único de acceso, el cual permite identificarlos correctamente cuando inician sesión mediante la cuenta de Google (proceso el cuál será explicado más adelante). A su vez, también se cuenta con un identificador único global (GUID), el cuál es generado y asignado en el momento en que se registra el usuario, y permite su posterior identificación en las peticiones.

Authenticate actors

Un punto importante para lograr un sistema seguro, es asegurarse, valga la redundancia, de que los actores sean quien dicen ser. En esto se basa la táctica de autenticación. Por petición del cliente, se tenía que el inicio de sesión debía ser mediante Google, con el fin de que los usuarios no tuvieran que crearse nuevas credenciales para acceder al sistema, y pudieran ingresar fácilmente mediante sus cuentas empresariales de Google. Para implementarlo, el equipo decidió utilizar el patrón de *Federated Identity* [55]. De esta manera, el sistema se desprende

de la responsabilidad de la autenticación de los usuarios, y se la delega a un proveedor de identidad de confianza (en este caso Google).

A continuación, en la figura 6.4 se observa un diagrama de flujo del proceso de *login* con Google que sigue el sistema.

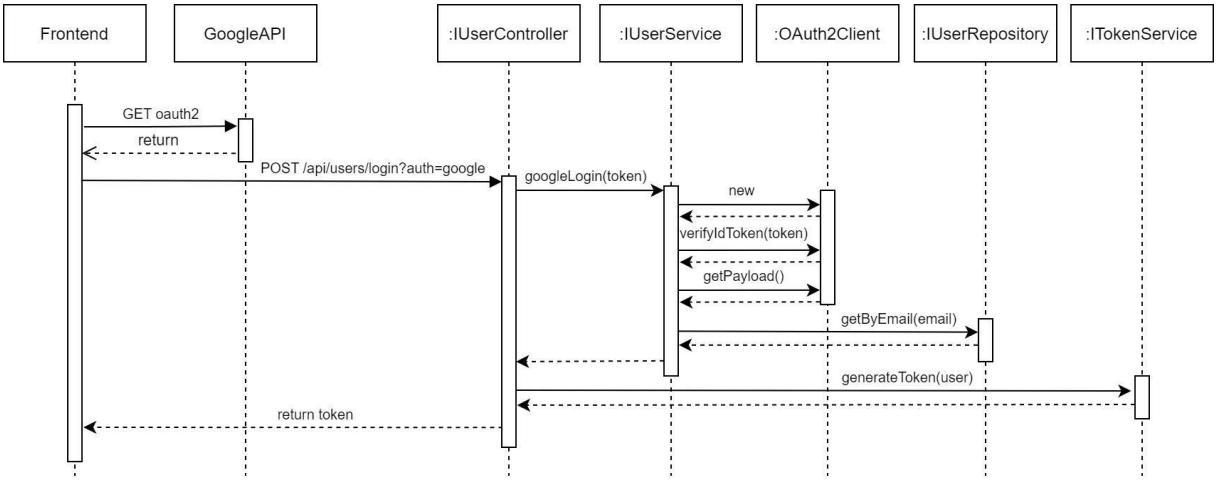


Figura 6.4: Diagrama de Secuencia del *Login* con Google

Al querer un usuario iniciar sesión en el sistema, deberá hacerlo mediante el botón de inicio de sesión con Google, encontrado en la pantalla inicial de la aplicación. Al hacer click en este, deberá ingresar sus credenciales, y si Google detecta que es un usuario válido, genera un token con su información, el cual es luego enviado al *backend* de la aplicación mediante una petición de inicio de sesión, para que el usuario pueda ingresar al sistema. El *backend*, al recibir esto, desarma (con ayuda de los servicios de Google) el token para obtener la información del usuario, y chequea que esta se corresponda con la de un usuario válido. En caso de que lo sea, el sistema le genera un nuevo token con algunos de sus datos registrados (los cuales sirven para luego identificarlo). Este token es enviado al *frontend*, el cual se deberá encargar de luego proporcionarlo en cada petición al *backend*, para que se pueda validar al usuario correspondiente, y realizar los procesos necesarios para responderle. Para hacer posible el manejo de tokens propios del sistema, se decidió utilizar Json Web Token (JWT), ya que es una herramienta ampliamente usada y testeada, que permite proporcionar un buen nivel de seguridad en la aplicación. Con estos dos mecanismos se puede asegurar la autenticación de los actores sobre el sistema, permitiendo el acceso solamente a usuarios válidos.

Además de los beneficios que brindó al sistema el uso de esta táctica, la misma permitió al equipo cumplir con el requerimiento no funcional RNF 10, lo que ayudó al equipo a finalizar dicha petición del cliente.

Authorize actors

La táctica de autorizar actores se basa en asegurarse de que un usuario tenga los permisos necesarios para acceder y/o modificar los recursos que está solicitando. Para esto, en base a las necesidades del cliente, se creó un sistema de roles en el que se definieron las capacidades de los diferentes usuarios. En cuanto a la parte del *frontend*, se crearon restricciones de acceso en la navegación, y vistas personalizadas para los diferentes tipos de usuario, de manera que estos solo puedan ver y acceder lo que tienen permitido. Respecto al *backend*, todos los *endpoints* (menos el de inicio de sesión y el de chequeo de salud) se encuentran debidamente asegurados. Las peticiones deben realizarse proporcionando un token de sesión, y el sistema corroborará la validez del usuario (es decir, que este exista y no esté eliminado) y que cuente con el rol necesario para acceder al recurso. En caso de que alguna de estas condiciones no se cumpla, no se le permite el acceso.

Limit Exposure* y *Limit Access

Estas tácticas se basan en limitar la exposición de un sistema o componente, y limitar el acceso no deseado a ciertos elementos del sistema, con el fin de reducir la probabilidad de potenciales ataques. Para este fin, se definieron y delimitaron los accesos a las instancias de AWS. La instancia de RDS (que contiene la base de datos) solo se encuentra accesible dentro del entorno de las instancias de los otros servicios de AWS, negando su acceso de cualquier petición proveniente del mundo exterior. En cuanto a la instancia de EC2 (que aloja al *backend* del sistema), este expone solamente un puerto, haciendo por tanto que solo tenga un punto de acceso. También como se mencionó, los *endpoints* que expone la API se encuentran debidamente controlados, validando para los diferentes casos que el actor de la petición sea un usuario válido y cuente con los permisos necesarios para utilizarlos.

Encrypt Data

La táctica de encriptación refiere a que la información se encuentre protegida de acceso no autorizado o deseado. Para este fin, para las instancias de producción se incorporó HTTPS para cifrar la comunicación entre cliente y servidor. De esta manera se asegura que los datos que

viajan entre ambas instancias se encuentran encriptados, proporcionando una mayor seguridad al sistema y cumpliendo con el requerimiento no funcional RNF 11.

6.4.3 Disponibilidad

La disponibilidad es la habilidad que tiene un producto de *software* de estar siempre disponible cuando los usuarios lo necesitan. Esto quiere decir que siempre que un usuario deba utilizar la aplicación, esta debería encontrarse en un estado que le permita responder correctamente a los estímulos generados por este.

Ping / echo

La idea detrás de esta táctica es básicamente enviar mensajes entre dos partes con el fin de determinar si la otra parte se encuentra activa. Con el fin de poder monitorear la salud y disponibilidad del sistema, se creó un *endpoint* en el *backend* para chequear su estado de salud. Este puede ser invocado mediante una petición de GET, para verificar rápidamente que la aplicación está activa y que los servicios asociados (como la base de datos) se encuentren disponibles. Al ser invocado, realiza los chequeos correspondientes para obtener el estado de los servicios, en caso de que esté todo correcto, retorna una respuesta exitosa, en caso contrario retorna una respuesta con código de fuera de servicio y se emite un *log* para que quede constancia de lo sucedido. Este mecanismo fue implementado en base a esta táctica, dado que al realizar la petición se intenta comunicar con sus servicios para ver el estado de estos y responder acorde a lo obtenido en el chequeo. Si bien el tener este *endpoint* público podría llegar a generar una vulnerabilidad de seguridad, es necesario para permitir chequear el estado de salud del sistema, que en otro caso, la responsabilidad debería recaer en el sistema teniendo que chequearse a sí mismo continuamente, pudiendo afectar la *performance* del sistema debido a todo este procesamiento.

Exception handling

Esta táctica trata acerca de manejar correctamente las excepciones en un sistema con el fin de mantenerlas controladas y que no afecten a la disponibilidad del mismo. Para este fin, en el desarrollo se intentó siempre ir un paso más allá y pensar en los casos que podrían generar un error, además se implementó un manejo de excepciones para evitar que el sistema pierda disponibilidad. En caso de que ocurra una excepción mientras se está ejecutando una petición, el sistema responde con un código de error informando que ocurrió un error y además se emite

un *log* con información acerca del error y dónde ocurrió el mismo, para luego poder realizar su debida trazabilidad.

Ignore Faulty Behaviour

Esta táctica se basa, en grandes rasgos, en omitir mensajes sospechosos o que no tengan el formato adecuado para evitar potenciales ataques. Dentro del sistema se implementó una serie de modelos para definir el cuerpo de las diferentes posibles peticiones, estos son utilizados para verificar que el cuerpo de las peticiones sea el adecuado, o en caso contrario, negar la petición y enviar una respuesta acorde. Además de estos controles, se implementaron *middlewares* que se encargan de examinar la correctitud de las *queries* en las peticiones, para evitar que sigan su camino si se detecta que no están bien formadas. Además de esto, como se mencionó, se utilizan *tokens* de sesión para controlar el acceso al sistema, en donde si estos no están bien formados o no corresponden a un usuario válido, ya sea porque no está registrado, o está eliminado, o no cuenta con el rol que necesita, se le deniega el acceso.

Transactions y ***rollbacks***

Mantener la consistencia de los datos y preservar la integridad de estos es importante, debido a que es en base a los datos del sistema que el cliente tomará sus decisiones de negocio. Con esto en mente, se intentó en el desarrollo tener cuidado con el manejo de datos, con el fin de intentar asegurar que los distintos flujos preserven la consistencia de estos. Para los procesos en los que se encontraban involucradas varias entidades, como por ejemplo un movimiento de una persona entre equipos, se decidió hacer uso de las transacciones que provee TypeORM. Gracias a estas, se pueden realizar estos procesos de forma segura, garantizando la correcta actualización de todas las entidades asociadas, o en caso contrario, y que ocurra una falla, aplicar *rollback* con el fin de que se anulen por completo los posibles cambios realizados, y así permitir que los datos se mantengan consistentes.

Además de este mecanismo, con el fin de garantizar la correctitud de los datos y no tener datos innecesarios, en el caso de los movimientos de personas entre equipos, se implementó un algoritmo que permite eliminar registros de movimientos de personas entre equipos que se realizaron en menos de una semana, ya que este no es un caso válido.

Exception prevention

Esta táctica refiere a intentar prevenir que ocurran excepciones en el sistema, como ya se mencionó, se crearon modelos que definen la estructura de los objetos de las peticiones, que permiten asegurar la correctitud de estas. Además, para evitar lanzar excepciones y cortar el flujo del sistema innecesariamente, se crearon objetos para realizar la comunicación entre clases. De esta manera se puede notificar y verificar el estado de las peticiones entre los diferentes pasos de su procesamiento, pudiendo ver si el mismo fue satisfactorio o tiene errores, antes de seguir al siguiente paso. Esto permite detectar un error fácilmente y responder con un mensaje de respuesta acorde, sin necesidad de cortar el flujo lanzando una excepción de manera forzosa. En adición a los chequeos de correctitud de los datos, también se realizan chequeos de unicidad, entre otros, con el fin de verificar que los datos se encuentren correctos antes de ser ingresados a la base de datos y prevenir que lleguen a esta datos erróneos o que pueden causar una excepción. Este doble factor de chequeos no solo permite evitar posibles excepciones y errores en la base de datos, también proporciona un nivel de seguridad extra, ya que no se recae únicamente sobre los chequeos establecidos en la base de datos, sino que se toman a estos como última instancia. Igualmente, en caso de que falle algún chequeo del sistema y se termine generando un error en la base de datos, como ya se mencionó, se tienen mecanismos para manejar estas excepciones.

Además de estas tácticas que permiten promover la disponibilidad del sistema, otro factor que favorece a la disponibilidad es el hecho de que todos los sistemas externos utilizados son de AWS o Google. Esto permite cumplir con el requerimiento no funcional de disponibilidad (RNF 12), ya que se sabe que estos servicios son altamente disponibles y brindan un buen servicio a sus clientes. Para tener una noción del porcentaje de disponibilidad que podría tener el sistema, se investigaron los acuerdos de SLAs de los diferentes servicios utilizados y luego se hizo un promedio con estos valores, obteniendo un total de 99,75%.

Servicio	**SLA**
EC2	99,5
RDS	99,5
S3 (*webApp* y fotos)	99,9
SSO Google	99,95

CloudFront	99,9

Tabla 6.1: SLAs de los diferentes servicios

Con este valor y tomando en cuenta diferentes causalidades que podrían ocurrir, o algún error no encontrado del sistema que generase una caída de este, es que se tomó el valor de 90. Dicho valor aplica únicamente dentro de la semana laboral, dado que es cuando ocurrirá el uso del sistema y debe estar activo. Los fines de semana, o eventualmente algún día fuera del horario de uso normal del sistema, se utilizan para liberar las nuevas versiones, como ya se mencionó, en este proceso puede que haya un período en el que el sistema se encuentre fuera de servicio, hasta que se terminen de realizar todos los cambios y quede actualizada la nueva versión. Es por esta razón que en estos lapsos no se asegura la disponibilidad.

6.4.4 Testeabilidad

La testeabilidad es la facilidad que tiene un sistema de *software* para lograr encontrar sus fallas, pudiendo estas encontrarse a base de pruebas manuales o automáticas. El lograr una mejor testeabilidad en el sistema permite al equipo generar más fácilmente pruebas con las cuales detectar funcionamientos erróneos en las diferentes funcionalidades del sistema. En particular para el proyecto, se realizaron tanto pruebas manuales como pruebas automatizadas.

Abstract data sources

La táctica de *abstract data sources* refiere a la capacidad que tiene el sistema de poder fácilmente alterar el estado en el que se encuentra. Esta tiene como beneficio que permite al desarrollador poder probar diferentes funcionalidades en distintos estados diferentes del sistema. Existen variadas formas de cómo alterar el estado del sistema para las pruebas. Un ejemplo de ello es tener interfaces específicas para el *testing* dentro del código de la aplicación. Otra forma muy utilizada es generar una base de datos para las pruebas, la cual se pueda alterar fácilmente desde los *tests*. El equipo optó por utilizar esta segunda opción dado a que es sencillo de utilizar, y hace que las pruebas no afecten directamente al código del sistema. Con esa base de datos de pruebas el equipo pudo generar pruebas automatizadas para casos específicos del sistema, pudiendo así probar escenarios de casos borde que podrían ser muy difíciles de realizar a mano.

Sandbox

Esta táctica refiere a tener una instancia del sistema aislada de los usuarios reales del mismo. Dicha instancia permite a los desarrolladores realizar pruebas en un ambiente controlado que luego no repercutan en el usuario final. El equipo generó para el sistema una instancia de desarrollo de la aplicación para poder tener esta instancia de pruebas. En ella se podían probar funcionalidades nuevas, a las que el cliente no tuviese acceso todavía, realizar arreglos o cambios sobre las funcionalidades ya generadas, y hasta realizar demos para el cliente en las cuales mostrarle nuevas funcionalidades que no estaban finalizadas para que este pudiese proveer *feedback* al equipo. Dicha instancia fue muy útil para el desarrollo del sistema dado que gracias a ella era posible probar fácilmente las funcionalidades nuevas del sistema, en un ambiente que fuese controlado, pero a su vez que se mostrase más realista comparado contra levantar el sistema localmente en la máquina del desarrollador. Una de las claves de haberla generado, fue que permitió a los desarrolladores ver rápidamente cómo se comportaba el código de funcionalidades que hacía poco habían implementado, junto a otras nuevas funcionalidades del sistema, pudiendo así rápidamente detectar *bugs* o fallas de usabilidad.

Limit structural complexity

Si los desarrolladores del sistema generan un código fácil de leer, que sea simple de modificar, y que tenga bajo acoplamiento y alta cohesión, estos elementos terminan promoviendo indirectamente a la facilidad de probar el sistema. La táctica de *limit structural complexity* describe justamente esto, que un código limpio hace más fácil la tarea de probarlo. Gracias a que el equipo utilizó múltiples tácticas de modificabilidad para mejorar el código, se prestó atención en factores como el uso de la nomenclatura correcta a la hora de nombrar las variables del sistema, y se apoyó de elementos como los principios SOLID [56], el sistema resultante fue fácil de probar y de encontrar fallas en el mismo.

6.4.5 Usabilidad

La usabilidad de un sistema es la facilidad con la cual los usuarios del mismo comprenden cómo utilizarlo para lograr sus cometidos. Este atributo de calidad es muy complejo de definir y medir dado que depende mucho del tipo de usuarios que sean objetivo para la aplicación, la familiaridad de estos con otros sistemas de software, entre otros factores.

En lo que respecta a la usabilidad del sistema, dado que los usuarios finales del sistema serían los integrantes del departamento de P&C de Qualabs, y que muchos de estos no tenían un nivel muy alto de alfabetización digital, este atributo de calidad fue tomado muy en cuenta para el desarrollo del sistema. Por ello, el equipo utilizó muchas técnicas y herramientas para lograr que la usabilidad final del sistema fuese del agrado de los usuarios. Entre los elementos que se utilizaron, como se menciona en el capítulo de "Marco de trabajo", el equipo incorporó a su proceso de trabajo la técnica de *Design Sprint* (se puede ver un ejemplo de esta técnica en el anexo de "*Design Sprint*"). Dicha técnica permitió que muchos de los diseños de pantallas que serían generados para el sistema final fueran validados por el cliente y usuarios previo a su implementación. Gracias a estas validaciones, y en base al *feedback* obtenido del cliente, el equipo pudo afirmar el conocimiento que se tenía sobre los procesos y flujos de la empresa. Esto hizo cada vez más rápido el proceso de generación de los diseños, dado a que el equipo comprendía mejor cómo solucionar los problemas del cliente, mejorando junto a ello la usabilidad final del sistema.

Otro de los elementos que se utilizaron para mejorar la usabilidad del sistema, fue el uso de las funciones asincrónicas en el sistema. Esto permitió que el cliente no tuviese que esperar a que todos los procesos se terminaran de ejecutar en la aplicación para poder seguir trabajando con la misma. Este tipo de funciones hizo que el sistema fuese más ágil de utilizar, haciendo que el cliente tenga menores tiempos de espera entre interacciones, evitando así posibles frustraciones que podrían haberse generado por tener que esperar mucho tiempo entre interacciones con la aplicación. Otro punto a través del cual el equipo intentó mejorar la usabilidad del sistema fue en base a hacer que el usuario comprendiese el estado en el que se encontraba el mismo en todo momento. Para ello se generaron elementos como las notificaciones, que avisan al usuario de si alguna interacción con el sistema falló, o si la misma concluyó correctamente.

Con el objetivo de lograr que los usuarios comprendiesen más fácilmente el funcionamiento de la aplicación, el equipo diseñó todas las páginas del sistema enfocándose en mantener la consistencia entre las mismas. Esto se realizó a través de cuidar que las páginas de la aplicación mantuviesen similitudes entre sí, haciendo que los elementos de acciones similares se encuentren siempre ubicados en la misma área de la pantalla. Gracias a esto, se facilitó que los usuarios pudiesen comprender en qué lugares de la aplicación se encuentran cada tipo de acciones.

Como se mencionó anteriormente, se prestó especial atención al *feedback* que brindaba el cliente, ya que este muchas veces generaba mejoras importantes de usabilidad en el sistema en base a cambios menores. Un gran ejemplo de esto es, como muestra la figura 6.5, el botón de cancelar. El mismo fue un elemento que se adicionó al sistema en base a los comentarios brindados por parte del cliente. Este elemento, si bien no es un gran cambio en lo que respecta a código, para el usuario fue una mejora importante de usabilidad, gracias a que con este tenían una forma clara y sencilla de como cancelar las acciones. Este fue uno de los múltiples elementos que se agregaron al sistema en base al *feedback* del cliente, y que si bien su complejidad de implementación no era alta, brindaba mucho en lo que respecta a la usabilidad (para ver el proceso seguido para adicionar estos cambios ver "Gestión del *feedback* recibido" del capítulo "Gestión de proyecto").

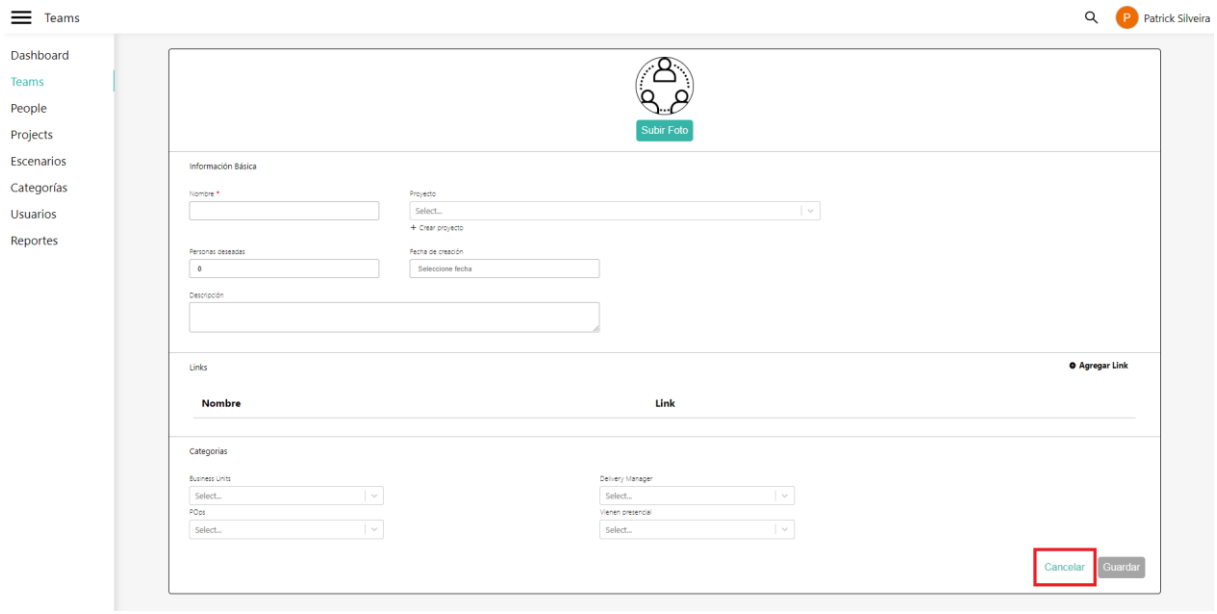


Figura 6.5: Botón de cancelar en ficha de equipo

En lo que respecta a las librerías utilizadas para el desarrollo de la aplicación, el equipo decidió cuáles serían las más óptimas en base a varios factores, siendo uno de ellos la usabilidad del sistema. Dado que una de las funcionalidades principales del sistema era el "Diagrama organizacional interactivo", y que esta necesitaba poder arrastrar elementos en pantalla (*Drag & Drop*), el equipo decidió realizar una investigación para encontrar la mejor librería que podrían utilizar para este caso. Tras dicha investigación el equipo seleccionó la librería *react beautiful dnd* [57], la cual es mantenida por la empresa Atlassian [58]. En esta librería, con el fin de mejorar su usabilidad, los desarrolladores de la misma llevaron a cabo múltiples

investigaciones para optimizar elementos como sus animaciones e interacciones. En el caso de las animaciones, por ejemplo, se buscó que las mismas fueran las más óptimas para que el usuario sintiese que el movimiento de los elementos fuese lo más natural posible. A su vez, dicha librería tiene gran accesibilidad al poder utilizarse con diferentes tipos de interacciones, como por ejemplo a través del uso del teclado, para permitir que personas con capacidades limitadas puedan hacer uso de la misma. Con estas ventajas de usabilidad, su facilidad de uso, y su gran rendimiento de *performance*, el equipo no dudó en utilizarla para el proyecto.

Para el desarrollo de las diferentes funcionalidades del sistema el equipo realizó prototipos, como mencionado anteriormente, pero en el caso de algunas de las funcionalidades más importantes fue necesario hacer validaciones extras además de estos. Un claro ejemplo de esto es nuevamente el "Diagrama organizacional interactivo", ya que, si bien se podría comprender perfectamente el funcionamiento de la interfaz visual del mismo, por medio de los prototipos, este necesitaba a su vez validar detalles extras que no se veían directamente reflejados en estos.

Para esta funcionalidad en particular se tuvieron que mantener pláticas de, entre muchas cosas, en qué momento guardar los cambios organizacionales realizados en el *backend* del sistema. Dado que este problema podría afectar tanto a la usabilidad del sistema, como a la arquitectura del mismo, se realizaron reuniones con los usuarios del sistema (departamento de P&C) y el departamento de Ingeniería del cliente, para poder llegar a la solución final. Las posibilidades que se tomaron en cuenta para esta discusión fueron las de hacer que cada movimiento en el diagrama impacte inmediatamente al *backend* del sistema, o hacer que se puedan realizar múltiples acciones sin afectar al *backend* y luego clickear un botón de guardado para impactar realmente estos datos en la aplicación.

La primera opción, si bien era la más inmediata, generaba tiempos de espera entre las acciones realizadas dado que se debía esperar a que una interacción se guardara correctamente para luego seguir utilizando el sistema. Además, debido a que las llamadas al servidor del sistema podrían fallar por diversas razones, se tenía que implementar un *rollback* visual al sistema para que en caso de fallos se volviese al estado anterior. Y, en caso de que el usuario pudiese editar en todo momento, generaría la posibilidad de que se moviese un empleado en el diagrama sin querer, y que el usuario pudiera no darse cuenta de esto.

Por otra parte, la posibilidad de realizar múltiples movimientos en el diagrama y luego guardarlos todos, permitía el uso más fluido de la aplicación. A contraparte, tenía desventajas

claras como el ser más complejo de implementar, en caso de que se generasen muchos cambios en el diagrama y que luego el guardado fallase, podría frustrar mucho al usuario, y además en caso de que el usuario olvidase guardar los cambios, estos nunca repercutirían contra el sistema.

Tras deliberación del equipo y el cliente, se tomó la decisión de implementar la primera opción. Esto ya que los usuarios lo veían como el método más intuitivo de usar, y a su vez dado que en base al *feedback* del cliente se le decidió agregar la posibilidad de activar y desactivar la interacción con el *Drag & Drop* en el diagrama, se evitaron algunas de las desventajas de esta postura. En el siguiente diagrama se observa el flujo que conlleva la realización de un movimiento de un empleado entre equipos para el sistema final.

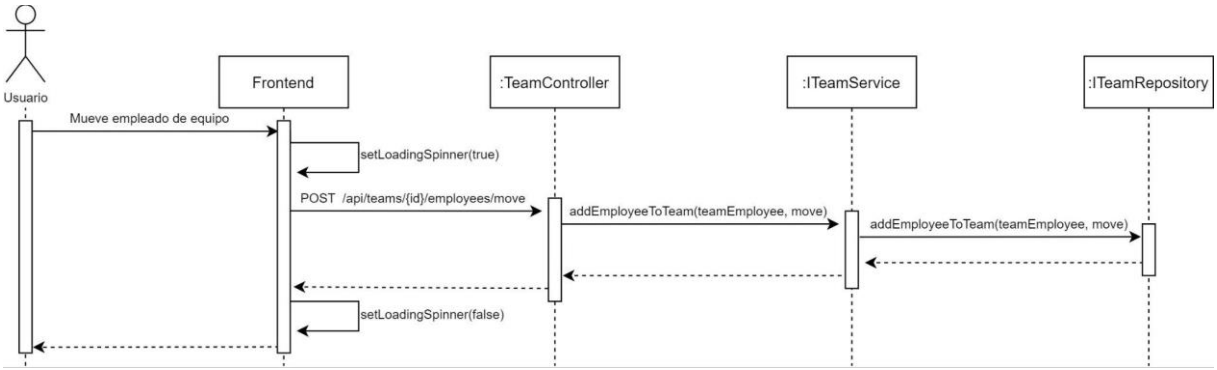


Figura 6.6: Diagrama de flujo del movimiento de empleados en el *Dashboard*

6.5 Conclusiones y lecciones aprendidas

En primer lugar, el equipo pudo notar el impacto del desarrollo incremental en la arquitectura, y con ello, la importancia de tomarse un tiempo para pensar en la implementación de la solución, de manera que esta esté abierta a los cambios. Gracias a que en el desarrollo se mantuvo el foco en promover el código limpio y en cuidar los atributos de calidad, es que se pudo ir adaptando satisfactoriamente el sistema en el tiempo. Este hecho fue beneficioso no sólo porque permitió ir modificando el sistema conforme se necesitara, sino también por los pequeños ahorros de tiempo que se pudieron lograr. Este último punto ayudó en parte a sopesar los contratiempos generados por el desconocimiento de las tecnologías, que en ocasiones provocó retrasos en el desarrollo o grandes dificultades (como fue el caso de la configuración de los *pipelines* y la creación de las instancias en la nube). El reuso y creación de componentes genéricos, principalmente en el *frontend*, fue de gran beneficio para sopesar la falta de recursos

con experiencia en esta área. Con esto, los integrantes con menos experiencia, pudieron en varios casos, utilizar dichos elementos, y disminuir en parte sus dificultades con estas tareas.

Si bien los *pipelines* llevaron su tiempo de configuración, brindaron una mayor agilidad en el proceso de desarrollo de las tareas, además de favorecer a la rápida entrega de valor. Lo que permitió que conforme se desarrollaba, se podía ver el avance en el ambiente de desarrollo, que era usado para mostrar los incrementos generados al cliente. También benefició la liberación de las nuevas versiones, ya que con simples pasos se podía incorporar de forma segura la actualización en la versión del cliente, permitiendo poner más foco en las otras tareas del proceso de liberación, que no eran tan sencillas.

El equipo considera que la arquitectura final se adapta a las necesidades del cliente y da lugar a que puedan mejorar sus procesos, permitirá lograr una mayor comunicación y visualización de la estructura de la empresa a diferentes áreas (que era lo que estaban buscando), además de organizar la información.

7 Gestión del proyecto

En el presente capítulo se narrará el proceso llevado a cabo para la gestión del proyecto, describiendo el proceso y planificación de los distintos *releases*, proceso y gestión del *feedback*, que fue una parte importante para el desarrollo, y los principales acontecimientos ocurridos que tuvieron impacto en el proyecto. También se mostrarán algunas de las métricas que tuvieron seguimiento a lo largo del tiempo, junto con sus valores obtenidos. Finalizando con la gestión de la comunicación y riesgos realizada.

7.1 Proceso

Antes de comenzar con el desarrollo, el equipo pasó por una etapa de descubrimiento, en la que se logró definir un plan inicial del proyecto, la primera versión del plan de *release* y un conjunto de funcionalidades priorizadas. Igualmente, dado que el sistema debía adaptarse a la empresa y su cultura, teniendo foco en el usuario, el descubrimiento estuvo presente en casi todo el proyecto. Esto, junto con el hecho de que la empresa estaba en pleno crecimiento y evolución (de su estructura y procesos), y se tuvo un *feedback* constante durante todo el proyecto, hizo que el sistema fuera evolucionando para adaptarse a la empresa del cliente. Como resultado, se fueron incorporando, eliminando y modificando requerimientos a lo largo del proyecto, generando cambios al plan de *release*. Este tuvo aproximadamente 5 cambios a lo largo del tiempo, tanto en funcionalidades como en tiempos, (como se puede ver en el anexo de "Evolución del plan de release").

Dado el contexto del proyecto, se decidió utilizar un marco de gestión ágil. Como se mencionó en el capítulo de "Marco de trabajo", se utilizó la metodología *Dual Track Scrum* debido al potencial de descubrimiento del proyecto. En este, se mantenían dos tableros distintos (pero coordinados) en JIRA, para la organización y manejo de las tareas de desarrollo y descubrimiento o investigación. En cuanto al tiempo de ejecución de los *sprints*, se decidió tomar lapsos de dos semanas. Antes de comenzar cada iteración, en algunos casos, se realizaba una serie de relevamientos o validaciones con el cliente para refinar requerimientos, y se preparaba el conjunto de historias (*backlog*) para el *sprint*. El proceso de ejecución comenzaba con la *planning*, terminaba con la *review* con el cliente y luego se realizaba la retrospectiva entre el equipo para analizar los resultados obtenidos (más detalles sobre este proceso se encuentran en el capítulo antes mencionado).

7.2 Gestión de *releases*

Como ya fue mencionado, por las características del proyecto, el plan de *release* fue cambiando en cuanto a su contenido a lo largo del tiempo. Para su definición, primero se llevaron a cabo reuniones con el cliente para descubrir y refinar requerimientos, luego estos se priorizaron, tratando de armar conjuntos de funcionalidades que brindaran valor al cliente y pudieran desarrollarse juntas. Definido esto, y tomando en cuenta la limitante de tiempo (establecida por la entrega académica), dejando un lapso para las actividades del cierre del proyecto (armado de documentaciones, *testing* y arreglo de *bugs*), se estableció la cantidad de *releases* que habría. Luego, el equipo analizó estos *releases*, y a grosso modo, se estimaron los tiempos que se creía iba a llevar su desarrollo, con el fin de establecer las posibles fechas de entrega de cada uno al cliente (tomando en cuenta el tiempo final establecido en el paso anterior).

Este análisis tenía mucha incertidumbre y el desarrollo podía llevar más o menos tiempo del previsto, por este motivo, el equipo decidió realizar un proceso de refinamiento junto al cliente, previo a comenzar cada *release*. Esto tenía el fin de validar las funcionalidades planeadas, tener un mayor entendimiento de lo que se iba a desarrollar, o realizar cambios pertinentes tanto en las definiciones de requerimientos cómo en el conjunto de estos en sí, para poder tener una noción del plan, previo a iniciar, y actuar conforme a lo necesario. Además, como se mencionó en el capítulo de “Ingeniería de requerimientos”, periódicamente se realizaban validaciones o refinamientos sobre los requerimientos para asegurar su correctitud y alcance antes de su desarrollo. Luego de empezar el desarrollo, ya con mayor noción de lo que se tenía que desarrollar, se realizaba nuevamente la estimación de tiempo de los *releases*, con el fin de tener una idea más clara del alcance y cumplimiento de los tiempos. En caso de que se vieran grandes diferencias en cuanto al tiempo planeado, se hacía un refinamiento y re-priorización de requerimientos junto al cliente, con el fin de establecer, dentro de lo ya planeado, el conjunto de requerimientos que brindaría mayor valor y dejar lo otro como deseable. Este proceso se siguió a lo largo del proyecto para mantener actualizado el plan de *release*, ejecutándose conforme se iban generando cambios, con el objetivo de maximizar el valor entregado.

Si bien al establecer las fechas se intentaba tomar el peor caso y prever márgenes de tiempo amplios procurando cubrir posibles contratiempos y malas estimaciones, en los primeros dos *releases* no fue posible llegar a la fecha planeada. Afortunadamente, el equipo pudo percatarse de esto con tiempo suficiente y le notificó al cliente, el cual no tuvo inconveniente con los

cambios de fechas. El hecho de mantener al cliente actualizado sobre el plan de *release*, en donde veía los cambios que iba teniendo (pudiendo ver sus solicitudes incorporadas en el mismo), y el estado en el que estaba el sistema respecto al plan, fue crucial. Esto, no sólo permitió al cliente ser consciente del valor que le estaba siendo entregado, también potenció este proceso y permitió al equipo ser transparente en el desarrollo. Debido a que la transparencia es un factor muy importante en la cultura del cliente, esto resultó en un mayor vínculo entre el cliente y el equipo, dando lugar a devoluciones y encuentros más profundos por la confianza lograda, y generación de más feedbacks, lo que permitió al equipo entender mejor al cliente y potenciar el valor entregado.

7.2.1 Proceso de liberación de *releases*

Previo a liberar cada *release*, el equipo armaba un archivo, el cuál contenía el número de la versión, el listado de sus funcionalidades y una breve descripción de estas, junto con los *links* a los ambientes en la nube de desarrollo y producción. Se decidió darle acceso al cliente a ambos ambientes para que tuviera la posibilidad de probar el sistema, para familiarizarse con este, sin "ensuciar" sus datos reales, y además, ver por sí mismos las funcionalidades que se iban desarrollando (ya que este ambiente era el utilizado para mostrar el incremento de los *sprints* en las *reviews*, y luego en cada *release*, se actualizaba la versión del cliente). La liberación de las nuevas versiones se intentaba realizar el fin de semana o en las noches, en ventanas de tiempo en las que se sabía el cliente no iba a utilizar el sistema, con el fin de evitar que el sistema quedara fuera de servicio (promoviendo el cumplimiento de RNF 12). Se puede ampliar sobre el proceso de *deploy* en el capítulo de "Arquitectura y Diseño". Una vez que la nueva versión se encontrara funcional, en su respectivo ambiente, se le notificaba al cliente mediante un mensaje y se le entregaba el archivo mencionado (el cuál era actualizado con cada versión).

Para el primer *release*, el equipo registró en producción algunos equipos y empleados reales de la empresa (con sus respectivas fotos, logos y datos básicos), y algunos otros datos. Se tomó esta decisión debido a que el sector de P&C no tenía mucho conocimiento en los procesos de este tipo, y venían viendo datos reales en el ambiente de desarrollo, por lo tanto, el encontrarse con la versión de producción sin estos datos, podría disminuir el impacto positivo esperado sobre este suceso. Este proceso se realizó gracias al conocimiento del personal de la empresa que tenía Patrick (integrante del equipo y único trabajador de Qualabs en ese momento). Al ver el cliente algunos de sus datos reales registrados, le dio una grata sorpresa (agradeciendo al

equipo por ella) y permitió que pudiera ver en mayor medida el valor del sistema en desarrollo, y contrastarlo con lo que estaba utilizando en ese momento para hacer sus tareas.

7.2.2 Plan de *release*

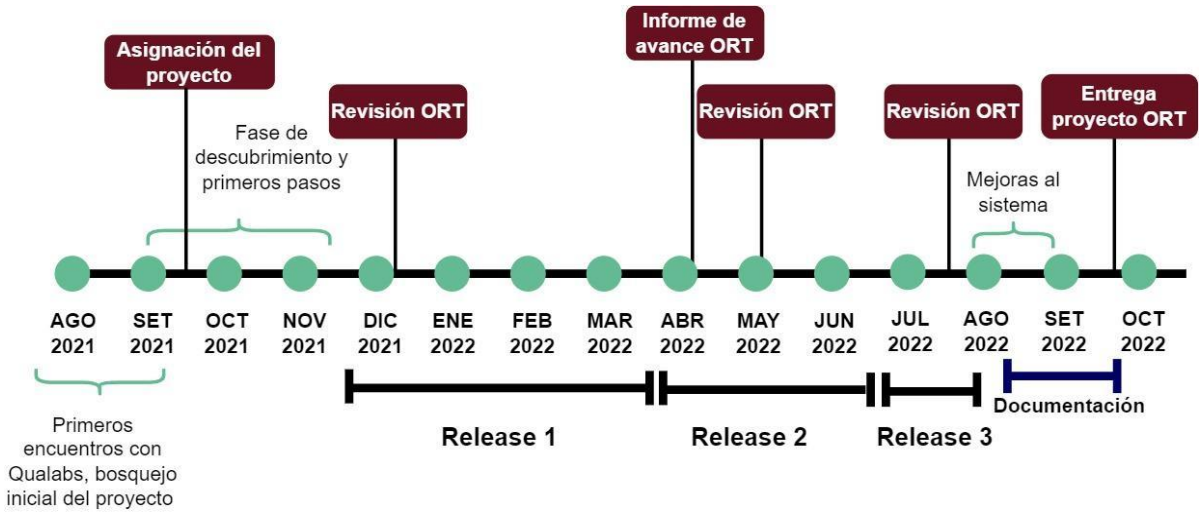


Figura 7.1: *Releases* y sucesos del proyecto en el tiempo

La versión final del plan de *release* fue la siguiente:

***Release* 1:**

Duración: 22/11/21 al 27/03/22

Sprints: 1 al 9

Funcionalidades principales:

- Lectura, creación, eliminación y actualización de Equipos, Proyectos y Personas
- Diagrama organizacional interactivo (con *Drag & Drop*)
- Vista de listas de Equipos, Personas y Proyectos
- Inicio de sesión mediante Google

***Release* 2:**

Duración: 28/03/22 al 19/06/22

Sprints: 10 al 15

Funcionalidades principales:

- Posiciones vacantes
- Categorías customizables (creación y asignación a entidades)

- Filtros en la pantalla principal (generales y por categorías)
- Buscador
- *Zoom in & out* en la pantalla principal

***Release* 3:**

Duración: 20/06/22 al 31/07/22

Sprints: 16 al 18

Funcionalidades principales:

- Lectura, creación, eliminación y actualización de usuarios
- Sistema de roles (permisos y vistas para los diferentes usuarios)
- Escenarios interactivos
- Históricos (línea del tiempo en la empresa) de Personas y Equipos
- Reportes exportables

Luego de terminado y liberado el *Release* 3, se obtuvo *feedback* con algunos arreglos, en su mayoría visuales, por lo que el equipo decidió tomar estas devoluciones y realizar estos y otros arreglos pendientes que aportarían un poco más de valor a la solución. Esto llevó un *sprint* más, quedando liberada la **versión** 3.0.1 del sistema el 15/08/2022.

7.3 Feedback y su importancia en el proyecto

El *feedback* es un pilar fundamental dentro de la cultura de Qualabs, ya que es lo que utilizan para impulsar a las personas y promover la mejora continua. Debido a que está tan arraigado dentro del cliente, también lo aplicaron en el proyecto, siendo este una constante a lo largo del transcurso del mismo. Si bien Scrum [59] establece una instancia de *review*, para obtener *feedback* y alienta la comunicación con el cliente en su definición, el punto recién mencionado potenció el poder del mismo, estando este presente en todo momento y teniendo un peso importante en el desarrollo (provocando grandes cambios en el sistema a lo largo del tiempo). Se centró principalmente sobre el sistema, siendo este sobre correcciones en el funcionamiento (para favorecer flujos de negocio), arreglos visuales (tanto de distribución de elementos como de nomenclatura), pero también en algunos casos se recibió *feedback* sobre el proceso, promoviendo mejoras para implementar en las reuniones, posturas a tomar frente a negociaciones entre otras.

En su mayoría, el *feedback* fue obtenido en las reuniones, mayoritariamente las de *review*, en dónde el cliente, al ver las funcionalidades desarrolladas, efectuaba comentarios sobre lo que le parecía, realizaba preguntas y proponía mejoras. Luego, el equipo destinaba unos minutos para promover más *feedback*, intentando llevar al cliente a pensar en los casos de uso, con el fin de poder evaluar si la funcionalidad se adaptaría correctamente a lo que necesitaban. Este *feedback* recibido era en parte, proposición de cambios (tanto de mejoras como de adición o eliminación de elementos), comentarios de felicidad y de buen logro del equipo y en algunos casos, ideas o propuestas sobre nuevas funcionalidades (dando lugar a negociaciones). En caso de que diera el tiempo y hubiera algo para consultar, se destinaban los minutos finales de la *review* para consultar sobre funcionalidades que iban a desarrollarse o se estaba iniciando su desarrollo (mostrando en algunos casos prototipos), para poder recibir comentarios tempranos sobre las mismas, negociar o establecer siguientes reuniones. El constante *feedback* recibido fue de gran valor para el equipo, ya que le brindó más seguridad en el desarrollo, gracias a las devoluciones de la satisfacción del cliente. Además, le permitía estar alineado con este, e ir adaptando el sistema a sus necesidades, realizando pequeños cambios en el tiempo, evitando grandes cambios o retrabajos luego.

Una vez que el sistema estuvo al alcance del cliente, este comenzó a utilizarlo y gracias a la confianza que tenía con el equipo, varios usuarios fueron brindando *feedback* y proponiendo ajustes. Estos utilizaban el sistema e iban generando una lista de mejoras, dudas e ideas de incorporaciones que luego enviaban al equipo, también, en algunos casos, se realizaron encuentros rápidos *online* para realizar pequeños intercambios. Incluso, Nicolás Levy (Director de Ingeniería) se grabó usando el sistema, mientras navegaba sobre el mismo e iba emitiendo comentarios de sus sensaciones, lo que le gustaba, lo que no, cambios que haría. Si bien se trata de un tipo de usuario distinto al que está enfocado el sistema (dado que es del área de Ingeniería), el poder verlo usando el sistema mientras comentaba sus pensamientos, fue de gran ayuda, y brindó muchos *insights* e ideas al equipo.

Como uno de los objetivos del proyecto, era generar un producto de calidad, que se adapte a la cultura del cliente y cumpla con sus expectativas, se decidió crear un proceso para gestionar los cambios, así poder dedicarles tiempo, relevarlos y descubrir las verdaderas necesidades detrás de las peticiones. Permitiendo luego ir incorporándolos al sistema, haciendo que fuera adaptándose tempranamente a las necesidades del cliente, este pudiera seguir usándolo, y no se acumularan cambios para más tarde.

7.3.1 Gestión del *feedback* recibido

Como se mencionó, se tenían dos grandes vías para recibir *feedback*, en las *reviews* y "*offline*" (de manera asíncrona). En cuanto a los *feedbacks* de las demos (*reviews*), durante esta reunión, los integrantes del equipo que no estaban presentando, tomaban notas de los comentarios que el cliente iba realizando y sus *feedbacks*. Al recibir los comentarios, si el equipo tenía dudas, realizaba preguntas y en algunos casos se negociaba sobre lo propuesto, con el fin de que las notas quedaran lo más completas y sin incertidumbre. Estás notas se pasaban a un archivo dentro de la carpeta "Demos", en el directorio de Google Drive, la que almacenaba toda la información de estas instancias. Este archivo se dividía internamente en dos, definiendo una sección de *feedbacks* y cambios solicitados, y otra de comentarios (por ejemplo, exclamaciones de alegría o similar), de manera que fuera fácil de diferenciarlos luego. Para los *feedbacks* recibidos de manera asincrónica (recibidos en forma escrita u otro formato, sobre el uso del sistema), se creaba un archivo en la carpeta de "*Feedbacks" en* Google Drive, teniendo como título el nombre de la persona que había enviado el *feedback*. Luego, se pasaban los datos a este archivo, analizando su contenido, y en caso de que se generaran dudas o algún punto no estuviera claro, se consultaba con el autor, ya sea por mensaje o agendando una reunión, para aclarar estos asuntos.

En ambos casos, el mecanismo para procesar los *feedbacks* era similar, primero se analizaban las anotaciones cuidadosamente, discutiendo los posibles motivos para la solicitud, tamaño, dificultad del cambio, e implicancia sobre el resto del código. Luego, se separaban las solicitudes en base al tamaño o dificultad del cambio, y posteriormente se los priorizaba, viendo para cada una si existían solicitudes similares de otros integrantes del cliente o en otro caso se ponderaba en base a quien había realizado la petición.

En caso de que el cambio fuese grande, faltara clarificación, o se tratara de un cambio que el equipo consideraba que podría llegar a ser interesante/aportar valor, pero solo había una solicitud, se pedía una reunión con el cliente para clarificar los puntos, priorizar y discutir detalles sobre la posible implementación de estos cambios. En algunos casos, en donde fuera pertinente, también se realizaban prototipos para luego presentarlos al cliente y validar los cambios, ya que algunos de estos terminaron siendo nuevas funcionalidades o pantallas adicionales. Una vez aceptado, validado y priorizado el cambio se procedía a incorporarlo en el *backlog* de JIRA para tenerlo presente y luego poder ser implementado.

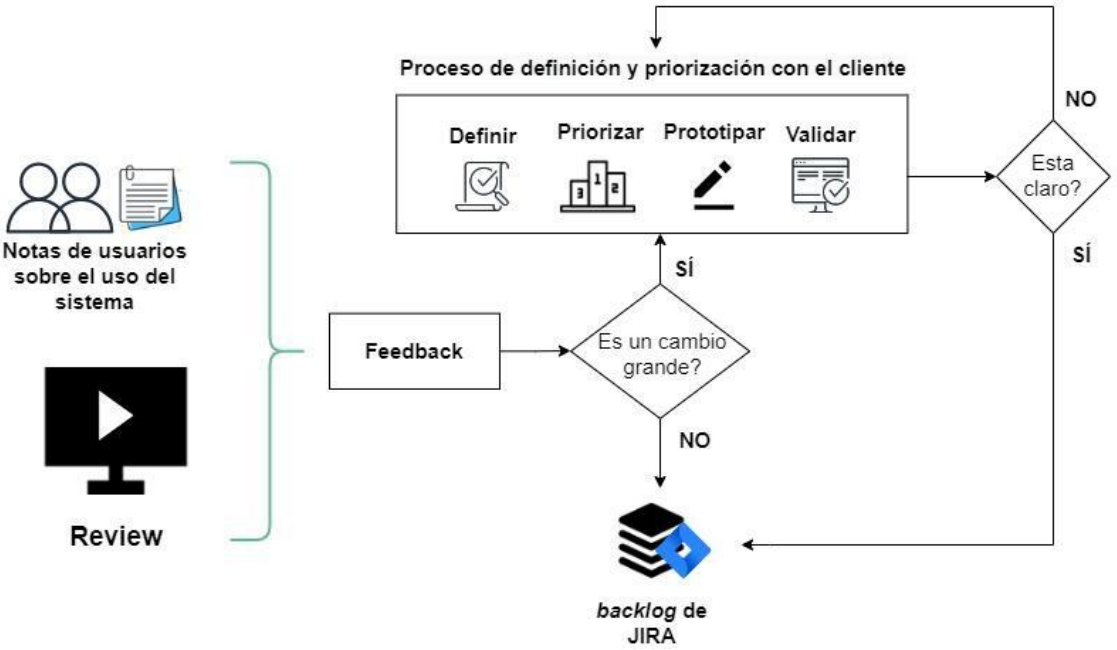


Figura 7.2: Esquema del proceso de gestión de *feedback*

Si era un cambio chico, fácil de hacer y aportaba valor, se creaba el *ticket* en el *backlog* de JIRA, bajo la épica de *feedback* para luego referenciarlo fácilmente, y se lo tenía presente para ingresarlo en el próximo *sprint*, o incluso adicionarlo al actual en caso de que las tareas se terminaran y no hubiera algo más prioritario. Generalmente, los cambios provenientes de la *review* solían ser pequeños y fáciles de realizar, por tanto, el equipo intentaba resolverlos en el *sprint* siguiente de manera de poder mostrarle el cambio al cliente en la próxima instancia, para que se sintiera escuchado y que el sistema iba avanzando y evolucionando según sus peticiones. Esto fue muy positivo para el proceso y satisfacción del cliente.

El hecho de contar con cambios de *feedback* pendientes en el *backlog* fue de gran utilidad, ya que permitieron agregar tareas al *sprint* en casos en donde no se conseguía comunicación con el cliente y se necesitaban validar o priorizar historias, o el equipo se encontraba atascado debido a incertidumbres y desconocimiento. De esta manera, se lograba tener una mayor cantidad de tareas en el *sprint*, y además, se aprovechaba a actualizar el sistema con los cambios solicitados, aportando valor al cliente y cumpliendo con el RNF 4.

7.4 Principales hitos del proyecto

El proyecto tuvo diferentes etapas y acontecimientos a lo largo del tiempo, a continuación se narrarán algunos de los principales sucesos que ocurrieron (tanto positivos como negativos).

Intercambios de implementación con Ingeniería

Durante el transcurso del proyecto hubo algunas ocasiones en las que el equipo se encontró con dudas respecto a la implementación, tanto de cambios solicitados (que no eran tan fáciles) como ideas propias sobre funcionalidades de las que querían tener una segunda opinión. Por este motivo se tuvieron, en varias instancias, reuniones con Nicolás Levy (área de Ingeniería) para tratar estos asuntos. Previo a estas reuniones, se realizaba un archivo con una descripción de la problemática a tratar, y otra información relevante, con el fin de que Nicolás estuviera al tanto y pudiera prepararse para la reunión. Esto permitió aprovechar mucho más el tiempo (que era acotado por su agitada agenda) y dar lugar a mejores discusiones e ideas.

Se tomó la decisión de llevar estas instancias a cabo, en parte porque el sistema iba a ser mantenido por el cliente, y por tanto el equipo quería que este fuera parte de las decisiones importantes. Por otra parte, cuentan con años de experiencia en desarrollo y por lo tanto podrían brindar otra mirada o consejos en base a la experiencia.

Un punto a destacar es que Nicolás además de ser el líder de Ingeniería de Qualabs, forma parte de los procesos de operaciones de la empresa, por lo tanto, sus ideas y opiniones no estaban solamente basadas en implementación, sino que las acompañaba con casos de uso o ejemplos de cómo podría llegar a ser usado el sistema. Esto le permitió al equipo tener un mejor panorama y contexto para decidir sobre la implementación.

Eliminación de una funcionalidad principal

Una de las grandes funcionalidades planeadas era el “Hipotetizador manual”. La misma fue un tanto polémica ya que se tuvo varias idas y vueltas sobre lo que debería hacer. En una primera instancia, se quería que permitiera realizar diferentes diagramaciones de empleados y equipos, para poder visualizar posibles cambios en la organización y así tomar mejores decisiones de negocio. Al momento de empezar a relevar y refinar esta funcionalidad, esta idea se descartó, y se quería un proceso más complejo, en el que se pudieran combinar equipos y personas

existentes, con proyectos, equipos y personas inexistentes, potenciales ingresos, que deberían poder crearse "*on the go*" para la hipotetización, pero no quedar registrados en el sistema.

Si bien en el plan inicial esta funcionalidad estaba prevista para el segundo *release*, dado al alto grado de incertidumbre que se tenía, y el gran proceso de refinamiento que se debía realizar, se decidió moverla para el tercer *release*, ya que no se iba a poder realizar a tiempo. Aquí fue (en el *sprint* 13), la primera instancia en que se tomaron historias de *feedback* que se tenían pendientes en el *backlog* para incorporar al *release*, con el fin de poder refinar y definir la funcionalidad del hipotetizador y en paralelo seguir desarrollando.

Para refinar y definir esta funcionalidad, se llevó a cabo el proceso de elicitación y validación de requerimientos que venía utilizando el equipo[6]. Como primer paso, se tomó como base los comentarios e ideas que habían surgido de la reunión con el cliente en la que se discutió sobre la segunda aproximación, se analizaron los comentarios y se ideó un conjunto de funcionalidades. Este proceso se llevó a cabo de forma individual, con el fin de promover una mayor cantidad de posibilidades y no sesgar ideas en una primera instancia, y luego se realizó una puesta en común con el equipo para llegar a la versión definitiva.

Una vez obtenida esta versión, se adentró en el proceso de *Design Sprint* (definido en el capítulo de "Marco de trabajo") en el que se diseñaron prototipos para estas funcionalidades. Posteriormente se validó con el cliente, y luego de varias revisiones, reajustes de diseño, y negociaciones, se logró llegar a la versión "final" a ser implementada (siendo este el proceso más largo de *Design Sprint* que enfrentó el equipo). Si bien se tenían estos prototipos y definiciones y una idea de lo que haría, se vio la necesidad de obtener casos de uso para poder realizar correctamente la funcionalidad (ya que la misma era bastante compleja). Debido a esto, se siguió en los *sprints* dando mayor énfasis al resto de funcionalidades planeadas y cambios de *feedback*, para evitar poner foco en el desarrollo de esta funcionalidad hasta que el equipo estuviera completamente seguro de la misma.

Finalizando la *review* del *sprint* 15 (habiendo estado casi 3 *sprints* con el refinamiento del hipotetizador), el equipo mencionó el plan de realizar entrevistas a diferentes integrantes en el *sprint* entrante para obtener casos de uso para la funcionalidad del hipotetizador. En este momento, Sofía (que en los *sprints* anteriores no había podido estar presente) dice que esto no

[6] Para más detalles de este proceso, ver capítulo de "Ingeniería de requerimientos".

iba a ser posible, dado que el proceso de armado de equipos estaba en plena evolución y cambio, y por lo tanto no tenían una manera de hacer explícito una serie de pasos, de forma tal que el equipo pudiera implementar una funcionalidad, ya que mismo ellos no la tenían definida. Al mencionar esto, el resto del equipo del cliente concuerda y mencionan que, aunque se llegara a una definición en la cual se pudiera establecer una serie de pasos para poder esbozar un flujo, este iba a cambiar en un futuro no tan lejano, haciendo que la funcionalidad quedara obsoleta. Por este motivo, deciden eliminar la funcionalidad.

Redefinición de plan de *release* y manejo de alcance

La ocurrencia del suceso anterior tomó un poco por sorpresa al equipo, ya que se había incurrido casi 3 *sprints* definiendo distintos aspectos de lo que sería esta funcionalidad, y luego terminó siendo eliminada. Sin embargo, el equipo decidió no desalentarse por este hecho, y tomarlo como aprendizaje. Gracias al pienso y proceso de refinamiento y definición llevado a cabo para la funcionalidad, el cliente pudo darse cuenta de que su proceso no estaba del todo maduro para ser sistematizado. De no haber realizado este proceso, se hubiera implementado una funcionalidad que iba a quedar obsoleta al poco tiempo o no iba a ser utilizada.

Igualmente, esto afectó el alcance. Para remediarlo, en seguida el equipo se puso en marcha con un proceso fuerte de ideación, elicitación y definición de nuevos requerimientos para poder lograr un alcance adecuado para el proyecto de facultad, pero que también tuviera valor para el cliente. Este proceso se siguió con el mismo procedimiento que para relevar los requerimientos al inicio del proyecto[7], y finalmente se logró definir un conjunto de funcionalidades para el *release* 3, obteniendo una nueva versión del plan de *release*. Este proceso de generación de nuevos requerimientos tuvo sus idas y vueltas, re-priorizaciones, e incluso validaciones con ingeniería. Sin embargo, gracias a que ya se tenían definidas algunas funcionalidades para el tercer *release*, que no fueron eliminadas, y también se tenían algunos arreglos de *feedback* en el *backlog*, parte del equipo se centró en trabajar en estas y avanzar con el desarrollo, mientras la otra, si bien seguía desarrollando, tenía mayor foco en el proceso de requerimientos.

[7] Este proceso puede verse en más detalle en la sección de “Proceso” del capítulo de “Ingeniería de requerimientos”.

Feedback y solicitudes de cambio

Mencionado en las secciones anteriores, el *feedback* tuvo un rol muy importante en el proyecto, hizo que el sistema fuera evolucionando a lo largo del tiempo, añadiendo, eliminando y modificando diferentes funcionalidades y cambios (puede verse parte de la evolución en el anexo “Evolución del Sistema”). De los cambios provenientes del *feedback* obtenido en diferentes momentos en el tiempo, se registraron en el tablero de JIRA un total de 39 *tickets*. De los cuales, 34 fueron realizados, siendo 23 desarrollados en el lapso del segundo *release* y 11 en el tercero. Si bien algunas de estas solicitudes fueron cambios menores, otras terminaron generando nuevas funcionalidades que se agregaron a los *releases.* En el segundo *release* se crearon 11 funcionalidades nuevas y en el tercer *release* 3. Cabe destacar que estos *tickets* que se mencionaron, son parte del *feedback* que se registró (siendo en su mayoría del obtenido de manera asincrónica), sin embargo, en algunas ocasiones el *feedback* fue recibido mientras se desarrollaban las historias, y por tanto, se iban realizando los cambio directamente en ellas sin crear un *ticket* específico.

7.5 Gestión del esfuerzo

Para registrar el esfuerzo en el proyecto se utilizó la herramienta Toggl. En esta, cada integrante registraba, con un título descriptivo, el esfuerzo invertido en las diferentes tareas, marcando el área del proyecto en la que trabajó. Se crearon proyectos en Toggl (que son identificadores con nombres y colores diferentes, que se le agregan a los registros) para referenciar a las distintas áreas de trabajo del proyecto, con el fin de que estos pudieran diferenciarse fácilmente y también agruparse por áreas. De esta manera, se pudo llevar un control del esfuerzo que se estaba dedicando en total al proyecto, y sobre cada área, por *sprint*, pudiendo fácilmente visualizar la diagramación de los mismos por proyecto o integrante, permitiendo sacar conclusiones, detectar problemas y proponer mejoras.

A continuación se mostrará la distribución del esfuerzo, referente a los registros realizados en los 19 *sprints* de desarrollo, por áreas, que comprende un total de 2085 horas registradas. Cabe destacar que previo a este período se tuvo una etapa inicial de descubrimiento y definición del proyecto, y primeros encuentros para conocer al cliente, en la cual no se realizó un registro de horas. Y luego, una etapa posterior a estos *sprints* en la que se dedicó principalmente a la construcción de la documentación académica, asuntos legales y otros entregables para el cliente.

Estas dos etapas llevaron su tiempo, pero no serán contempladas en el siguiente estudio. El período de los 19 *sprints* tomó del 22 de noviembre de 2021 al 15 de agosto de 2022.

Aclaración: Lograr que el equipo se adaptara a registrar frecuentemente el esfuerzo llevó un tiempo, debido a que era una actividad que no estaban acostumbrados a hacerla. Esto hizo que al inicio y hasta gran tiempo después el equipo no fuera consistente con los registros, generando que muchas horas quedaran sin registrarse dificultando el buen seguimiento del mismo.

Los 5 proyectos creados en Toggl, para identificar las áreas del proyecto, son los siguientes:

- **Codificación**: Destinado a todas las tareas relacionadas con el desarrollo de las aplicaciones de *frontend* y *backend*, incluyendo configuraciones de *pipelines* y creación de ambientes, junto también en algunas ocasiones, con investigaciones menores que se necesitaran para el desarrollo.
- **Gestión**: Engloba todo lo relativo a tareas de diseños de interfaz, creación de prototipos, diseños de arquitectura, relevamiento, elicitación o refinamientos del plan o historias, entre otras diversas actividades de gestión del proyecto, como creación de archivos (tanto de acontecimientos o seguimiento de *sprints*, metodologías o procesos a seguir, entre otros), armado de archivos para facultad (presentación de revisión, informe de avance y de revisiones, entre otros).
- **QA**: Esfuerzos referentes a las tareas del proceso de chequeo y aseguramiento de las historias, creación de *tests* automáticos, y ejecución de otras tareas de aseguramiento. Es importante mencionar que algunas actividades de aseguramiento de la calidad, sobre todo la creación de archivos para luego ejecutar las tareas, quedaron registradas dentro de “Gestión”.
- **Investigación**: Contiene todo lo relacionado a investigaciones (grandes) planeadas con anterioridad y necesarias para el proyecto, que generalmente terminaban en la generación de un archivo o prueba de concepto. También ocurrieron algunas investigaciones de menor tamaño, necesarias en el desarrollo, que quedaron bajo el proyecto de “Codificación”, ya que no necesitaban de pruebas de concepto y el propio autor de la tarea podía resolver la problemática por sí mismo mientras desarrollaba.
- **Reuniones**: Estaba pensada principalmente para marcar el esfuerzo de las ceremonias de Scrum (todas, a excepción de la *Backlog-grooming*), reuniones con el tutor y reuniones entre el equipo o con el cliente en las que se trataban diversos temas.

Si bien la idea de este desglose era buena, en algunas ocasiones, se realizaban tareas de estas áreas en reuniones, ocasionando que las horas quedaran catalogadas allí, en vez de en su correspondiente sección.

En la figura 7.3 se observa la distribución de esfuerzo sobre los diferentes proyectos definidos.

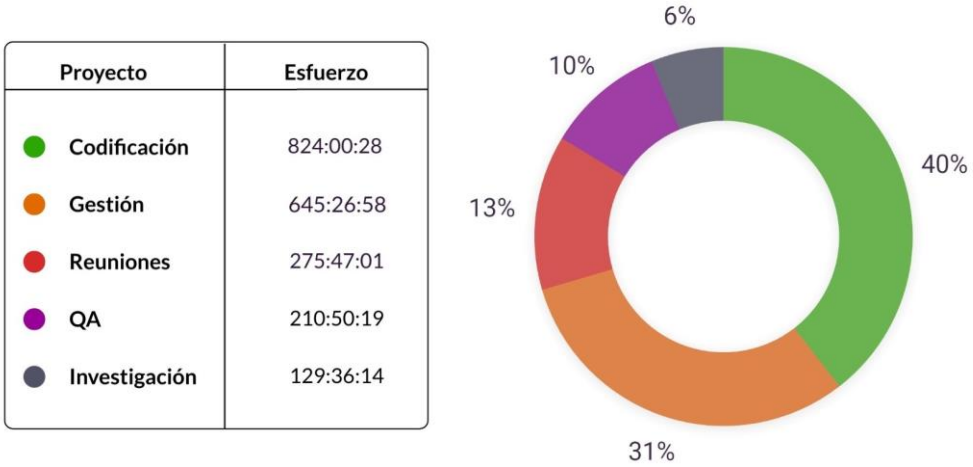


Figura 7.3: Distribución de esfuerzo entre las diferentes áreas del proyecto

Como se puede observar, el área de “Codificación” es la que tiene mayor dedicación ya que comprende las principales actividades que permitieron la creación del producto y sus ambientes. Luego, se aprecia que el área de “Gestión” tiene un peso importante, esto se debe a todo el proceso de descubrimiento y refinamientos realizado en el transcurso del proyecto, que ya se ha mencionado en los capítulos anteriores. Seguidamente se ven las áreas de “Reuniones” y “QA”, la primera englobó todas las instancias de las ceremonias de Scrum, junto con otras reuniones varias, y la segunda, parte de las actividades de aseguramiento de calidad, que igualmente se debe recordar que algunas tareas de “QA”, más q nada el armado de archivos para luego realizar las correspondientes actividades, generalmente se anotaban dentro de “Gestión”.

En conclusión, dadas las características del proyecto, con el alto grado de descubrimiento y participación del cliente, era esperable obtener un gran porcentaje de dedicación en el área de “Gestión” y “Reuniones”, ya que refleja que realmente se tomó el tiempo para estas tareas en el desarrollo del proyecto. Adaptando los procesos según las necesidades del cliente, obteniendo un buen resultado, ya que el cliente quedó muy satisfecho, y el equipo pudo lograr completar

el alcance planeado, cumpliendo con el objetivo establecido de “Aplicación del proceso de Ingeniería”.

7.6 Métricas de gestión

En el transcurso de los *sprints* se llevaron diferentes registros y anotaciones sobre los procesos y acontecimientos que ocurrían, con el fin de tener información para luego poder sacar métricas y conclusiones. Por *sprint* se registraban los objetivos, riesgos, historias planeadas y realizadas, velocidad, esfuerzo, registro de *reviews* y retrospectivas, notas de reuniones, archivos con reglas de procesos a seguir, entre otros. Todo esto permitió tener una noción de cómo iba el proyecto a lo largo del tiempo, detectar problemas y proponer mejoras.

El desarrollo del proyecto se llevó a cabo en 18 *sprints* completos, y un último *sprint* de desarrollo en el que se realizaron algunos arreglos pendientes de *feedback* y *bugs* menores que aportaban valor a la solución. Sin embargo, este *sprint* fue más leve en cuanto a su dedicación en código, ya que se estaba comenzando con la fase de documentación. Si bien luego se siguió contando el tiempo a modo de *sprint* para tener un orden y control (y si se daba lugar se resolvía algún *bug* puntual), no se tomaron en cuenta para estas métricas, ya que el principal objetivo de estos “*sprints*” posteriores fue la documentación.

Horas de desarrollo por *sprints*

En la figura 7.4 se observa el esfuerzo registrado por el equipo en el área de desarrollo en los *sprints* mencionados. En este caso, el “área de desarrollo” engloba a los proyectos anteriormente definidos de “Codificación” y “QA”, ya que este último era parte esencial para lograr la completitud de las historias.

Como ya se mencionó, el proyecto tenía un alto grado de descubrimiento, el cual tomaba su tiempo de dedicación. Con el fin de intentar cumplir satisfactoriamente con el plan del proyecto y no perder el foco en el desarrollo, se estableció el objetivo de “Asegurar la entrega continua al cliente”. En este, los integrantes se comprometían a dedicar al menos unas 60 horas de desarrollo (Código + QA) por *sprint*, para asegurar un compromiso base al área de desarrollo, con el fin de no dejar de dar valor al cliente. Se llegó a este número en base a la sugerencia de la universidad de un esfuerzo de 120 horas por *sprint* (en equipos de cuatro integrantes) y que

se consideró razonable apuntar, por defecto, a un esfuerzo equitativo en ambas áreas ("desarrollo" y "descubrimiento").

Claro está que, por las características del proyecto, iban a haber momentos en los que el esfuerzo iba a estar por debajo y otros por encima de este número, pero se quiso poner un valor como base por dos principales motivos. Primero, para asegurar la entrega continua, ya que como la otra área no tenía estimación (dado a su naturaleza), de no limitarse, fácilmente podía pasarse más tiempo del necesario, dejando de lado el desarrollo. Segundo, para poder controlar y refinar el rendimiento del equipo, ya que con el registro de horas y las estimaciones de las historias se podía tener una noción del rendimiento del equipo, y sacar conclusiones o proponer mejoras en base a los números que se iban obteniendo.

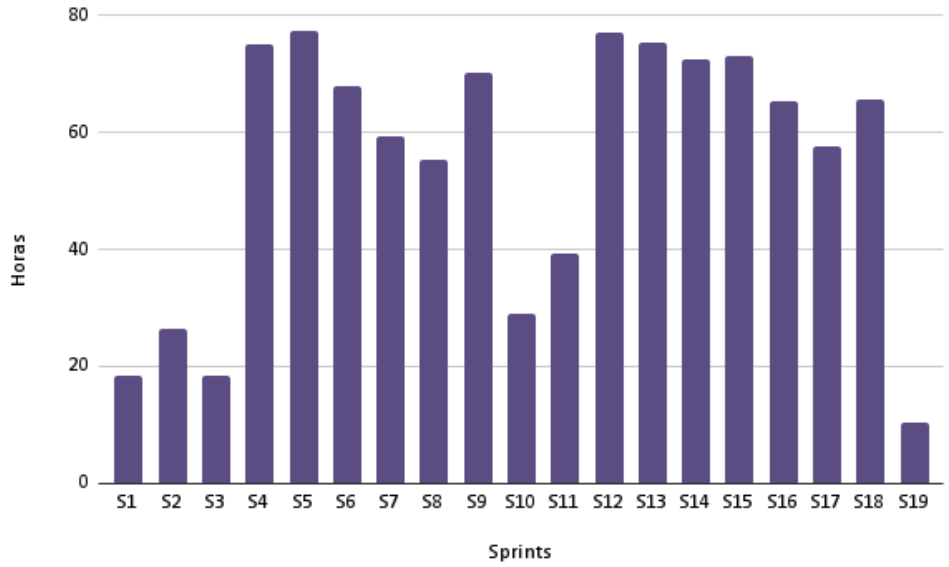


Figura 7.4: Distribución de esfuerzo del área de desarrollo por *sprints*

En el gráfico se puede apreciar que el objetivo pudo ser cumplido en casi todos los *sprints*, e incluso superarlo en varios de ellos. En los tres primeros *sprints* el equipo se estaba amoldando, encontrando el ritmo de trabajo, y, todavía con un importante grado de descubrimiento (entendiendo más al cliente, el contexto y realizando diseños). Además, todos los integrantes se encontraban terminando el semestre académico, lo cual afectaba el rendimiento esperado, y luego se encontró el período de las fiestas de fin de año. Un factor no menor, es que el equipo tuvo grandes dificultades para hacer del registro de esfuerzo un hábito, sobre todo al inicio del proyecto, generando que parte del esfuerzo quedara sin registrar y por tanto no pueda ser reflejado en el gráfico. En cuanto a los *sprints* 10 y 11, estos se encontraron afectados, en gran

parte, por las tareas académicas referentes al proyecto (“informe de avance” y “segunda revisión”) que llevaron su tiempo de creación y refinamientos. Por último, el *sprint* 19, como ya se mencionó, tenía un mayor foco en la documentación académica, el desarrollo solo se utilizó para realizar algunos arreglos sobre el sistema, por lo tanto, era esperable que no se alcanzara el objetivo.

El hecho de llevar esta métrica permitió al equipo ver rápidamente si se estaba dedicando el esfuerzo esperado y detectar problemas, en caso de no lograrlo. Además, el tener el objetivo planteado, estableciendo un mínimo a alcanzar, hizo que el equipo tuviera siempre una meta constante a la cuál debía llegar, impulsando a intentar proponerse a más, y mejorar en cada iteración para lograr buenos resultados.

Story points planeados y realizados

Al inicio de cada *sprint* se estimaban, en *story points* [60], las posibles historias a realizar, y se llegaba a un número el cuál el equipo se comprometía a completar en ese lapso, y se registraba. Al finalizar el *sprint*, se comparaban los puntos realizados con los planeados para sacar conclusiones. Esta métrica sirvió para detectar posibles errores de estimación, y tener una noción de cuánto estaba rindiendo el equipo, y poder evaluar de este modo, en base al plan del proyecto en su totalidad, si era necesario destinar más esfuerzo para lograr completar lo planeado. El tener esta métrica también fue de utilidad para ir teniendo una noción del estado del avance del proyecto, lo que fue de utilidad para negociar con el cliente, y además, poder tener la posibilidad de evaluar la factibilidad de variar la cantidad de esfuerzo, en base a la etapa del proyecto, sin generar grandes consecuencias.

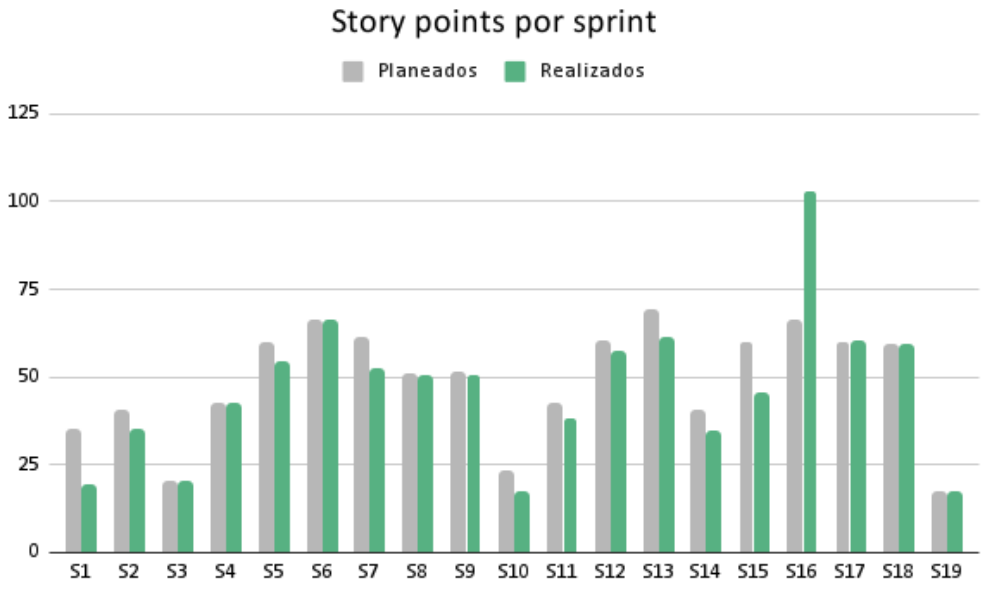


Figura 7.5: *Story points* planeados frente a realizados en los *sprints*

Se puede observar que, en general no se presentan grandes diferencias entre los valores de lo realizado y lo planeado. Respecto a las variaciones de los valores planeados entre *sprints*, algunas son debidas a las características del proyecto, teniendo tiempos en los que se requería mayor foco en descubrimiento que en desarrollo, o viceversa, y riesgos contemplados en los que se bajaba la intensidad, ya que se sabía que iba afectar al rendimiento.

En los primeros dos *sprints* se puede ver que hay una diferencia, entre lo planeado y lo realizado, esto se debe a que el equipo estaba recién empezando, y debía encontrar un ritmo de trabajo, además que tenía desconocimientos sobre las tecnologías, lo que no le daba mucha seguridad para estimar. En adición a esto, en este tiempo los integrantes se encontraban en época de parciales y entregas de materias de facultad, lo que afectó el esfuerzo dedicado.

El *sprint* 3 englobó las fechas de navidad y fin de año, por este motivo, los integrantes se tomaron un tiempo para pasar con las familias, e incluso en algunos casos viajar a visitarlos, por tanto, se decidió bajar la carga de trabajo, ya que iba a ser altamente probable que el esfuerzo dedicado fuera menor en estas fechas.

Luego, se aprecia cómo se va aumentando y estabilizando el trabajo, y mejora la relación entre lo planeado y lo realizado, dado que el equipo tenía un mejor conocimiento del proyecto y del rendimiento del equipo, y pudo ir refinando la estimación. Si bien hubo algunas diferencias entre estos dos valores, generalmente se debieron a errores de estimación (teniendo tareas que

tomaron más tiempo que el planeado, en su mayoría debido al desconocimiento), o riesgos que tuvieron un mayor impacto del previsto, que afectaron al rendimiento logrado. Los casos del *sprint* 10 y 16 serán detallados a continuación, en el estudio de la velocidad. En el último *sprint*, se nota una baja en lo planeado, esto era el comportamiento esperado, ya que, como se mencionó, se estaban haciendo arreglos menores e iniciando la época de documentación.

Registro de velocidades del equipo

Al cerrar cada *sprint* se registraban las historias realizadas y se sumaban los *story points* de cada una de ellas para obtener la velocidad del equipo en la iteración. Una vez calculada la velocidad, se comparaba con la cantidad de puntos planeados para evaluar el rendimiento del equipo y sacar conclusiones. Este valor era luego utilizado para realizar la estimación de los siguientes *sprints*, tratando de apuntar a una mejora o mantenimiento del rendimiento en el tiempo. Esta métrica le permitió al equipo tener una mejor perspectiva del resultado de las iteraciones, pudiendo detectar problemas, buscar explicaciones y proponer mejoras.

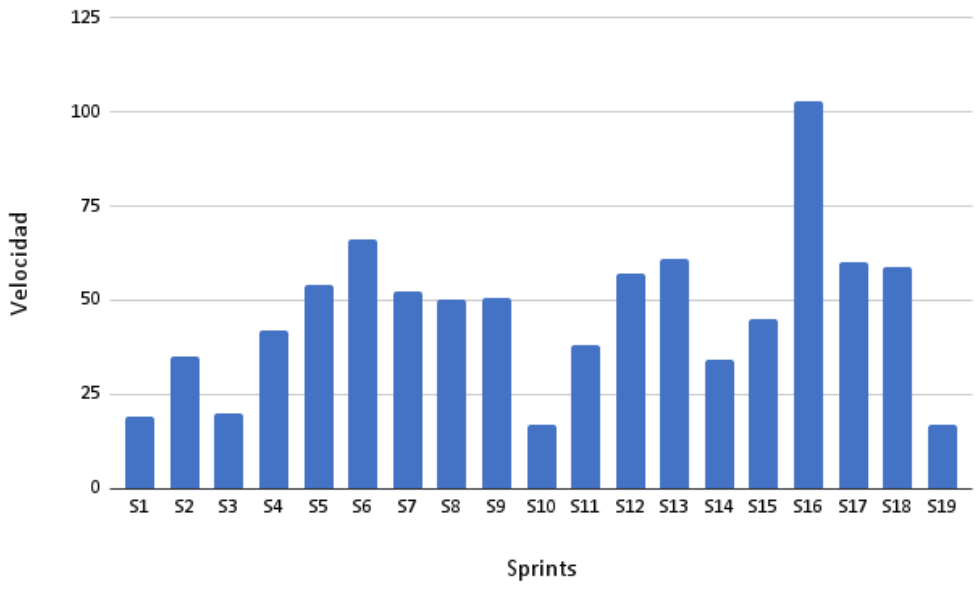


Figura 7.6: Velocidad obtenida a lo largo de los *sprints* del proyecto

Si bien la velocidad tuvo algunas variaciones, se puede observar que intenta mantenerse por el entorno de los 50 *story points*. Algunas de las variaciones se deben a las características propias del proyecto, teniendo *sprints* que fueron más dedicados a investigación que a desarrollo y viceversa. Otro factor que generó variaciones fue la ocurrencia de ciertos riesgos que afectaron

al proceso, causando repercusiones negativas y positivas en la velocidad.

En el último *sprint* se aprecia una disminución en la velocidad, esto es debido a que se estaba terminando el proceso de desarrollo y comenzando la etapa de la documentación. Si bien se realizaron unos últimos arreglos menores de código, en esta etapa el foco estaba en la documentación, y por tanto era esperable este comportamiento.

Se puede ver que el *sprint* 10 es el punto más bajo de la gráfica (omitiendo el último, que era esperado por lo antedicho), este *sprint* tomó lugar a finales de marzo hasta el 11 de abril. En este rango se tenía la entrega académica del informe de avance, el cuál es un archivo solicitado por la universidad, en el que se describe el progreso del proyecto hasta el momento (su entrega estaba inicialmente planeada para el 12 de abril, luego sobre la fecha se aplazó unos días por semana de turismo). Por lo tanto, se decidió poner foco en el armado de este documento, y disminuir un poco el desarrollo, además había un integrante cursando la enfermedad del COVID-19, lo que provocaba una prominente baja en su rendimiento. Si bien estos factores se tuvieron en consideración a la hora de planear la carga de trabajo para el *sprint* (que se puede observar en la figura 7.5), tuvieron un peso más fuerte del esperado. Además, se sumaron otros riesgos, que si bien estaban contemplados, también repercutieron en mayor medida a la baja del desarrollo. Estos riesgos pueden verse en detalle en el anexo “Riesgos por *sprint*”.

En cuanto al *sprint* 16 (pico más alto), si bien, como se mencionó en la sección de hitos del presente capítulo, este *sprint* comenzó con un reajuste de alcance, el equipo pudo actuar rápido y lograr una nueva versión de plan de *release* junto con el cliente. Igualmente, muchas historias no habían sido eliminadas en ese proceso, por lo que el equipo tuvo tareas en las que trabajar en ambas semanas. En la segunda semana del *sprint*, un integrante del equipo estaba de licencia en el trabajo, por lo que pudo dedicar más esfuerzo a las tareas de desarrollo, incluso realizando más tareas de las previstas, provocando un aumento en la velocidad.

7.7 Gestión de la comunicación

Durante el proyecto, la comunicación entre las diferentes partes involucradas fue esencial para poder llevarlo a cabo. Se debía mantener una comunicación activa entre varios actores, teniendo que organizar reuniones frecuentemente en horarios que a todos les sirviera. El equipo optó por realizar las reuniones en forma remota debido a la situación sanitaria de COVID-19. Estas

fueron realizadas remotamente durante el proyecto, excepto la realización de las pruebas de usuarios y la preparación de la presentación final, que se realizaron en forma presencial.

Para la organización de las reuniones se utilizó Google Calendar, donde se agendaba cada llamada o reunión, como también las actividades fuera del proyecto que lo podrían afectar. Los actores principales fueron: el equipo, el cliente y el tutor.

7.7.1 Comunicación interna

A lo largo del proyecto, el equipo mantuvo una comunicación constante y activa mediante varios medios de comunicación. Como se ha mencionado, el equipo realizó *dailies* mediante llamadas por Google Meet. Las otras ceremonias de Scrum también se realizaron mediante esta herramienta. Además, el equipo se mantuvo actualizado diariamente a través de la aplicación de Whatsapp. Se abrieron 3 grupos de Whatsapp; uno para la comunicación general del equipo, otra para todo lo que corresponde al desarrollo *Backend*, y el último para todo lo relacionado al desarrollo *Frontend*.

Además de las reuniones ya mencionadas, se coordinaron varias reuniones que surgieron durante el proyecto. Estas incluyeron reuniones de diseño, prácticas para las revisiones, reuniones informativas generales, programación en pares, entre otras.

El equipo también utilizó la aplicación Discord. En esta, se abrieron canales para poder mandar información de diferentes tópicos, que queden guardados y fáciles de acceder. Algunos de estos canales fueron: *to-do, bugs-conocidos, aws-links-ambientes, daily-notes* y muchos otros más. Esto fue de mucha ayuda para mantener ordenada la información del equipo para cada tema. En Discord también se realizó la programación de pares en varias instancias utilizando los canales de voz.

7.7.2 Comunicación con el cliente

Las reuniones con el cliente fueron realizadas mediante Google Meet. Estas fueron agendadas para los viernes cada dos semanas. Como se ha mencionado, estas reuniones fueron de gran importancia ya que el cliente tuvo la oportunidad de ver los avances del proyecto y brindar *feedback*.

Otra forma de comunicación fue en la oficina. El hecho de que Patrick trabajara en la empresa desde el inicio del proyecto permitió realizar algunas conversaciones de pasillo, que permitieron evacuar algunas dudas rápidas, recordar sobre reuniones o eventos, entre otros.

Para las reuniones, el equipo tuvo que agendarse en un calendario que tenía el cliente, con el detalle de los horarios disponibles de cada miembro. Al principio, asistieron a las reuniones Lucia Lizarazu y Sofía Labadie del departamento de P&C, el CEO, Juan Pablo Saibene, y el encargado del área de ingeniería, Nicolás Levy. Más adelante, a lo largo del proyecto, la empresa fue creciendo y más personas empezaron a interesarse en asistir a las reuniones.

Debido a esto, no fue simple conseguir que todos pudieran reunirse a la misma vez, considerando además de que el cliente tenía una agenda muy ocupada. Esto implicaba tener que a veces mover las reuniones o cancelarlas a último momento, o realizarlas con algunos de los participantes y no todos.

Por último, el equipo contó con un grupo de Whatsapp con el cliente, en donde se avisaba sobre reuniones y se despejaban dudas. También, en el entorno de trabajo, se abrió un *channel* de Slack en donde Patrick, y luego Valentina, pudieron comunicarse con el cliente durante el horario laboral, si fuera necesario.

7.7.3 Comunicación con el tutor

Con el tutor, se optó por utilizar también Google Meet ya que fue la herramienta familiar y cómoda para el equipo. Al principio del proyecto, las reuniones se realizaron una vez cada dos semanas. Luego, hacia el final del proyecto, se decidió hacer estas reuniones una vez por semana para poder mantener una comunicación más frecuente con el tutor.

En estas reuniones, el equipo actualizaba al tutor acerca del estado del proyecto, se realizaban dudas y preguntas, y se recibían consejos y ayuda.

Además de esto, se creó un grupo de Whatsapp con el tutor que permitió despejar dudas de forma rápida y ágil.

7.8 Gestión de riesgos

Previo a la gestión, se llegó a una definición en común de riesgo a utilizar en el proyecto; según el PMBOK, un riesgo es un “evento o condición incierta que, de producirse, tiene un efecto

positivo o negativo en uno o más de los objetivos del proyecto…" [61]. Luego, se definió el proceso a llevar a cabo, el cual consiste en identificarlos, analizarlos, crear un plan de respuesta y registrarlos, para poder irlos evaluando en el tiempo.

7.8.1 Identificación de los riesgos

Antes de empezar el proyecto, se realizó un análisis de los riesgos que podrían afectar al proyecto en su completitud, tomando en cuenta características del mismo, los conocimientos de los integrantes, entre otros. Este análisis fue realizado mediante la técnica de *brainstorming* individual (para promover una mayor posibilidad de opciones). Como resultado, se efectuaron propuestas de cada integrante, que posteriormente fueron evaluadas y filtradas por todo el equipo, obteniendo el conjunto final.

Luego, antes de comenzar cada iteración, el equipo se tomaba unos minutos para analizar los posibles riesgos que podrían ocurrir en ese período, tomando en cuenta los diferentes aspectos del proyecto y los eventos o compromisos de cada uno. A los riesgos identificados, se les estudiaba la probabilidad de ocurrencia e impacto para generar un valor de magnitud que permitía categorizarlos en niveles (alto, medio, bajo) establecidos por el equipo. En base a estos niveles se definía la respuesta a tomar en caso de que se materializaran y se registraban, o en el caso de tratarse de riesgos ya conocidos, se actualizaban sus valores (según correspondiera). Para poder llevar un mejor control y facilitar el análisis de los riesgos estos fueron organizados en siete familias:

- **Técnico:** relacionado con las tecnologías, uso de *frameworks* o librerías, codificación, ambientes de *hosting* o repositorios, entre otros.
- **Salud**: asociado a enfermedades, covid, o malestar en general (estrés, problemas personales, entre otros).
- **Social:** referente a compromisos, vacaciones, visitas de familiares, ambiente interno del equipo.
- **Comunicación**: asociado a comunicaciones, entendimiento de las solicitudes, procesos del cliente, y coordinar reuniones (tanto con el cliente o con el equipo).
- **Gestión:** relacionado con la gestión del proyecto en sí, aplicación de metodologías, alcance, planificación, entre otros.

- **Académico:** asociado a eventos relacionados con las materias de facultad, entregables y eventos requeridos por la universidad referentes al proyecto, y eventos de cursos por fuera de la facultad.
- **Externo:** referente a elementos fuera del control del equipo, como el ambiente, sistemas externos, roturas de herramientas de trabajo (monitores, computadores).

7.8.2 Análisis

Al analizar los riesgos se define la probabilidad de ocurrencia e impacto que podrían tener sobre el proyecto, tomando los diferentes factores del mismo. Luego, con estos dos valores se establece, mediante la matriz, la magnitud y nivel.

7.8.2.1 Escala de probabilidad de ocurrencia

Cada riesgo tiene asignado una probabilidad, que indica qué tan probable es que suceda el riesgo, se tomó como referencia esta escala:

0.1	Improbable
0.3	Poco probable
0.5	Probable
0.7	Bastante probable
0.9	Muy probable

Tabla 7.1: Escala de probabilidad de ocurrencia

7.8.2.2 Escala de impacto

La siguiente escala refleja el grado de efecto (positivo o negativo) que tiene el riesgo sobre los principales objetivos del proyecto.

0	Ninguno
1	Muy bajo
2	Bajo (puede generar retrasos poco significativos)
3	Moderado (puede retrasar o adelantar el proyecto)
4	Alto (puede hacer detener el proyecto)
5	Muy alto (fracaso del proyecto)

Tabla 7.2: Escala de impacto

7.8.2.3 Matriz de probabilidad e impacto (Magnitud)

Con base en las combinaciones de escalas de probabilidad e impacto, se construye una matriz para asignar la magnitud de los riesgos, dividiéndose en tres zonas:

- Riesgo bajo (valores entre 0 y 1) (zona verde)
- Riesgo medio (valores entre 1 y 2,6) (zona amarilla)
- Riesgo alto (valores mayores a 2,6) (zona roja)

Marcador de magnitud para riesgos (P x I)						
Impacto / Probabilidad	Ninguno 0	Muy bajo 1	Bajo 2	Moderado 3	Alto 4	Muy alto 5
Muy probable (0,9)	0	0,9	1,8	2,7	3,6	4,5
Bastante Probable (0,7)	0	0,7	1,4	2,1	2,8	3,5
Probable (0,5)	0	0,5	1	1,5	2	2,5
Poco probable (0,3)	0	0,3	0,6	0,9	1,2	1,5
Improbable (0,1)	0	0,1	0,2	0,3	0,4	0,5

Figura 7.7: Matriz de probabilidad por impacto (magnitud) de un riesgo

7.8.3 Respuesta

Para cada riesgo identificado, se determinan las acciones a tomar en base a su nivel de magnitud, con el fin de que todos sean conscientes de cómo actuar frente a estos. La respuesta contiene la descripción de lo que se realizará, dependiendo del caso, teniendo como objetivo intentar reducir (lo máximo posible) su magnitud y evitar su materialización. Dependiendo de la naturaleza, en algunos casos la respuesta podría ser ejecutada antes y en otros casos luego de la ocurrencia del riesgo.

7.8.4 Seguimiento de los riesgos

Este proceso permitió llevar a cabo un seguimiento de los diferentes riesgos que estuvieron presentes a lo largo del proyecto (se hayan materializado o no), teniendo para cada uno su ID, tipo, pequeña descripción, categorización y respuesta; siendo este actualizado, según correspondiera, en cada iteración. En el anexo de **“**Registro de riesgos**”** se encuentran las tablas del registro de los riesgos contemplados en el proyecto.

7.8.4.1 Evolución de los principales riesgos detectados

A lo largo del proyecto fueron identificándose y ocurriendo varios riesgos, algunos tuvieron apariciones fugaces y otros, por la naturaleza del proyecto (con alto grado de incertidumbre, descubrimiento y foco en el usuario), se mantuvieron presentes durante todo el camino, variando su intensidad en diferentes momentos. El llevar el proceso mencionado en cada iteración, con su debido registro y actualizaciones, permitió obtener una clara visualización de la evolución de estos riesgos. Se tomaron cinco de los principales riesgos que estuvieron presentes en todo el proyecto, para mostrar en mayor detalle su evolución.

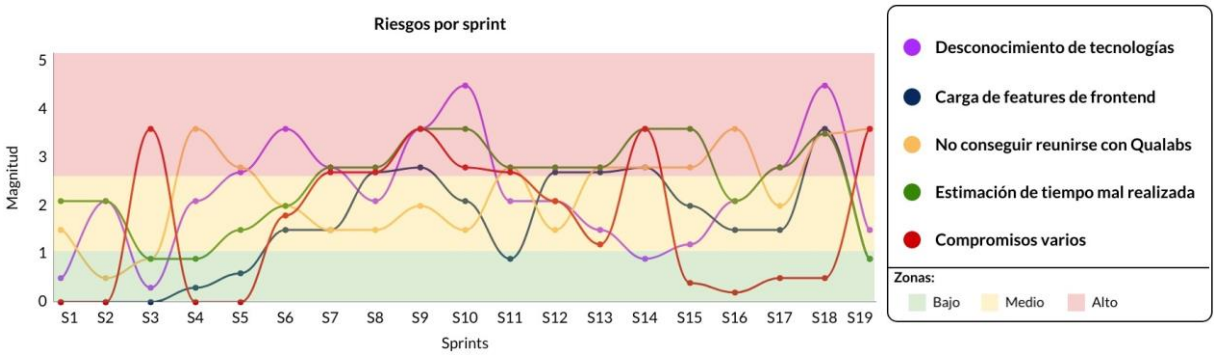


Figura 7.8: Evolución de los principales riesgos en el tiempo

Como se puede observar en la figura 7.8, si bien la intención siempre fue intentar disminuir la magnitud de los riesgos, muchas veces estos dependían de diversos factores fuera del control de los integrantes, condiciones del proyecto, o falta de experiencia, que terminaban haciendo que fuera difícil lograr este fin en períodos sostenidos en el tiempo. Los riesgos tomados en cuenta para este análisis fueron los siguientes:

- **Desconocimiento de tecnologías:** Este riesgo estuvo presente desde el inicio del proyecto y afectó al equipo en diferentes ocasiones durante el transcurso del mismo. Debido a que la empresa del cliente iba a encargarse del posterior mantenimiento y evolución del sistema, estableció las tecnologías, el servicio de *hosting*, y otras herramientas a utilizar. Sin embargo, como se mencionó en la sección de "Características del equipo" del capítulo de "Marco de trabajo", el equipo contaba con poca o ninguna experiencia en las tecnologías a utilizar. Para mitigarlo se llevaron a cabo diferentes instancias de aprendizaje e investigaciones, lo que permitió (en muchos casos) disminuir la magnitud del riesgo correctamente (que se puede apreciar con las bajadas en la gráfica). Sin embargo, en varias ocasiones el equipo se enfrentó con

errores, problemas o tareas que terminaron tomando más tiempo del pensado debido al desconocimiento (como fue el caso por ejemplo de la creación de *pipelines* y configuración del despliegue automático entre el repositorio y el ambiente de la nube).

- **Cargas de *features* de *frontend*:** Avanzado el proyecto, el equipo se percató del gran peso de desarrollo de *frontend* que iba a tener el sistema, suceso que no fue menor, ya que el equipo estaba dividido, por experiencia y comodidad, en *backend* y *frontend*. Sin embargo, del equipo de *frontend*, no todos los recursos contaban con conocimientos de la tecnología utilizada y por tanto debía especializarse y aprender sobre la marcha. Sumado a esto, el equipo de *backend* debía también especializarse en esta tecnología y además en el desarrollo de *frontend* (ya que no era su fuerte). Para mitigarlo, los integrantes vieron videos y cursos para el aprendizaje de React. Además, en el caso del área de *backend*, como estos no habían estado activos desde el inicio en el desarrollo interno de *frontend*, por estarse dedicando a la otra parte, previo a comenzar las tareas en este sector, tuvieron un traspaso de conocimiento, donde se informaron de la estructura del sistema y las buenas prácticas que se estaban manteniendo. A pesar de los esfuerzos, en algunas ocasiones esto resultó en un desarrollo más lento, y resaltó la falta de recursos con experiencia en *frontend.*
- **No conseguir reunirse con Qualabs:** Como se mencionó en capítulos anteriores, por las características y contexto del proyecto, la comunicación con el cliente era vital para el desarrollo de este. Sin embargo, los integrantes del grupo del cliente tienen una agenda muy apretada, haciendo que, en varias ocasiones, fuera una tarea de gran dificultad el conseguir un horario en el que tanto el cliente como el equipo estuvieran disponibles. Si bien la disponibilidad del cliente variaba semana a semana, y se podían conseguir huecos, en ocasiones también surgían eventos o reuniones de mayor prioridad que afectaban la disponibilidad. Este riesgo fue difícil de tratar ya que su control no estaba bajo el poder del equipo, teniendo que muchas veces aceptar la situación o recurrir a un camino alternativo. Con el fin de disminuir su magnitud, además de lo mencionado, el equipo se comprometió a organizarse y planificarse previamente, para intentar prever las reuniones necesarias de antemano y ver el calendario del cliente con antelación para aumentar la posibilidad de conseguir reuniones. También se fijó un horario bisemanal para la reunión de demo, en el que (al momento de establecerlo) todos los participantes podían, comprometiéndose el grupo del cliente a concurrir (o al menos la mayor cantidad de integrantes posibles) y en caso de que ninguno pudiera, notificar

al equipo para re coordinar esa reunión. Asimismo, con el tiempo, conociendo más a los actores del cliente, el equipo pudo entender sus roles y el conocimiento que cada parte podía aportar, lo que permitió establecer reuniones estratégicas con diferentes personas, de acuerdo con las necesidades a resolver, facilitando el conseguir horarios y lograr una buena comunicación. Gracias a esto, se pudo ir controlando el riesgo en el tiempo.

- **Estimación de tiempo mal realizada:** El hecho de que el equipo nunca había trabajado en conjunto como grupo, y que era la primera vez que se enfrentaba a un proyecto de esta magnitud y características, generó varias incertidumbres y problemáticas. La falta de conocimiento entre el equipo, tanto a nivel de desarrollo como personal, hizo que no se tuviera seguridad en las estimaciones, al menos al inicio, porque no se tenía claro las capacidades, velocidad y esfuerzo que tenían los integrantes. El alto grado de descubrimiento del proyecto, también afectó en los tiempos, ya que requería de mucha coordinación y compromiso entre el equipo y el cliente. Además, muchas veces los resultados no eran los esperados, o se cometían errores en el proceso que resultaban en tener que volver a realizarlo, intentando efectuar mejoras o mejorar el entendimiento, retrasando los tiempos pensados. Los riesgos anteriores también afectaron a este, en cuanto a la comunicación, el no conseguir validaciones o reuniones con el cliente generaba que algunas tareas o procesos quedaran estancados. Los riesgos técnicos, hicieron en muchos casos que el desarrollo o ejecución de tareas, insumiera más tiempo de lo planeado. Todos estos factores, junto con el hecho de que a veces no se conseguía un esfuerzo constante en el desarrollo, hicieron que fuera difícil llegar a una estimación acertada para todos los casos, ya que no siempre se lograba cumplir con lo propuesto. Por lo tanto, no se tenía una comparación segura de la cual partir para realizar buenas estimaciones, haciendo que este riesgo fluctuara.
- **Compromisos varios:** Este riesgo abarca todo lo que son vacaciones, visitas de familiares y compromisos sociales que hacen que los recursos no estén disponibles, o afectan el rendimiento de estos por un tiempo. Al inicio se había tomado cada uno de estos riesgos como si fueran distintos, pero luego se vio que todos resultaban en una no disponibilidad debido a una condición social, y por ende se decidió, para su mejor entendimiento y control, tomarlo como uno solo que podía generarse por diferentes motivos. Dos de los integrantes del equipo tienen familiares en el exterior, y tenían previsto sus visitas (o visitarlos) en el tiempo del proyecto, por lo que en ese periodo iban a estar ausentes o con menor disponibilidad hacia este. También, en el tiempo del

proyecto se encontraban las fiestas y el verano, por lo que en algunas ocasiones se dio que algunos integrantes se tomaron tiempos libres o de menor intensidad por estos motivos. Para minimizar la magnitud del riesgo, en los casos de que fuera un largo periodo, se solicitaba al integrante que avisara al equipo con antelación así podían prepararse, y sino se mencionaba en el momento de ver los riesgos del *sprint*. En cualquiera de los casos, se intentaba que el resto del equipo aumentara el esfuerzo para cubrir la baja y además, en caso de ser posible, que el integrante en cuestión se organizara de manera de brindar un esfuerzo mayor antes o después del suceso, o que se hiciera de tiempos libres para intentar disminuir el impacto del riesgo.

7.9 Conclusiones y lecciones aprendidas

En el transcurso del proyecto el *backlog* tuvo diversas variaciones, principalmente debido al peso del *discovery* y la evolución por la que estaba pasando la empresa. Esto hizo que se fuera ajustando continuamente el producto a las necesidades de los usuarios, y el proceso a las características del proyecto. El hecho de contar con una comunicación y *feedback* continuo con el cliente hizo que esto fuera posible, ya que permitió la realización de los cambios de una buena manera y que el equipo lograra un mayor entendimiento de lo que se necesitaba. Esto fue de utilidad para las negociaciones y poder controlar o dirigir de mejor manera la dirección de los cambios.

Además de lo mencionado, el mantener el proceso de *discovery* durante todo el proyecto, realizando refinamientos de requerimientos periódicamente con el cliente, permitió a este darse cuenta, en varios casos, de funcionalidades que no eran necesarias o que de momento la empresa no contaba con la madurez en sus procesos como para poder definirlas. Esto, si bien en una ocasión significó tener que realizar un retrabajo, en muchas otras permitió evitar desarrollo innecesario y potenciar el valor del producto, ya que con estos refinamientos se lograba definir lo que realmente brindaría utilidad a la empresa, beneficiando a ambas partes. Al cliente, porque obtenía un producto adaptado a sus necesidades y que le diera valor, y al equipo, porque podía ver los frutos del trabajo y la satisfacción del cliente.

El hecho de que el cliente comenzara a utilizar el sistema fue de gran valor para el proyecto, ya que dio lugar a una mayor cantidad de *feedback* y fue otro factor que permitió al equipo ir mutando el sistema para que se adaptara a los procesos del cliente. También permitió incorporar

mejoras de usabilidad, que además de brindar un mayor agrado a los usuarios, permitieron mejorar la eficiencia de varias tareas dentro del mismo.

Llevar un registro de los sucesos que iban ocurriendo en el proyecto y algunas métricas, fue de gran utilidad en este contexto de cambios, ya que permitió tener una noción de cómo iba el proyecto, pudiendo descubrir problemas y puntos a mejorar. Esto favoreció a la organización de las tareas y tiempos, lo cual fue muy útil para prever reuniones necesarias con el cliente y lograr conseguir espacio en su agenda.

El realizar la gestión de los riesgos durante todo el proyecto, llevando un registro actualizado en cada iteración fue de gran utilidad ya que permitió tomar cuidados e ir balanceando la carga de trabajo, con el fin de intentar cumplir de la mejor manera con el plan. Si bien no siempre podían eliminarse o minimizar su impacto / probabilidad como se deseaba, ayudó a tomar conciencia de lo que estaba sucediendo y en ocasiones implementar mejoras en los procesos.

En conclusión, el equipo considera que se pudo realizar una gestión del proyecto adecuada, adaptando los procesos a las características de este. Se logró realizar un correcto seguimiento de los *sprints*, ajustando los flujos y cargas de trabajo de acuerdo con las necesidades del cliente, etapa del proyecto y obligaciones académicas.

8 Gestión de la calidad

En el presente capítulo, se definirá la calidad para Ariadne's Thread. Se mencionarán las prácticas utilizadas para el aseguramiento de la calidad, como la aplicación de estándares de código, revisiones y pruebas. Para esto se analizarán métricas y los objetivos relacionados a la calidad.

8.1 ¿Qué es la calidad para Ariadne's Thread?

La calidad siempre fue un atributo de gran importancia. A lo largo del proyecto se fueron incorporando varias prácticas para promoverla y mantenerla. Se definieron objetivos relacionados a la calidad, con los cuales se puede medir si se cumple o no con lo esperado. Para ver estos objetivos, se puede ir a la sección de “Objetivos del producto” del capítulo Introducción.

Para empezar, el equipo decidió llegar a una definición en conjunto de qué conformaría un producto de calidad, la cual es, aquel producto capaz de satisfacer las necesidades del cliente, en lo posible, evitando errores críticos tanto en el código como en el proceso de construcción del producto de *software*.

En base a los objetivos de calidad del producto establecidos al comienzo del proyecto y consideraciones de mantenibilidad y usabilidad, el equipo decidió enfocarse en una serie de puntos los cuales le parecía necesario establecer métricas para su control:

- Que el cliente sienta que el equipo cumplió con sus expectativas y que el producto cumpla con sus necesidades.
- Generar un producto que funcione como el cliente espera.
- Cumplir con lo establecido en los objetivos y requerimientos no funcionales que afecten aspectos de la calidad.
- Tener una buena usabilidad de la interfaz de usuario

Para estos objetivos, se estableció una medición de éxito para evaluar si se cumplieron.

8.2 Prácticas de aseguramiento de la calidad

8.2.1 Aplicación de estándares

Se definieron estándares a seguir a lo largo del proyecto, tanto de documentación como de código y diseño UX/UI. Éstos fueron definidos con el fin de favorecer la mantenibilidad y asegurar que todos los miembros del equipo sigan las mismas prácticas.

8.2.1.1 Aplicación de estándares de código

Al principio del proyecto, el equipo definió un estándar de buenas prácticas a seguir durante el desarrollo del producto. Estos estándares fueron verificados a lo largo del proceso de desarrollo y fueron acordados y aprobados por el cliente.

El equipo optó por seguir las prácticas de *Clean-Code* [62], para asegurar la mantenibilidad y unificar el estilo de programación ya sea en *backend* como *frontend*.

Algunas de las de técnicas y normas utilizadas fueron:

- El uso de nombres nemotécnicos (para clases, variables y métodos).
- El uso de *CamelCase* para variables, métodos y clases, cumpliendo el estándar de nombre de acuerdo con su tipo (ej: métodos comienzan en minúscula y clases en mayúscula).
- Mantener consistencia de nombres (ej: *Add, Get*) para las operaciones del mismo tipo.
- No dejar *imports* sin utilizar.
- Todos los métodos privados deben ir debajo de los públicos, si un método llama a otro, el llamador debe ir arriba del método llamado, exceptuando las recursiones.
- Se prefiere no utilizar comentarios, a menos que sea absolutamente necesario, como para dar información.

Tanto en el *backend* como en el *frontend*, se instalaron las extensiones de Prettier [63] y ESlint [64] para asegurar un estándar de código acorde a la industria, y que se cumpla el formato establecido por todos los integrantes de forma automatizada.

8.2.1.2 Documentación

Para el estándar de la documentación se siguieron los siguientes documentos provistos por la Universidad ORT Uruguay:

- Documento N.° 302 - Normas específicas para la presentación de trabajos finales de carrera (TFDC) de la Facultad de Ingeniería. [65]
- Documento N.° 303 - Hoja de verificación de pautas de presentación de trabajos finales de carreras de la Facultad de Ingeniería. [66]
- Documento N.° 304 - Normas para el desarrollo de trabajos finales de carrera. [67]
- Documento N.° 306 - Orientación para títulos, resúmenes o abstracts e informes de corrección de trabajos finales de carrera. [68]

8.2.1.3 UX/UI

Debido a que la usabilidad y la implementación de una interfaz gráfica atractiva fue de gran importancia para el cliente, se decidió utilizar la metodología de *Design Sprint*, como ya fue mencionado en el capítulo de "Marco de Trabajo". El uso de esta metodología ayudó a asegurar que los diseños estén aprobados y revisados por el cliente antes de la etapa de desarrollo.

En la fase de validación, se validó con el cliente que los diseños sean de agrado y que cumplan con sus necesidades. Luego, en la construcción del *software* se verificó que las pantallas implementadas se adapten al modelo previamente avalado.

Durante el proyecto, el equipo se apoyó en las Heurísticas de Nielsen [69], para encontrar oportunidades de mejoras en la usabilidad antes de hacer las pruebas con los usuarios. Se realizó un análisis heurístico, detallando cómo el sistema cumple con éstas. Éste se puede encontrar en el anexo "Heurísticas de Nielsen".

Por último, se realizaron pruebas con usuarios para poder testear el cumplimiento de los diseños de UX/UI con el cliente. Éstos se pueden encontrar en el anexo "*Tests* de usuarios". También se realizó una encuesta de satisfacción para validar si el sistema cumple con los estándares esperados. Ésta se puede encontrar en el anexo "Encuesta de satisfacción".

8.2.1.4 *Definition of Done*

El equipo estableció una serie de reglas globales a las historias de usuario, también conocidas como *Definition of Done.* Los criterios son:

- Codificación acorde a los estándares mencionados en la sección “Aplicación de estándares de código”.
- Cuando se terminan los cambios, se debe crear un *Pull Request* que debe ser aprobado por al menos un miembro del equipo, como será mencionado en la sección “Proceso de *code review”* de este capítulo.
- Todas las pruebas unitarias deben ejecutarse y no fallar. Este aspecto se ve cubierto por Husky, para ver más información, ir a la sección “Pruebas” de este capítulo.
- La funcionalidad debe haber sido probada localmente, y luego en el ambiente de desarrollo en AWS.
- Todos los criterios de aceptación deben ser cumplidos.

8.2.1.5 Registro de errores

Para lo que fue el registro de errores, se le dio un documento tanto al cliente como al equipo con las bases del registro de un error, siendo estos:

- Un título o descripción del error. (Opcional)
- Pasos de reproducción, se describe la lista de pasos que llevó al error. También se puede agregar un contexto que explique el estado del sistema cuando sucedió el error.
- Resultado esperado, que se supone que el sistema debe tener, pero no está logrando.
- Resultado obtenido, que tiene el sistema actualmente debido al error.
- Otros, parámetro opcional que pueden ser imágenes o comentarios de la situación pertinente.

Además, se ideó un proceso para su seguimiento:

- Cuando el desarrollador encuentra un *bug,* se pregunta en el grupo del equipo de Whatsapp si lo encontrado es en efecto uno. En caso de serlo, el equipo discute las posibles causas del error, y la severidad del mismo.

- En caso de que se haya podido detectar y entender la causa del error rápidamente, se soluciona en la rama a la que pertenece, o la rama en la que se está desarrollando.

- En caso de no poder detectar sus causas rápidamente y que se necesite investigación, se lo reporta en JIRA junto con sus pasos de reproducción, o en algunas circunstancias, si no se tiene acceso a esta herramienta, se anota en el grupo de Discord, para luego ser pasado a JIRA. Usualmente se deja en el *backlog* para que luego sea resuelto.

- Si el *bug* es de extrema severidad, y causa problemas en el transcurso normal de la ejecución del sistema, se vuelve imperativo solucionarlo inmediatamente.

- En la mayoría de los casos, los errores son anotados tanto en JIRA como en Discord, con esto, se logra mantener un registro de todos los errores sucedidos a lo largo del desarrollo del producto.

8.2.1.6 Clasificación de errores

Los errores fueron clasificados en base a cinco severidades distintas, siguiendo la escala provista del campo “Prioridad” de los *tickets* de JIRA. Esto logró que pudieran ser visibles las diferentes severidades dentro de los *tickets* de *bugs* en el tablero, ya que se encontraban destacadas con su respectivo símbolo y color, permitiendo la rápida identificación de este atributo a los desarrolladores. Estas son:

- **Mínima**: Los errores de severidad mínima indican errores visuales que no afectan el uso de la aplicación en absoluto, no son más que poco agradables a la vista.

- **Baja**: Los errores de severidad baja son errores visuales que afectan mínimamente al uso de la aplicación.

- **Media**: Son errores que afectan el uso normal del sistema, pero no lo imposibilitan.

- **Alta:** Son errores que afectan gravemente el uso del sistema, haciendo que sea mucho más difícil o engorroso de utilizar.

- **Máxima** (o errores críticos): Estos errores imposibilitan el uso del sistema, es decir, impiden el uso de una funcionalidad o directamente hacen que el sistema sea imposible de utilizar.

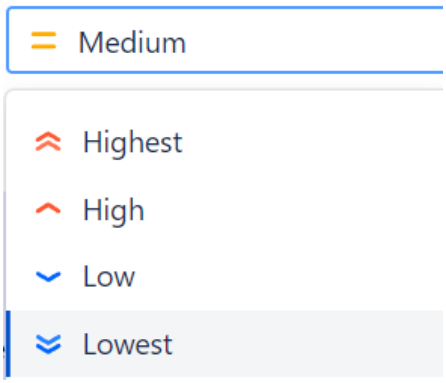


Figura 8.1: Lista de clasificación de errores en JIRA

8.2.2 Capacitaciones

Como ya fue mencionado en capítulos anteriores, el proyecto tuvo un gran componente de investigación y capacitación, debido a que la mayoría de los miembros del equipo no tenían conocimientos respecto a las tecnologías utilizadas.

Las dos investigaciones principales del proyecto fueron respecto a las tecnologías, React para el *frontend,* y Nodejs con Typescript para el *backend.* Para aprender React, se tomó como base la página oficial de React [70], y para Typescript se utilizó la documentación oficial [71].

8.2.3 Revisiones

8.2.3.1 Proceso de *Code Review*

Se implementó como parte estándar del proceso de un *ticket*, la instancia de *Code Review* -o revisión entre pares-.

A la hora de implementar cualquier tipo de cambio, funcionalidad, arreglo o *hotfix,* se crea una nueva rama en el repositorio, siguiendo la nomenclatura establecida en el "Estándares de trabajo para el ramificado", explicado en el capítulo de "Gestión de la Configuración". Al terminar de desarrollar, se crea un *Pull Request* (PR) desde la rama en la que se trabajó dirigido hacia la rama de *develop* (o *master*), comunicando en el grupo de Whatsapp de su creación, y queda a la espera de que otro miembro del equipo pruebe el sistema, revise el código y sus estándares, y valide que cumple con los criterios de aceptación definidos.

Si el código precisa arreglos o mejoras, o si alguna parte de la historia no se presenta cubierta por el código actual, el desarrollador que testea la rama puede sugerir o solicitar cambios con el fin de mejorar la calidad del sistema en general o de hacer notar la falta de completitud en la misma.

En caso de que el sistema funcione conforme a lo esperado, el código esté correcto, dentro de los estándares definidos, funcionen las pruebas y los criterios de aceptación estén cubiertos, entonces se aprueba y luego se hace un *merge* a la rama objetivo (ya sea *master* o *develop*). Para más información sobre el ramificado, ver la sección “Estándares de trabajo para el ramificado” de “Gestión de la Configuración”.

Una vez mergeado a *develop* (o *master*), el desarrollador que probó el *Pull Request*, se encarga de probar la funcionalidad o arreglo en el ambiente de desarrollo, en caso de que no funcione correctamente, se avisa al desarrollador y se arreglan los problemas. Una vez que funcione todo lo probado en la nube, la tarea se considera completa, pasa y puede seguir su camino hacia *Done*.

8.2.3.2 Programación de a pares

El equipo utilizó en ciertas ocasiones la programación de a pares con el fin de que dos desarrolladores trabajarán juntos en una funcionalidad. En la práctica, se utilizaba comúnmente para que un desarrollador menos experimentado en una herramienta, trabajara con otro con más conocimientos, viendo si el código era correcto, y solucionando errores en casos donde sucedieran.

De esta forma, el desarrollador con menor experiencia aprende de su par, mientras el otro supervisa mientras trabaja en otra sección del código o funcionalidad. Esto logra que se produzca un código de mayor calidad, y que el desarrollador menos experimentado gane confianza en su trabajo.

8.2.3.3 Revisiones académicas

En base a lo requerido por la universidad ORT, a lo largo del proyecto el equipo tuvo tres instancias de revisión previas a la entrega.

En cada una de estas se realizó una presentación mostrando el avance del proyecto a un revisor de la universidad, siendo en cada instancia un revisor distinto. De esta manera, el equipo obtuvo

feedback valioso, poniendo a considerar las fortalezas y debilidades del equipo. Los revisores de la ORT fueron Álvaro Ortas, Amalia Álvarez y Martín Solari.

8.2.4 Pruebas

Backend

Se realizaron pruebas automatizadas de integración para probar la solución y se utilizó Husky [72], librería que permite ejecutar *git hooks* al hacer un comando de *git*, configurándola para que corra un *hook*, ejecutando todas las pruebas del sistema previo a un *git push*. Con este, se imposibilitaba la ejecución del comando si uno de los *tests* del *backend* fallaba, por lo que era necesario tener todos los *tests* funcionando para poder subir los cambios realizados. Esto logró evitar errores en el *backend*, dado que aseguraba de que todo siguiese funcionando y que el cambio no generase ningún error. De esta manera, se tenía un control para prevenir errores o encontrarlos de manera temprana y poder solucionarlos.

Como se mencionó, las pruebas automatizadas eran de integración. Se seleccionó está modalidad de *tests* ya que el equipo consideró que los *tests* que se realizaban en estos casos deberían ser significativos, y permitir chequear contra un comportamiento parecido al que tendría la aplicación, ya que se estos invocan a las clases reales y utilizan una base de datos real. Para el caso de este proyecto, se configuró una base de datos de prueba para ejecutar correctamente las pruebas.

Dado que en las pruebas unitarias tradicionales -que *mockean* los componentes del sistema-, cabe la posibilidad de no detectar algún error o problema que se podría dar en un flujo normal con los componentes reales del sistema, se decidió utilizar pruebas de integración para cubrir las funcionalidades y flujos principales del sistema. Este tipo de *testing* se consideró una buena opción debido al tiempo disponible para el proyecto, dando resultados confiables de manera rápida. No obstante, esto causó problemas en los *pipelines*, imposibilitando la ejecución de los mismos, siendo esta la razón clave para utilizar Husky. Para ver más detalles de los *pipelines*, dirigirse al capítulo “Gestión de la Configuración”.

Además, se decidió que, en los casos como los flujos principales o funcionalidades importantes, los criterios de aceptación deberían verse cubiertos por una prueba de integración de *backend*, para así corroborar su correcto funcionamiento en el tiempo.

Frontend

Para las pruebas del *frontend*, en un principio se consideraron hacer *tests* automáticos de interfaz. Al investigar, se descubrió que estos son costosos de realizar [73], conllevan mucho tiempo de mantenimiento y no son sencillos de implementar. Además, estas pruebas, si bien se fijan en que el sistema funcione, no permiten dar resultados acerca del grado de la usabilidad, lo cual es un factor de gran peso en el sistema.

Debido a esto, se decidió no realizar *tests* automatizados de interfaz y dejar que los errores se descubrieran mediante *tests* exploratorios [74]. Si bien estos *tests* automatizados podrían detectar errores, no podrían percatarse de problemas de usabilidad, que sí puede el cliente o un humano.

Para que las pruebas de usabilidad fueran lo más completas posibles, se documentaron escenarios de prueba (también llamados *scripts*), simulando casos específicos, y detallando el comportamiento esperado. De esta manera se asegura, mediante la ejecución de estos, que al realizarse cambios en el sistema, estos no afecten el funcionamiento esperado del mismo.

A lo largo del proyecto, cada nueva funcionalidad fue desarrollada con el proceso de *Design Sprint*, realizando pruebas de usuario para corroborar los diseños. Esto permitió al equipo programar con mayor seguridad, ya que tenían los diseños elaborados y acordados con el cliente.

Además, en cada demo con el cliente, se le mostraba el avance del sistema, recorriendo cada funcionalidad nueva, obteniendo retroalimentación y posibles mejoras. Luego, se dejó deployada una versión en producción para que el cliente lo pudiera ir probando, y que de esta manera pueda sugerir potenciales cambios que se vayan encontrando.

Pruebas con usuarios

Se realizaron pruebas de tipo moderadas y no moderadas [75] con los usuarios del sistema para poder evaluar la usabilidad del mismo. Además, se realizaron *tests* de funcionalidades específicas durante reuniones.

Las pruebas de funcionalidades se realizaron sin planificación anticipada, usualmente en las reuniones bisemanales (*review*), donde se solicitaba a un usuario realizar un *test* de pocos minutos. En estas, se mostraba el avance del sistema, los prototipos realizados y se escuchaban

solicitudes y sugerencias. En varias instancias, se realizaron pruebas rápidas que consistían en que uno de los usuarios presentes, compartiera su pantalla y pruebe la aplicación, narrando en voz alta las acciones que iba tomando. El equipo tomaba nota y además grababa las pruebas para luego poder revisarlas de nuevo.

Para la prueba no moderada, el usuario grabó un video de su pantalla, utilizando la técnica *Thinking Aloud* [75], en donde iba explicando en voz alta cada acción que iba tomando. Luego, analizando este video se pudieron detectar en qué partes el sistema presentaba dificultades, malentendidos o problemas de usabilidad.

Se realizaron además pruebas de usuarios de tipo moderadas (se puede encontrar un resumen de la ejecución y resultados de estas pruebas en el anexo "*Tests* de usuarios"). En esta ocasión, el equipo presentó casos de prueba en el cual le pedía al usuario realizar ciertas acciones. De aquí se podía observar cómo el usuario interactuaba con el sistema en cada escenario. Estas pruebas fueron utilizadas para poder asegurar la calidad y sacar métricas relacionadas a la usabilidad, que se detallarán más adelante.

Por último, se realizó un cuestionario en Google Forms para poder ver el nivel de satisfacción de los usuarios con el sistema. Este también fue utilizado para sacar métricas relacionadas a la usabilidad del sistema. Estos se encuentran en el anexo "Encuesta de satisfacción".

8.3 Métricas

8.3.1 Mantenibilidad

Para el código de *Backend*, se utilizó la herramienta llamada Sonarqube [76]. Esta herramienta de código abierto realiza análisis estáticos de código fuente en base a otras herramientas como PMD o CheckStyle.

Una ventaja que proporciona Sonarqube en lo que respecta a la mantenibilidad, es que brinda un dato que se denomina "deuda técnica" o *tech debt*. Este representa el tiempo estimado en horas que necesita una persona para reparar todos los problemas y fallas detectados por la herramienta. En la siguiente imagen se muestran los diferentes valores obtenidos con esta herramienta respecto a la mantenibilidad.

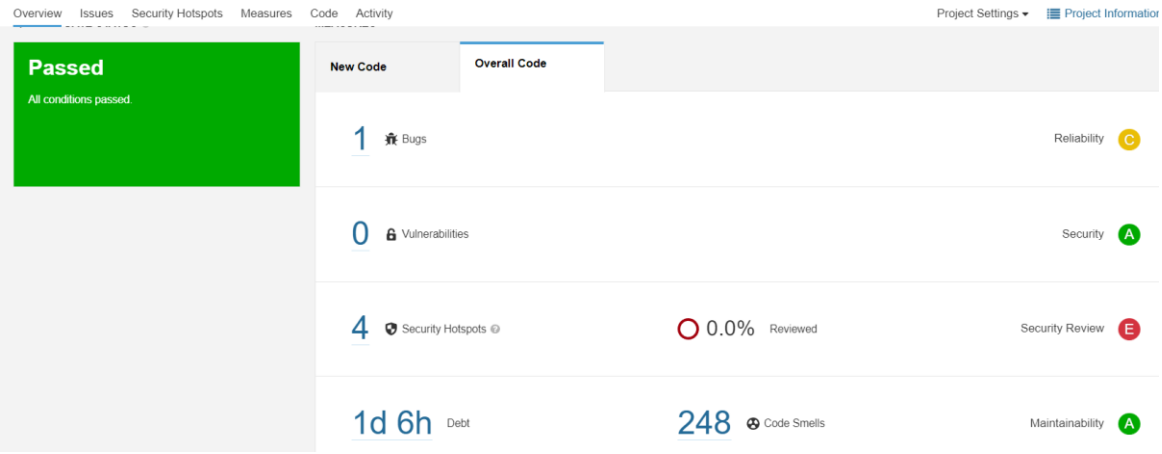


Figura 8.2. Imagen de la ejecución de Sonarqube en *backend*

Como se puede ver, el análisis presenta buenos valores en lo que respecta a la mantenibilidad, un número aceptable de *code smell*s en base a Sonarqube y dentro de los parámetros definidos en los requerimientos no funcionales (RNF 5). La métrica de deuda técnica que utiliza esta herramienta [77], refiere al esfuerzo que detecta que llevaría solucionar los *code smells* de un sistema, estando este esfuerzo tomado en base a días de trabajo de 8 horas. Como se puede observar, del análisis se obtuvo que el sistema presenta un valor de 1 día y 6 horas de deuda técnica (lo que serían 14 horas según la escala mencionada), tomando en cuenta la cantidad de líneas realizadas y el tamaño del sistema, considerando además que el puntaje total es de grado A, se puede decir que se obtuvo un buen resultado.

Por otro lado, el valor de la Security Review es una E. Esto se debe a un problema con CORS [77], y un problema con la librería utilizada para subir las imágenes de los usuarios a un bucket de Amazon S3, lo cual presenta ese problema ya de por sí en su diseño base, haciéndolo imposible de solucionar.

Para el *frontend* y el *backend* se utilizó ESlint, esta librería permitió que se pueda cumplir automáticamente el conjunto de normas definido entre todos los desarrolladores, favoreciendo el buen código y la mantenibilidad, como ya se mencionó previamente en la sección de "Aplicación de estándares de código" de este capítulo.

Para el *backend*, se realizaron pruebas de integración para todas las funcionalidades del sistema. Como ya fue mencionado anteriormente, el fin de las pruebas de integración es probar el sistema, de la forma en la que funciona realmente, estas fueron implementadas utilizando Mocha [78] junto con Chai [79] . Mocha es un *framework* de *Unit Testing* diseñado para ser

usado con Nodejs, es simple de usar, corre las pruebas serialmente y es extremadamente flexible, lo cual le permite trabajar con Chai, el cual provee interfaces para utilizar como *assert*, *expect* y *should*, además de ser altamente extensible, permitiendo utilizar *promises* con *chai-as-promised*. En conjunto, Mocha describe las pruebas y las separa, y *Chai* realiza las llamadas por medio de *request* y las interfaces previamente mencionadas. Para estructurar los *tests* se utilizó el patrón de *Unit Testing Arrange-Act-Assert* [80], este se basa en subdividir una prueba en tres partes principales, *Arrange* -crear las partes que deberán interactuar-, *Act* -actuar, realizar la acción deseada y guardar el resultado- y *Assert* -validar que el resultado es el esperado-. En la figura 8.3 se muestra un ejemplo de uno de los *tests* del sistema, en el que se pueden ver las diferentes partes mencionadas de la estructura.

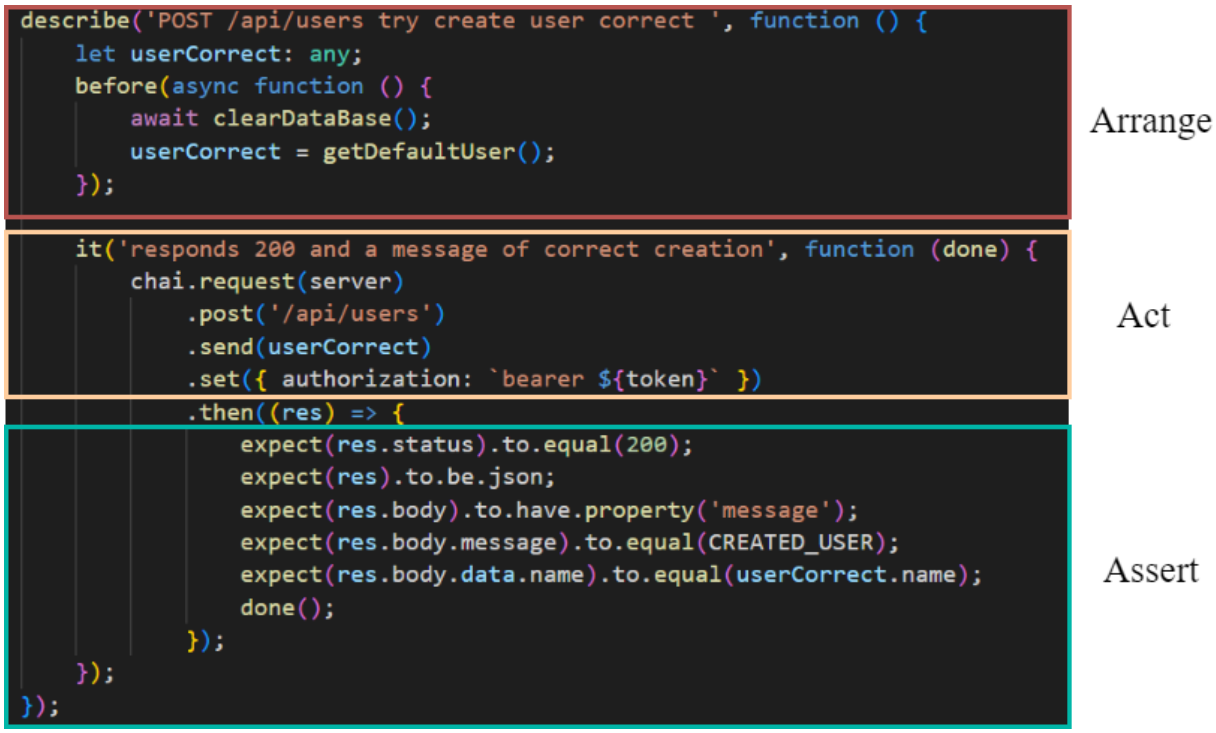


Figura 8.3: Imagen de prueba de integración dividida en secciones

La correspondencia entre secciones de la prueba y Mocha con Chai es la siguiente:

- **Mocha:** Se usa para la organización de la prueba en sí, y para la sección del *Arrange*. También se puede utilizar para la parte posterior a una prueba, de ser necesario.

- **Chai:** Se utiliza para la sección de *Act*, realizando la llamada a la API, y para la sección de *Assert* utilizando la interfaz de *Expect.*

Por último, se mostrará una evidencia de la ejecución de las pruebas en *Backend*. En la imagen se muestra la ejecución de 118 pruebas en 8 segundos gracias a Mocha.

```
User update
  PUT api/users update correct
    ✓ responds 200 and a message that says the update was correct
  PUT api/users repeat mail
    ✓ responds 400 and a message that says the update was incorrect

Get all users
  Should return list with one element
    ✓ responds 200 and an empty list
  Should return list with roles
    ✓ responds 200 and a list with 2 items

Delete user
  Try delete a non existent user
    ✓ responds 404 and a message that says the user could not be deleted
  Delete correct
    ✓ responds 200 and a message that says the delete was correct
  Try delete a deleted user
    ✓ responds 200 and a message that says the delete was correct

Enable user
  Try enable a non existent user
    ✓ responds 404 and a message that says the user could not be enabled
  Enable correct
    ✓ responds 200 and a message that says the enable was correct
  Try enable a enabled user
    ✓ responds 200 and a message that says the enable was correct (84ms)


118 passing (8s)
```

Figura 8.4: Ejecución de las pruebas de integración

Además de las pruebas, debido a que el sistema estaba pensado para ser utilizado diariamente por los miembros de P&C (por lo que debía adaptarse a sus flujos) y que el proyecto presentaba un fuerte componente de *feedback* por parte del cliente, el equipo decidió definir dos métricas para controlar estos puntos. Las métricas definidas fueron el tiempo promedio de solución de un *bug* en *sprints* y tiempo promedio en *sprints* de implementación de un *feedback* dado por el cliente. Por medio de la medición de estas métricas, se pudo ir controlando la velocidad con la que se estaban solucionando estos aspectos, y asegurarse que estuviesen siendo atendidos, o de lo contrario promover su atención, ya que eran necesarios para mejorar el sistema en tiempo y forma. Se ven detallados los *bugs y feedback* que fueron solucionados en el anexo “Diferentes incidencias por *sprint*”.

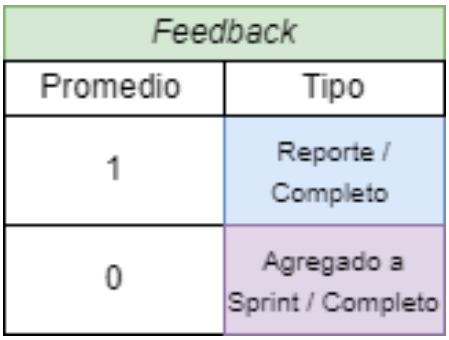
Como se mencionó, la primera métrica tomada fue respecto a los *bugs*, se le llama *Mean Time to Fix* [81] o *Mean Time to Repair*, ya que representa el tiempo entre el reporte del *bug*, y la resolución del mismo. Para esto, se llevó a cabo un *tracking* de los errores que fueron reportados a lo largo del tiempo, y, para agregarle más valor a los cálculos, se tomó en cuenta la severidad de los errores. Con esto, se pudo realizar cálculos en base al tiempo medio en *sprints* para cada severidad, siendo "cero *sprints"* referente a ser solucionado en el mismo *sprint* que se reportó el *bug*, "uno" si fue solucionado en el *sprint* siguiente, y así sucesivamente. En la siguiente figura, se mostrarán los promedios que se obtuvieron, a lo largo del proyecto, para cada tipo de *bug*.

Bugs	
Promedio	Severidad
0	Maxima
0	Alta
1	Media
0	Baja
1	Minima

Figura 8.5: Imagen de los promedios entre reporte y solución de los errores.

Como se puede apreciar, los errores de severidades altas fueron solucionados en los *sprints* en los que fueron reportados, conforme baja la severidad, aumenta el tiempo para solucionarlo, teniendo el pico más alto en la severidad media. Esto se debe a un *bug* relacionado a un cierre de la aplicación, si el *backend* se encontraba apagado, lo cual sesgó ampliamente el promedio de la severidad media. Con estos números, se puede ver que el equipo mantuvo el objetivo de la celeridad a la hora de solucionar *bugs* en el sistema, poniendo énfasis en aquellos de mayor severidad.

La segunda métrica tomada fue el tiempo medio de aplicación de los cambios de *feedback*. Esta métrica fue creada para cubrir la necesidad de tener una forma de evaluar cómo funciona el ciclo de *feedback* del equipo, y visualizar si los cambios solicitados por el cliente se realizaban con celeridad. Para esto, se llevó a cabo un registro de la fecha de reporte, agregado al *sprint*, y finalizado, de cada uno de los *feedbacks* dados por el cliente, siendo evaluado de la misma forma que la métrica de reporte de *bugs*. En la siguiente figura se muestran los promedios entre que se reportó, o agregó a un *sprint*, y se completó.



Feedback	
Promedio	Tipo
1	Reporte / Completo
0	Agregado a Sprint / Completo

Figura 8.6: Imagen de los promedios entre reporte, agregado y finalización de un *feedback*.

Como se puede observar, el promedio entre que se reportó y se completó fue de un *sprint*, mientras que el de agregado a un *sprint* y su finalización fue de cero *sprints*. Cuando se reportaba un *feedback*, este debía ser evaluado mediante el proceso definido para este fin[8], antes de ser solucionado, por lo que es esperable que presente un mayor tiempo frente a la otra medida. Igualmente, este valor significa que, en la mayoría de los casos, el cambio era solucionado al siguiente *sprint*, generalmente quedando solucionado antes de la siguiente s*print review*, pudiendo mostrar en tiempo y forma la solución y los cambios hechos por el equipo, en base al *feedback* provisto por los usuarios del sistema. Respecto a la diferencia entre ser agregado a un *sprint* y su finalización, fueron solucionados en el mismo *sprint*, evitando así que pase a otro *sprint* siguiente.

Objetivos y resultados

En base a las acciones realizadas para la mantenibilidad, y los análisis de métricas utilizados, se podrá ver el cumplimiento o no de los objetivos mencionados anteriormente, ya sean tomados de Objetivos o de Requerimientos no funcionales.

[8] Pueden verse más detalles sobre el proceso seguido para los *feedbacks* en la sección “Gestión del *feedback* recibido” en el capítulo de “Gestión del proyecto”.

- RNF 5**:** Como se mencionó en la sección de Métricas de Mantenibilidad, el rating en *Sonarqube* es A, presenta 258 *Code Smells* en 12.000 líneas de código, y la repetición de líneas es de 4.2%, mostrado a continuación.

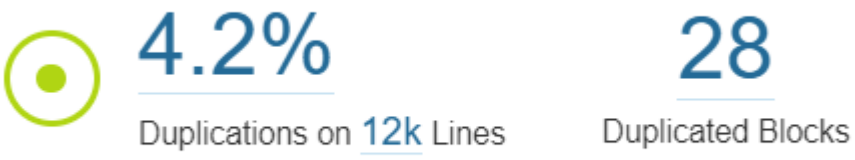


Figura 8.7: Porcentaje de líneas de código duplicadas.

De esta forma, se considera cumplido este requerimiento.

- Objetivo del producto: Generar un producto de calidad

Al momento de entregar el sistema, el producto cuenta con cinco peticiones de arreglo pendientes, dado que el objetivo estableció un máximo de diez arreglos, se concluye que este objetivo fue cumplido. En la figura 8.8 se muestra la completitud de las épicas de *feedback* en el tiempo, tomando en cuenta que T2 y T3 refieren a que estos cambios se realizaron dentro del tiempo de desarrollo del *release* 2 y 3 respectivamente.

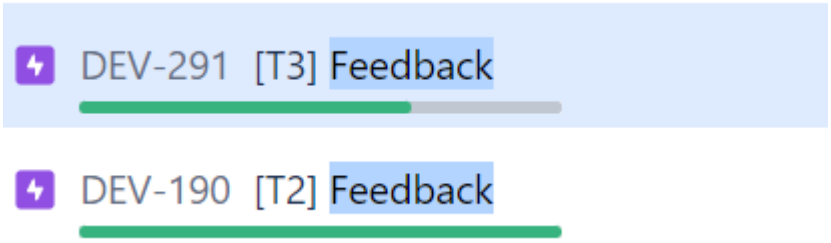


Figura 8.8: Evidencia de la completitud de las épicas de *feedback*

A continuación, en la figura 8.9 se muestran las peticiones de arreglo faltantes, registradas en JIRA.

Figura 8.9: Evidencia de las cinco peticiones de arreglo que quedan.

- RNF 8**:** Como se presentó en la sección de Métricas de Mantenibilidad, los errores fueron solucionados con celeridad, teniendo promedios de cero *sprints* para severidad máxima, alta y baja, y promedios de un *sprint* para los errores de severidad media y mínima. De los resultados se puede destacar que el equipo tuvo una buena velocidad a la hora de enfrentar un *bug* y una rápida solución una vez se detecta la causa del mismo.

  Es por esto que se considera cumplido este requerimiento, alcanzando los valores esperados y en ciertos casos, superándose con creces.

- RNF 9: Como se expuso en la sección de Métricas de Mantenibilidad, las peticiones de arreglo (o *feedback*) fueron solucionadas con celeridad, teniendo promedios de un *sprint* entre reporte y arreglo, y cero *sprints* entre agregado a un *board* y su arreglo. De los resultados, se puede destacar que el equipo tuvo una buena velocidad a la hora de encontrarse e implementar una petición de cambio de *feedback*.

  Se considera cumplido este requerimiento, alcanzando los valores esperados.

- Objetivo del producto: Entregar un producto sin errores críticos.

  Para probar este objetivo, se hizo un escaneo rápido sobre los errores ya resueltos, y los no resueltos que quedan en el sistema, tomando en cuenta que T1 refiere al periodo de tiempo previo al *Release* 1, y así sucesivamente, siendo estos:

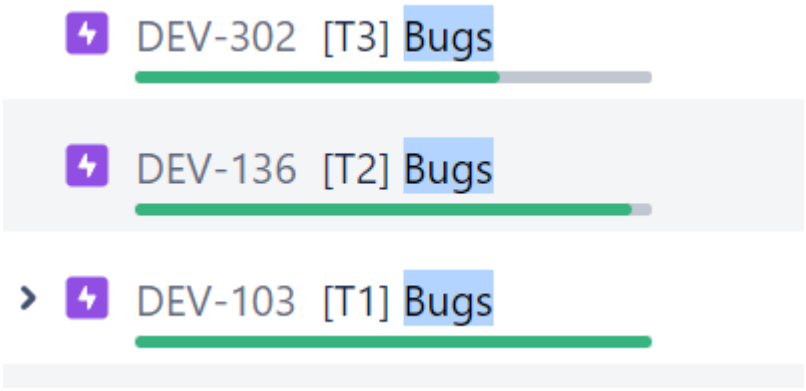


Figura 8.10: *Bugs* restantes respecto a épicas.

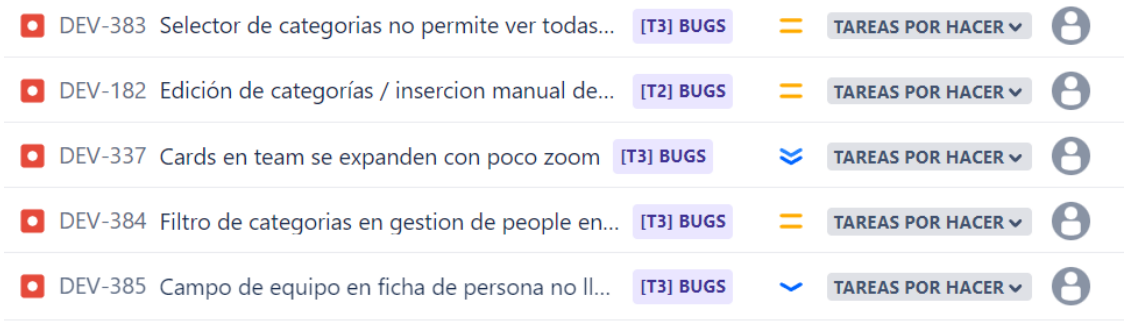


Figura 8.11: *Bugs* restantes

Como se puede ver en las imágenes superiores, quedan cinco *bugs* por solucionar, de los cuales ninguno es de severidad máxima. Tampoco hay de severidad alta, por lo que se considera cumplido este objetivo.

- RNF 7**:** Para este requerimiento se analizaron todos los *bugs* encontrados, separándolos en reportados por el cliente, y reportados por los desarrolladores. El número total de bugs encontrados a lo largo del proyecto es de 55, en las siguientes dos imágenes se verá un desglose entre los encontrados por el equipo, y los encontrados por el cliente.

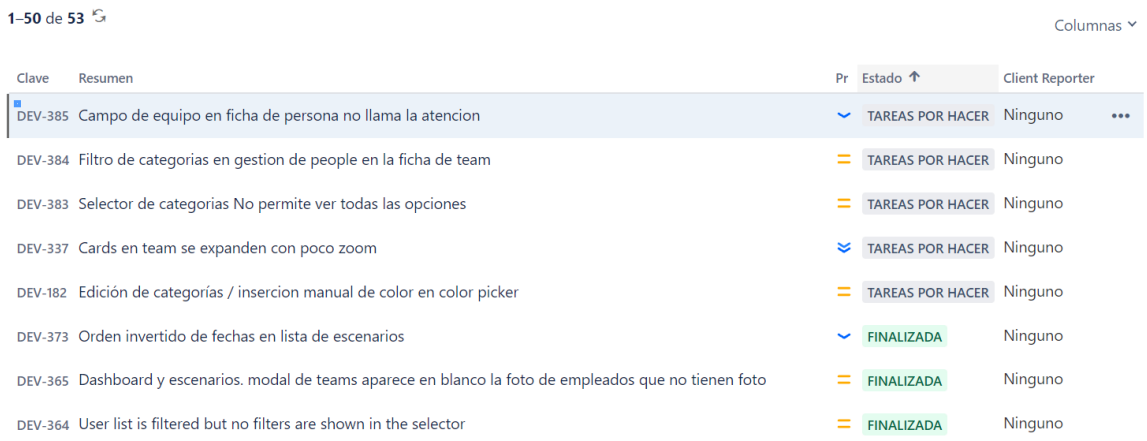


Figura 8.12: *Bugs* reportados por los desarrolladores

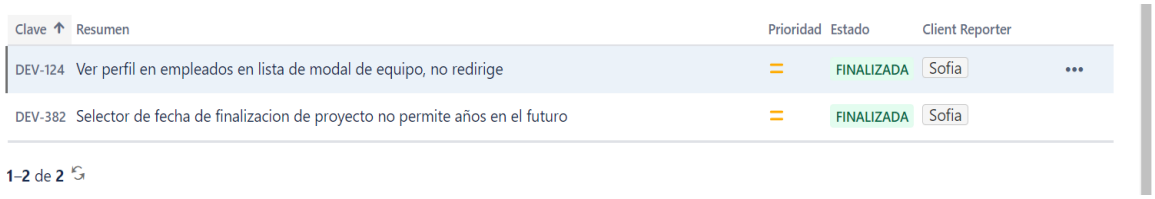


Figura 8.13: *Bugs* reportados por el cliente

En la figura 8.12 se puede ver que el número de *bugs* reportados por los desarrolladores fue de 53, mientras que en la figura 8.13 se observan los 2 *bugs* reportados por el cliente.

Tomando en cuenta estos números, se llega a que 53 de los errores encontrados fueron por el equipo, y 2 fueron por el cliente. Pasándolo a porcentajes, se obtiene que el 96% de los errores fue encontrado por el equipo y el 4% por el cliente. Por lo tanto, de lo expuesto y que este RNF establecía que el 90% o más de los errores debía ser encontrado por el equipo, se puede decir que este requerimiento no funcional de mantenibilidad fue cumplido.

8.3.2 Usabilidad

Como se mencionó en la sección de “Pruebas”, se realizaron pruebas con usuarios y una encuesta de satisfacción. Las pruebas se realizaron con 4 usuarios e incluyeron 7 tareas diferentes. Al realizarlas, se observaron varios factores para luego poder sacar métricas y verificar que se cumplieran los objetivos y los RNF establecidos. Las métricas se dividieron en

tres categorías: la efectividad, la eficiencia y la satisfacción. La efectividad y la eficiencia se midieron mediante los *tests* de usuarios, y la satisfacción mediante una encuesta, los comentarios y los *feedbacks* recibidos durante el proyecto.

Efectividad

Para calcular esta métrica, se plantearon varias tareas para que los usuarios realicen. Durante estas, se observó si el usuario podía completar cada tarea solicitada con éxito, y se tomaron notas de todas las acciones realizadas por el usuario. Se puede ver con más detalles las pruebas en el anexo "*Tests* de usuarios".

El primer usuario fue uno inexperiente, con muy poco uso del sistema. En ocasiones, le costó realizar algunas tareas y se demoraba, pero finalmente pudo realizar todas las tareas solicitadas y así completar con un puntaje de 7 de 7 (o 100%) la prueba.

El segundo usuario fue uno experto en el sistema, ya que lo había utilizado en varias ocasiones. Este realizó las tareas de forma rápida y pudo completar las 7 tareas casi sin ningún problema, concluyéndolas con un puntaje de 7 sobre 7 (o 100%).

El tercer usuario tenía un conocimiento moderado del sistema, teniendo idea de algunas de las funcionalidades básicas, pero no lo había utilizado mucho. Le costó realizar algunas tareas, pero al final, también pudo completar las 7 tareas correctamente.

El cuarto usuario tenía un conocimiento moderado-alto del sistema, es decir, ya había utilizado las funcionalidades iniciales o básicas del sistema, pero no contaba con experiencia en la versión final del mismo. Pudo completar las 7 tareas con éxito.

Como se puede ver, todos los usuarios a los cuales se les realizaron las pruebas, pudieron completar todas las tareas con éxito.

Con esto se puede concluir que se cumplió el RNF 1 y el objetivo de "Buena usabilidad de la interfaz de usuario" con una tasa de éxito de 100%.

Eficiencia

Para la eficiencia, se decidió medir el tiempo que les llevó a los usuarios realizar cada una de las tareas que se plantearon anteriormente.

Como fue mencionado, se realizaron 7 tareas en total, 5 sobre la creación principal de proyecto, equipo y personas, y otras 2 para los filtros del sistema. Para esta métrica, se tomaron en cuenta las 5 tareas principales, sin incluir los filtros, para luego poder comparar si se cumplió el RNF 2.

	Creación de proyecto	Creación de equipo	Agregar persona a equipo	Asignación de persona por búsqueda de filtro	Creación de persona y asignación a equipo	Total
Usuario 1	4.46 min	1.25 min	28 s	30 s	3.05 min	**9.7 min**
Usuario 2	2.13 min	38 s	23 s	27.58 s	1.35 min	**4.9 min**
Usuario 3	2.09 min	48 s	2.17 min	1.10 min	1.20 min	**7.36 min**
Usuario 4	2.01 min	1.02 min	40 s	32 s	1.09 min	**5.32 min**

Tabla 8.1: Tiempo que le tomó a los usuarios realizar cada tarea

Como se puede ver, los usuarios pudieron completar las tareas principales en menos de 10 minutos, cumpliendo así el RNF 2.

Satisfacción

Luego de entregar el último *release* al cliente, se realizó una encuesta de satisfacción. Esta fue enviada a los usuarios principales del sistema. La encuesta constó de una pregunta de tipo NPS [82] (preguntas de múltiple opción con escala de 1-10) para calificar la satisfacción general del sistema. El resto de las preguntas fueron en base a la escala de Likert [83] (preguntas de múltiple opción con escala de 1-5).

Resultados de la encuesta:

NPS

En la figura 8.14 se puede ver la gráfica obtenida a partir de los resultados de la pregunta de satisfacción general con el sistema. Los resultados variaron entre 8 y 10, con la mayoría contestando 10.

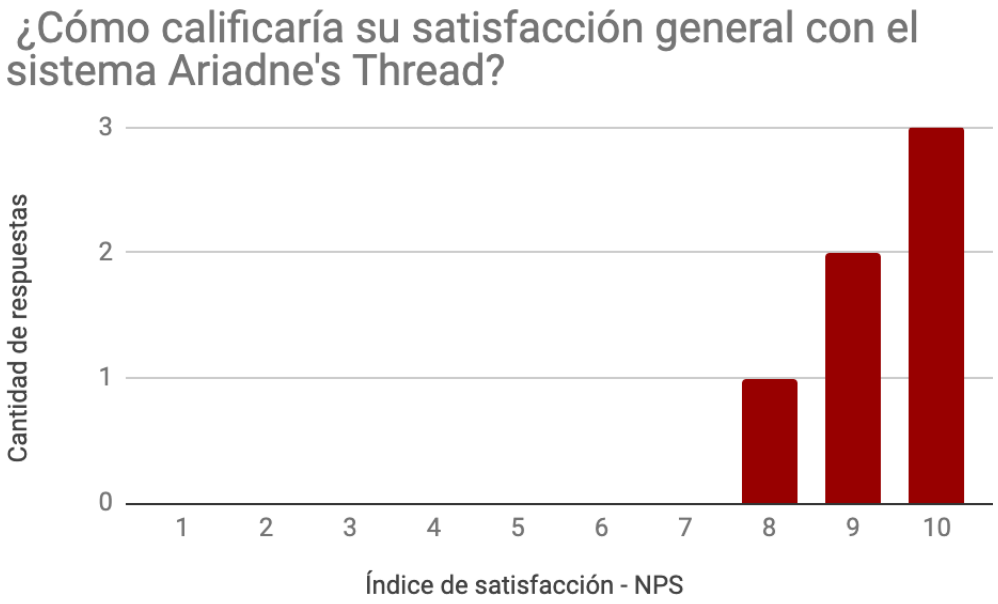


Figura 8.14: Gráfica de los resultados de la pregunta de tipo NPS

Para calcular el NPS, se utilizó la siguiente fórmula:

$$\text{NPS} = \frac{\text{Promoters} \times 100}{\text{Total Respondents}} - \frac{\text{Detractors} \times 100}{\text{Total Respondents}}$$

[84]

Figura 8.15: Fórmula para calcular el NPS

Para hacer más eficiente el cálculo, se utilizó una hoja de Excel que calculaba el NPS de forma automática, lo que permitió obtener los resultados de forma rápida y precisa. Esto se puede encontrar en la sección de “Cálculo de métricas” en el anexo “Encuesta de satisfacción”.

El NPS obtenido fue de 83 puntos. Según *Bain & Company*, el inventor del sistema NPS, "Una puntuación entre 50 y 80 suele considerarse ejemplar" [85], por lo tanto, se puede decir que se logró llegar a un muy buen resultado.

Como se puede ver a partir del NPS y de la gráfica presente en la figura 8.14, la satisfacción del usuario fue alta, con la mayoría de los usuarios eligiendo el puntaje máximo de 10. Una calificación de 9 a 10 es una respuesta positiva [85]. De los resultados se observa que, 5 de los 6 encuestados eligieron estos altos puntajes, habiendo obtenido así un 83% de usuarios satisfechos. Este porcentaje es mayor al 70%, por lo que se puede concluir que se cumplió con el objetivo de "Cumplir con las expectativas del Cliente" y el RNF 3.

Likert

Para calcular las métricas de las preguntas de tipo Likert se utilizó otra hoja de Excel que se puede encontrar en la sección de "Cálculo de métricas" en el anexo "Encuesta de satisfacción". En este se realizó el promedio de las 7 preguntas, obteniendo así una puntuación general para todas las preguntas de tipo Likert.

La puntuación que se obtuvo fue de 4,6. Considerando que Likert es una escala del 1 al 5, se puede decir que este puntaje es muy positivo.

8.4 Conclusiones y lecciones aprendidas

La gestión de la calidad fue uno de los grandes desafíos de este proyecto, dado que se debía crear un producto fuerte en base a las necesidades del cliente, teniendo en mente que este producto sería mantenido por la empresa.

Para esto, se utilizaron prácticas para el aseguramiento de la calidad, como la aplicación de estándares de código, procesos de *Code Review*, definiciones de *Done* trabajando en conjunto con los criterios de aceptación de cada historia, y el manejo de pruebas y revisiones a lo largo del proyecto. También se tuvo en cuenta el manejo y guardado de errores (*logs*), con el fin de llevar un análisis de estos últimos.

De esta forma, el equipo logró generar métricas respecto a la mantenibilidad y usabilidad del sistema, ya sea por medio de herramientas como Sonarqube, métricas respecto a los *bugs*, *feedback* recibido, pruebas de usuarios y encuestas. Gracias al uso de estas métricas, el equipo

pudo evaluar si las acciones realizadas respecto a la calidad fueron efectivas, o, en caso contrario, justificó la razón por la que no se logró el cometido.

A lo largo del proyecto, el equipo pudo profundizar sus conocimientos respecto a la gestión de la calidad, logrando entender sus conceptos, ventajas y dificultades. Así como también expandir sus conocimientos en áreas como la mantenibilidad y la usabilidad de un sistema de software.

Concluyendo, gracias a una buena planificación y creación de estándares, se logró crear un sistema que el equipo considera de buena calidad y que satisface las expectativas del cliente (se puede ver algunos de los comentarios del cliente en el anexo “Comentarios del cliente”).

9 Gestión de la configuración

En este capítulo se mencionarán los diferentes elementos que componen la gestión de la configuración del sistema. Se comenzará por identificar los elementos de configuración, finalizando con el flujo de trabajo para la investigación y para el desarrollo, mostrando el ciclo de vida de las tareas respectivamente.

9.1 Elementos de la configuración

Un elemento de configuración es toda información o artefacto generado como parte del proceso de ingeniería, como los documentos, y el código fuente.

9.1.1 Código Fuente

9.1.1.1 Control de Cambios

El código fuente de la aplicación fue almacenado en dos repositorios de Bitbucket, separados en *frontend* y *backend*. Bitbucket es un servicio de alojamiento de código basado en *git*, el cual pertenece al ambiente de productos de Atlassian como JIRA, Trello, entre otros. Este fue utilizado por el equipo dado que los repositorios fueron provistos por el cliente, lo que permite que el pasaje de código del producto final sea más sencillo. Dicha herramienta permitió al equipo concentrarse en el desarrollo del producto separadamente, para luego poder unificar los incrementos a través de la creación de *Pull Requests* a las ramas comunes (como *develop* o *master*). Además, dado a que esta herramienta contaba con una función de *CI/CD* integrada (*bitbucket pipelines*), el equipo pudo hacer uso de esta para mejorar la calidad del producto.

En cuanto al control de cambios, se utilizó la herramienta de git, que es un sistema de control de versiones distribuido de código abierto, se seleccionó esta herramienta ya que era conocida por el equipo y además, como se mencionó, es compatible con el servicio de alojamiento de código. Esta herramienta permitió a los desarrolladores trabajar separadamente, pudiendo luego unir los cambios, solucionar conflictos, en caso de que hubiera diferencias en el repositorio, y mantener un control de versiones.

9.1.1.2 Estándares de trabajo para el ramificado

Con el fin de conseguir tener un trabajo ordenado y estandarizado, se decidió crear un estándar propio para el ramificado, en base a la metodología de *git flow* [86], para esto se realizaron un par de cambios y adaptaciones, siendo estos:

- **Estándar de nombres**: De manera de poder llevar un control eficiente de las ramas en la herramienta de desarrollo y poder realizar un mapeo más rápido entre rama y *ticket* relacionado, se le asignó a la rama el número del identificador de la historia seguido de una breve descripción de lo que se trata, separado entre guiones para favorecer la lectura. El formato usado fue de ***id-descripción***, de forma tal que se pueda relacionar rápidamente con su correspondiente *ticket* de JIRA. Lo mismo aplica para los *commits*, con identificador de *ticket* y una descripción de lo realizado.
- **Ramas principales**: Las ramas principales se llaman *master* y *develop*. En la primera, se encuentra el código que conforma la versión de producción (el cuál es utilizado exclusivamente por el cliente, y se actualiza con cada *release*), y en la segunda se encuentran los últimos cambios en el desarrollo, que serán reflejados en la siguiente versión del sistema. En esta rama se mantiene el código referente al desarrollo, incluyendo estos los incrementos que el equipo va realizando en los diferentes *sprints*, que finalmente conformarán la siguiente versión del sistema.
- **Ramas de Apoyo**: El funcionamiento de las ramas de apoyo se basa en:
    - **Ramas principales**: Creadas desde *develop*, son nombradas en base a ***id-descripción***, representando a los *tickets* de JIRA. Estas son las que se utilizan para trabajar en la construcción del sistema. Una vez finalizado el desarrollo, se crea un *Pull Request* desde estas hacia *develop* para incorporar los nuevos cambios. El propósito de estas ramas es el desarrollo de funcionalidades, *feedback* y *bug fixing* en general.
    - **Ramas de *hotfix***: Nombradas como ***hotfix-descripción***, estas ramas se utilizan para cambios que deben ser solucionados rápidamente, debido a un error detectado o funcionamiento no esperado de la versión de producción. Se crea la rama nueva desde *master*, se realizan los arreglos correspondientes, creando luego un *Pull Request* hacia esa rama, para incorporar el cambio. Una vez aprobado, este cambio es pasado a *develop*.

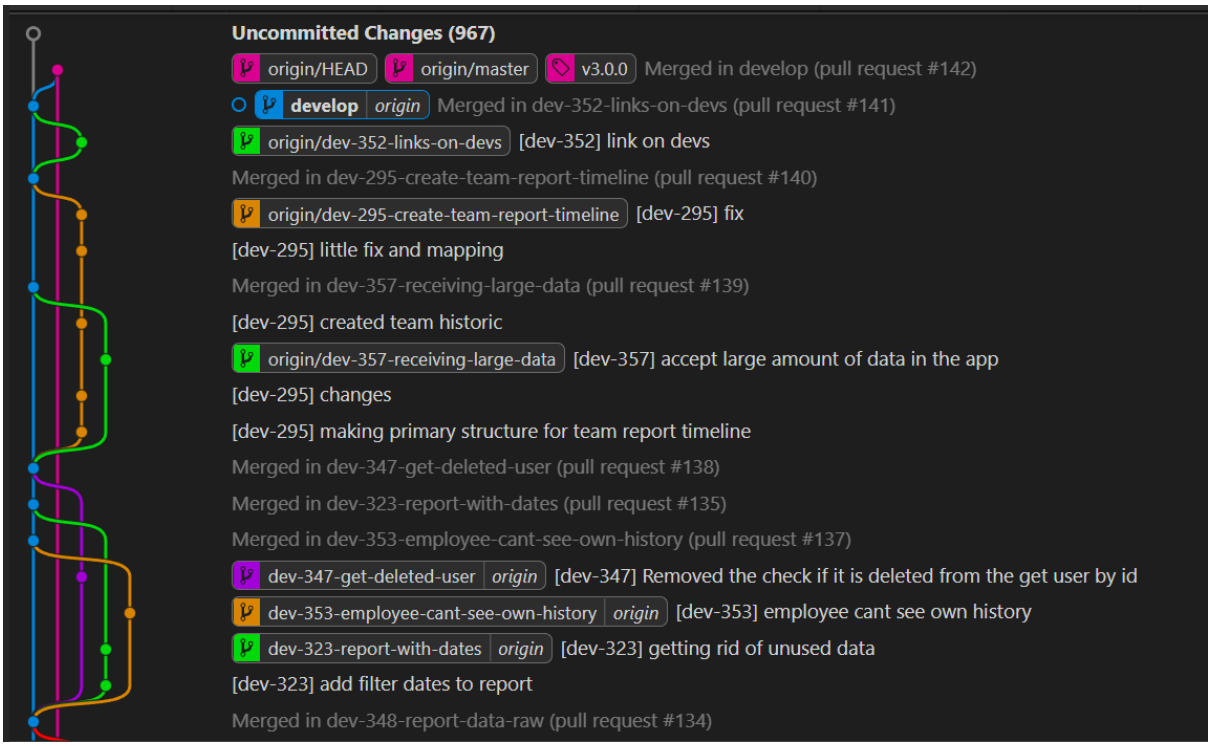


Figura 9.1: Extracto del flujo de ramas en el repositorio de *backend*.

9.1.1.3 Versionado

El versionado para los *releases* del producto y sus actualizaciones se basó en el versionado semántico de Tom Preston-Werner [87] que utiliza tres números X.Y.Z, en donde:

- **X:** Se le denomina *major versión* o versión mayor y hace referencia a la versión principal del *software*. Es incrementada cuando se agregan nuevas funcionalidades importantes (como por ejemplo conjunto de varias funcionalidades que hacen a un nuevo *release*) o se incorporan cambios no compatibles.
- **Y:** Se le denomina *minor version* o versión menor y hace referencia a nuevas funcionalidades. Es incrementada cuando se agregan funcionalidades (no cruciales) que no generan un gran impacto al sistema y son retrocompatibles.
- **Z:** Se le denomina *patch version* o versión parche y hace referencia a arreglos menores. Es incrementado cuando se realizan arreglos de *bugs* que son retrocompatibles.

Respecto a la aplicación del versionado, en cada *release*, debido a que se trataba de un cambio mayor y no retrocompatible, el salto de la versión era en base a la X, por ende en estos casos, los saltos se daban de la forma de 1.0.0 a 2.0.0, y así sucesivamente. Al haber tenido tres

releases, la última versión del producto es la versión 3.y.x (inicialmente 3.0.0, luego se le incorporaron unos arreglos y cambios, quedando la versión final en 3.0.1, en caso de otros *fixes* futuros, se pasará a una subversión de este número). Para el caso de los *patches* y *hotfixes*, el incremento de versión era en base al último número, como pasar de 3.0.0 a 3.0.1 al arreglar un *bug* en una funcionalidad importante del tercer *release*. Además, en el repositorio a los *releases* y cambios se les colocaban *tags* (con su número de versión) para identificarlos rápidamente.

Al liberar un nuevo *release*, se le entregaba al cliente una documentación que contenía un listado de todas las funcionalidades agregadas en la nueva versión, teniendo cada una su respectiva descripción, junto con información sobre actualizaciones relevantes en otros elementos del sistema. Esta documentación se iba actualizando a lo largo del tiempo, para remarcar cambios al sistema, ya sean de requerimientos o por *feedback* del cliente.

En el siguiente diagrama, se muestra un ejemplo de la aplicación de *git flow* junto con el versionado, mostrando los tres *releases* del sistema, y el *hotfix* mencionado en el párrafo anterior.

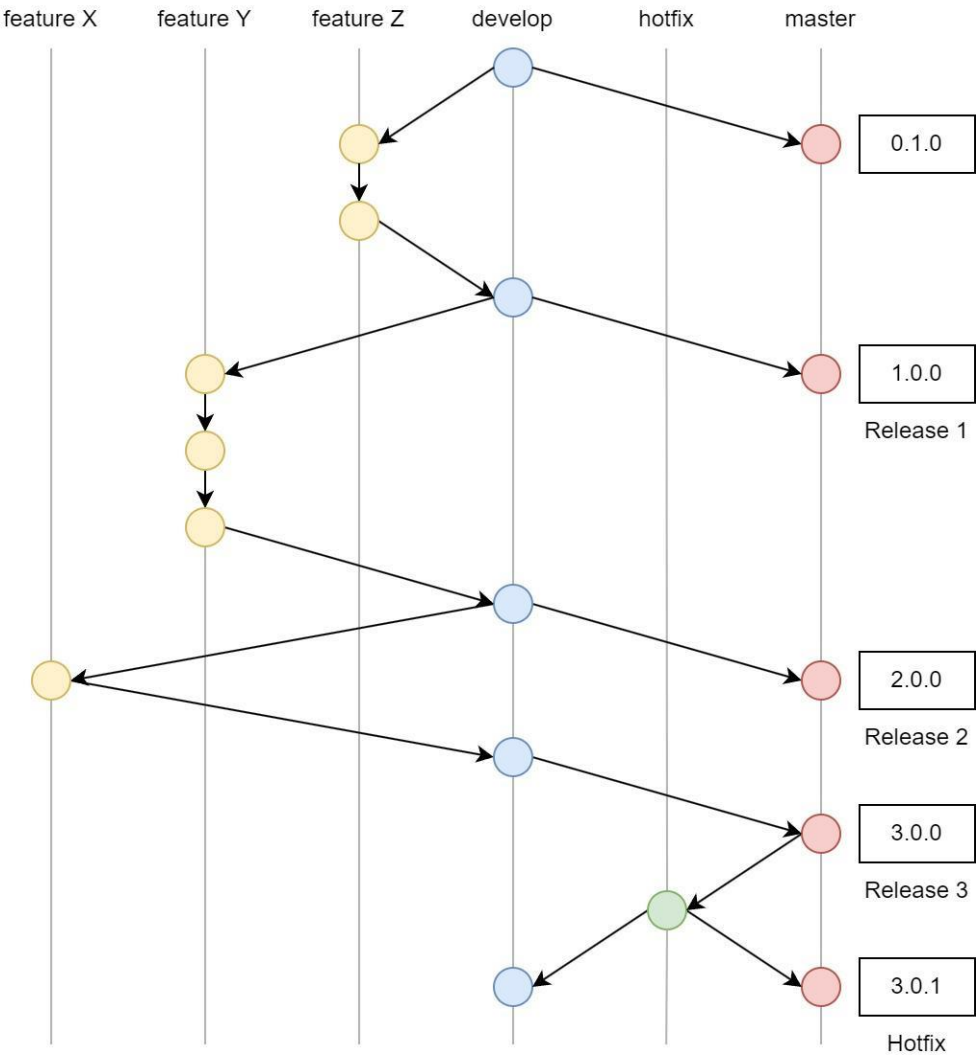


Figura 9.2: Diagrama *de git flow* con versionado

9.1.1.4 Proceso de *Deployment*, Pipelines y CI/CD.

Respecto al proceso de *deployment*, se incorporó *Continuous Integration / Continuous Delivery* [88]. Para esto se crearon y establecieron *pipelines* en los repositorios de Bitbucket, y se crearon entornos en *Amazon Web Services*, de forma tal que cuando se hiciera un *merge* a *develop* automáticamente se construyera y se hiciera el proceso de *deploy* en AWS, logrando mantener de forma actualizada el sistema en la nube, conforme avanza el proyecto. Para más información sobre el proceso de *deploy* y las liberaciones de los *releases*, se puede ver el capítulo de "Arquitectura y Diseño".

Esta forma de trabajo resultó ser una práctica muy útil y ayudó a asegurar la entrega del *software* en tiempo y forma. El equipo incorporó la "integración continua" desde el inicio del proyecto, pues se estaba trabajando en diferentes ramas con diversas funcionalidades, para finalmente

probarlas localmente y unirlas a *develop*. Luego se decidió incorporar la "entrega continua" con el fin de que el cliente pudiera ir obteniendo rápidamente los incrementos generados, y las nuevas versiones del sistema. Esto provocó también que los usuarios pudieran probar el sistema tempranamente. Lo que permitió descubrir falencias en la usabilidad, y brindar valioso *feedback*, funcionalidades para mejorar y mejoras sobre el uso del mismo. Para este proceso de configuración de *deploy* se utilizó el entorno de AWS, dado que era una de las restricciones establecidas por el cliente.

Como último punto de esta forma de trabajo, el estar en un entorno ágil implica la entrega de incrementos usables del sistema a lo largo del proceso, lo cual se agiliza enormemente gracias al uso de CI / CD.

Pipelines

El uso de los *pipelines* permitió que se construya y despliegue el sistema al hacer un *merge* hacia *develop* o *master*, sin embargo, este sistema presentó problemas en la parte del *backend.* Debido a que las pruebas del sistema eran de integración, para poder ejecutarlas en el *pipeline* era necesario tener una base de datos a utilizar en el mismo. La primera opción era realizarlo en AWS, lo cual incurriría en un costo extra que el cliente no estaba dispuesto a pagar. La otra opción, era utilizar una imagen de Docker para levantar la base de datos que el *pipeline* usaría, lo cual implicaría un nuevo componente de investigación, sumado a la inherente dificultad del trabajo con Docker, teniendo que configurar en el *pipeline* que se levantara una imagen de PostgresSQL y la imagen de Docker de la aplicación para este fin. Con ambas opciones descartadas, se decidió tomar un camino alternativo. Se implementó el uso de Husky previo a la subida al repositorio, como fue explicado en el capítulo de "Gestión de la calidad". A continuación, se muestra un ejemplo de una ejecución del *pipeline* en Bitbucket.

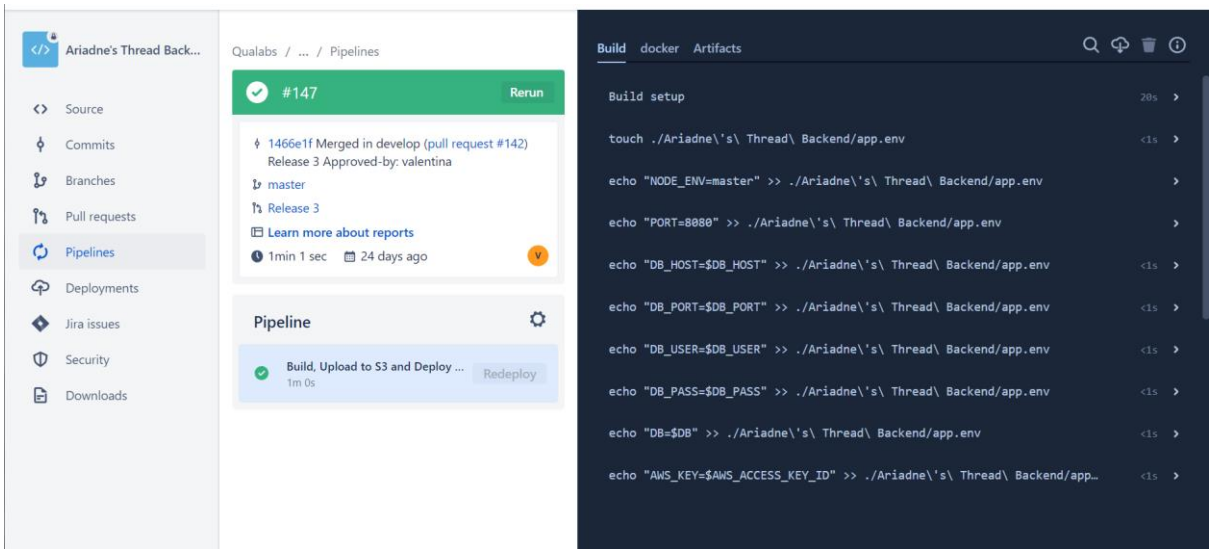


Figura 9.3: Ejemplo de ejecución del *Pipeline*

Para solucionar esta problemática, el equipo decidió utilizar la librería de Husky, la cual permitió que los *tests* se ejecuten frecuentemente con cada *push*. Este hecho se encuentra explicado en mayor detalle en la sección de "Pruebas" en el capítulo de "Gestión de la Calidad". Con esta configuración, se logra solucionar el problema de los *tests* en los *pipelines*, permitiendo que estos sean ejecutados de manera frecuente, asegurando que el cambio incorporado no perjudique el funcionamiento esperado del sistema, y, al mismo tiempo, asegurarse de que el sistema siguiera funcionando como se esperaba.

9.1.1.5 Funcionamiento en la práctica de CI / CD

Como fue mencionado anteriormente, los *pipelines* eran ejecutados activamente siempre que se aceptaba un *Pull Request* hacia *develop* o *master* en cualquiera de los dos repositorios. Al correr unos minutos, Bitbucket avisaba si el *build* funcionaba correctamente o no, en caso de no funcionar, era necesario realizar cambios hasta que este funcionase correctamente.

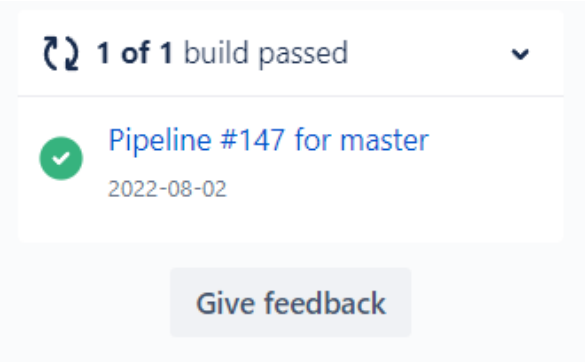


Figura 9.4: *Build* en *Master* funcionando

9.1.2 Documentación

Todos los documentos generados por el equipo cuentan como elementos de la configuración del sistema. Para generar el archivo correspondiente a la documentación académica, se hizo uso de Google Docs y sus múltiples extensiones, esto permitió editar el documento en forma concurrente, y además establecer un formato que permita seguir los parámetros establecidos por el Documento 302 [65]. Para la creación de los diversos diagramas que figuran en este documento se utilizó la herramienta de Diagrams.net.

En cuanto al guardado de los diferentes archivos creados a lo largo del proyecto, se utilizó Google Drive, y sus herramientas de documentación para la creación de los distintos archivos, algunos ejemplos de estos son, documentación informal, borradores de texto, documentos de gestión como gestión de calidad, gestión de la configuración, gestión de riesgos, notas de *feedback*, notas de *reviews*, entre otros archivos. También se utilizó Google Slides para las presentaciones de revisiones, y para el diagramado y evolución del plan de *releases*. Los informes de avance y acciones de revisión fueron realizados en Google Docs.

Por otro lado, la documentación creada del proyecto en sí, documentación para el cliente, investigaciones y otro *set* de reglas a seguir (como el manejo de los *boards*, *pull requests* y estándares de nomenclatura de ramas y versionado) fueron alojados en Confluence. Esta es una herramienta parte del ecosistema de Atlassian para el guardado de documentación, que permite vincular directamente archivos de Confluence a *tickets* de JIRA. También se realizó el *trackeo* de Requerimientos Funcionales y no Funcionales en esta herramienta.

Finalmente, se utilizó Discord, plataforma que, si bien generalmente se utiliza para hablar, chatear y realizar *streamings*, en el caso del equipo se usó como repositorio de datos de interés,

almacenando en esta *links*, notas de *daily meetings*, y también en un principio, el registro de ciertos *bugs* antes de pasarlos a JIRA. También, en ocasiones muy circunstanciales, se utilizó como herramienta para mostrarle al cliente los diseños generados en Figma, como parte del proceso de *Design Sprint*, mencionado anteriormente en el capítulo de "Marco de trabajo".

9.2 Flujo de Trabajo

9.2.1 Ciclo de Vida de una tarea de investigación

Las tareas de investigación se crean durante la reunión de *sprint planning*, o a lo largo del *sprint* dependiendo de si surge un nuevo requerimiento. Para esto, como se mencionó en el capítulo de "Marco de trabajo", se utilizó un tablero de Kanban, en el que se definieron tres columnas:

- ***To Do***: En esta columna se encuentran las tareas referentes a las investigaciones que se consideran necesarias para el avance del proyecto, priorizando las más importantes en base al estado actual del mismo. Cada tarea contiene información relevante para su realización, un nombre explicativo, una breve descripción y objetivos de la investigación, indicando también si existe una salida esperada de la misma, como un documento. Durante la reunión de *sprint planning*, a la tarea se le coloca un tiempo límite en base a su "dificultad" y prioridad. También, se definen ciertos criterios de aceptación para la tarea, ya sean documentos o pruebas.
- ***In Progress***: Una vez que la tarea es tomada por un integrante del equipo o por todo el equipo, se pasa a *In Progress*. Esto genera dos flujos de trabajo distintos, en base a si es tomada individual o grupalmente, siendo estos:
    - **Individual**: El individuo realiza las investigaciones pertinentes, generando un informe o archivos de la investigación, ya sean documentos o pruebas de concepto. Lo generado en la tarea es revisado por otros miembros del equipo, chequeando los criterios de aceptación y discutiendo lo investigado. Si todo está en orden, la tarea se pasa a *Done.*
    - **Grupal**: En caso de ser grupal, la tarea se subdivide en pequeñas secciones para ser realizadas por los integrantes del equipo, realizando varias reuniones para ejecutar las actividades correspondientes. Una vez que se considera que se llegó a un buen resultado, la tarea se pasa a *Done.*
- ***Done***: En esta columna se encuentran las tareas que se consideran terminadas, y que servirán como base para las tareas de desarrollo en el otro *board.*

9.2.2 Ciclo de Vida de una tarea de desarrollo

Como se mencionó en el capítulo de "Marco de trabajo", las historias de desarrollo se crean durante la reunión de *Backlog Refinement*, y en algunas ocasiones también dentro del *sprint*, dependiendo de si surge un nuevo requerimiento o solicitud de *feedback*. Para estas, se utiliza un tablero de Scrum en JIRA, llamado "Tablero de Desarrollo". En la reunión de *sprint planning*, se estiman y se añaden los *tickets* correspondientes al tablero del *sprint*. Con el fin de poder llevar de forma eficiente el trabajo, se crearon 5 columnas, que describen el siguiente flujo de trabajo.

- ***To Do***: En esta columna se colocan todos los *tickets* antes de empezar a ser tomados por uno de los integrantes del equipo. Al ser tomado, el desarrollador se lo asigna y lo pasa a la siguiente columna.
- ***In Progress***: Al pasar a esta columna, se crea una rama de desarrollo con el identificador asociado, y el desarrollador empieza a trabajar.
- ***Code Review***: Al completar el *ticket* y cubrir los criterios de aceptación, este pasa a *Code Review*. Se crea un *Pull Request* (PR) en el repositorio, y queda a la espera de que otro desarrollador lo vea. En caso de estar correcto, se acepta el PR y se *mergea* a *develop*. En caso contrario, si se detecta que no cumple los requisitos necesarios para seguir su camino, se avisa al desarrollador y vuelve a la columna anterior.
- ***Testing***: Una vez en *develop* y completado el *build*, gracias a los *pipelines*, se va a poder testear el *ticket* en el ambiente de desarrollo. La persona que aceptó el PR se encarga de testear que todo funcione correctamente en la nube.
- ***Done***: Una vez completado el *testing*, se pasa a *Done*, lo cual indica que el *ticket* ha sido completado correctamente, está el cambio incorporado en la rama de *develop* y en el ambiente de AWS, quedando lista para ser mostrada el próximo incremento.

9.3 Conclusiones y lecciones aprendidas

La gestión de la configuración fue muy importante para el proyecto, dado que permitió mantener organizados los archivos, repositorios y entregables generales del proyecto, junto con un correcto versionado y manejo de cambios en el código fuente del proyecto. De esta forma, se aseguró un proceso correcto y controlado, previendo posibles problemas como la pérdida de código.

El uso de *git flow* para el ramificado, brindó la posibilidad del trabajo colaborativo y en paralelo entre los miembros del equipo, avanzando con la certeza de que existe un proceso a seguir, siendo este entendido por el equipo. Si bien parte del grupo tenía experiencia con *git flow*, una lección aprendida fue la importancia de mantener una buena nomenclatura de ramas, ya que el formato establecido permitió rápidamente identificar a qué tarea pertenecía la misma, pudiendo detectar si era una *feature*, o era un *hotfix*, dando mayor agilidad y eficiencia al desarrollo. Además, gracias a la descripción se pudo discernir rápidamente si la rama hacía referencia a una *feature*, *feedback* o un *bug*.

Por otra parte, el uso de *pipelines* fue esencial para el despliegue rápido y continuo del sistema, brindando un producto actualizado y que puede ser probado al cliente. Los *pipelines* resultaron ser una buena inversión de tiempo, porque garantizaron que las entregas en un *release* o *hotfix* fueran rápidas, junto con las entregas de forma continua al hacer cambios en la rama de desarrollo.

Finalmente, fue de gran utilidad para el equipo el hecho de utilizar diferentes formas para organizar la información, ideas, investigaciones y reportes, teniendo un sistema especializado para cada parte de la documentación. El uso de Confluence resultó extremadamente útil para escribir investigaciones, y para vincular los documentos con *tickets* de JIRA. Por otro lado, el uso de Discord, que, si bien es usado diariamente como herramienta de comunicación por los integrantes del equipo, resultó útil gracias a sus funcionalidades de creación de canales, favoreciendo a la organización del proyecto. Por último, Google Drive y sus múltiples herramientas y extensiones, resultó clave a la hora de realizar el resto de la documentación, diagramas y presentaciones.

10 Conclusiones

En este capítulo se presentarán las principales conclusiones del proyecto, las lecciones aprendidas por el equipo y los próximos pasos a seguir.

10.1 Conclusiones generales

Las conclusiones generales del proyecto emergen a partir de los objetivos planteados en la sección de “Objetivos” en el capítulo “Introducción”.

10.1.1 Objetivos académicos

Aplicar conocimientos de la carrera

Este objetivo buscaba que el equipo aplicara, a lo largo del proyecto, al menos tres áreas de conocimiento obtenido en la carrera. El equipo cree que este objetivo fue cumplido, dado que se aplicaron conocimientos de diferentes áreas, como la ingeniería de requerimientos, la arquitectura, el diseño, tanto de experiencia de usuario como de interfaz, y las gestiones de configuración, proyecto y calidad. Si bien, parte del equipo ya había puesto algunos de estos conocimientos en práctica en el contexto laboral, el hecho de haber aplicado estos conceptos en un proyecto propio, adaptando los mismos de acuerdo con las necesidades y diferentes circunstancias, y que se haya podido lograr la construcción de un sistema que logró satisfacer las necesidades de un cliente real; fue una experiencia totalmente enriquecedora para todo el equipo.

Puntaje de excelencia

Este objetivo busca un puntaje de al menos 95 en el proyecto final. Dado que esta nota aún no ha sido dictada (cuando se escribió el documento), se considera este objetivo como inconcluso.

Aplicación del proceso de Ingeniería

El objetivo actual plantea tener un 85% de las principales funcionalidades completas a la hora de entregar el proyecto. Al momento de escribir el documento, se encuentra un 100% de las principales funcionalidades, por lo que se considera completo este objetivo. En la siguiente figura, se muestra el plan de *releases* al final del proyecto. Este archivo se fue actualizando

conforme se completaron los *releases* del sistema. Como se puede ver, todas las funcionalidades del plan fueron completadas.

Figura 10.1: Plan de *releases* al final del proyecto.

Uso de nuevas tecnologías

Este objetivo plantea que todos los integrantes aprendan al menos una tecnología nueva a lo largo del proyecto. Como ya fue mencionado en secciones anteriores, solo un miembro del equipo tenía experiencia y conocimiento trabajando con la tecnología utilizada para el desarrollo del *frontend* (React), por lo que el resto del equipo tuvo que aprender dicha tecnología. Otro punto a mencionar fue el aprendizaje del manejo de Bitbucket, *pipelines* y *Amazon Web Services* en general, conocimientos los cuales fueron utilizados para generar instancias de la aplicación en la nube, y mantener un proceso de integración continua. Agregando, dado que se realizaban revisiones de código periódicas cuando se terminaba una funcionalidad o arreglo, los miembros tuvieron que aprender la o las tecnologías asociadas para poder comprender y/o corregir dicho elemento, por lo que efectivamente, todos los miembros tuvieron que aprender al menos una tecnología nueva a lo largo del proyecto.

10.1.2 Objetivos del producto

Cumplir con las expectativas del cliente

Este objetivo plantea cumplir con las expectativas y necesidades del cliente, lo cual el equipo decidió evaluar por medio de una encuesta de satisfacción, teniendo como objetivo un porcentaje de satisfacción mayor al 70%. En la sección de "Satisfacción" del capítulo de "Gestión de la Calidad" se muestra el resultado obtenido por medio del uso de la escala de NPS. El resultado fue de 83% de usuarios satisfechos, superando el objetivo establecido.

Buena usabilidad de la interfaz de usuario

El objetivo planteaba que el 80% de los usuarios pudieran hacer uso del sistema sin ningún tipo de ayuda externa, por lo que se utilizaron pruebas de usuario para validarlo. En la sección de "Efectividad" en el capítulo de "Gestión de la Calidad" se muestran los resultados obtenidos, llegando a un 100% de usuarios que fueron capaces de dar uso al sistema sin ningún tipo de ayuda externa.

Estas pruebas se realizaron con cuatros usuarios con diferentes niveles de conocimiento sobre el sistema. Si bien el número de cuatro personas no parecería ser un número que se pueda considerar como una muestra significativa de usuarios, considerando que son solo 6 personas las que conforman el departamento de P&C del cliente (principales usuarios), es una muestra suficiente para el número de usuarios que van a dar uso al sistema, por lo que se considera cumplido este objetivo.

Generar un producto de calidad

El objetivo plantea que no hayan más de diez peticiones de arreglo (pedidos de cambio por medio de *feedback*) cuando se entregue el producto al cliente. Para esto, se tomó un registro de las peticiones restantes, siendo evidenciado en la sección de "Generar un producto de calidad" en el capítulo de "Gestión de la calidad". En este se puede ver que solamente quedan cinco peticiones de arreglos sin resolver, la mitad de lo que establece el objetivo, por lo que se lo considera cumplido.

Entregar un producto sin errores críticos

Este objetivo plantea entregar un producto sin errores críticos y un máximo de diez errores no críticos, tomando en cuenta el estándar de severidad de *bugs* establecidos en la sección "Clasificación de errores" en el capítulo "Gestión de la calidad". Para esto se tomaron en cuenta los errores restantes, siendo estos cinco en total de varias severidades, pero ninguno de severidad máxima (o error crítico). Estos datos se ven evidenciados en la sección de "Entregar un producto sin errores críticos", en el capítulo de "Gestión de la calidad". Tomando en cuenta estos valores, se puede considerar que este objetivo fue cumplido.

10.1.3 Objetivos del proyecto

Asegurar la entrega continua al cliente

Este objetivo plantea el aseguramiento de una entrega continua al cliente, buscando llegar a unas 60 horas de trabajo de desarrollo por *sprint*, en el proyecto, con el fin de mantener al equipo comprometido con el aspecto tecnológico del mismo. Este objetivo fue analizado previamente en la sección "Horas de desarrollo por *sprints*" del capítulo "Gestión del proyecto", en el cual se demuestra el cumplimiento del mismo.

10.2 Lecciones aprendidas

Interacción con el cliente

El equipo se mantuvo conectado con el cliente a lo largo del proyecto, escuchando el problema y sumergiéndose en el mismo. De esta forma, el equipo pudo entender lo que le afectaba al cliente, y en conjunto divisaron una solución y la fueron construyendo paso a paso. A lo largo del tiempo, el equipo interactuó con el cliente, llevando una buena relación y escuchando el *feedback* provisto por el mismo, llegando finalmente a un producto conforme a sus necesidades y que se adapta a su empresa.

El equipo siente que una de las principales lecciones que se lleva es respecto a la interacción con el cliente. Este considera que pudo mejorar ampliamente sus habilidades de comunicación gracias a la continua interacción que se mantuvo con Qualabs. Los integrantes consideran que los aspectos que mostraron una mejora más clara fueron: la habilidad de escuchar y comprender las necesidades reales del cliente; la capacidad de brindar argumentos válidos en las

negociaciones; y la apertura con la cual recibir y comprender el *feedback* que se les brindaba. Estas habilidades blandas son normalmente difíciles de adquirir en el entorno de la Ingeniería de Software, y a su vez, son muy útiles para poder generar una buena comunicación tanto con el cliente como con el equipo de trabajo.

Metodologías

Otra experiencia que el equipo se lleva, es respecto a las metodologías utilizadas, particularmente *Dual Track Scrum* y todo lo que esto conlleva. El manejo del mismo fue de particular importancia, dado que se debían organizar correctamente las actividades para cada *sprint*, para lograr buenos resultados a lo largo del proyecto. El uso de esta metodología también implicó entender la importancia de realizar investigaciones para entender mejor el problema, y las diferentes formas de solucionar partes del mismo. Por otro lado, también resaltó la importancia de diseñar y validar las interfaces con el cliente antes de implementarlas, logrando una interfaz de agrado para el cliente y evitando el retrabajo.

Gestión de riesgos

Los riesgos fueron un factor influyente a lo largo del proyecto, por lo que el equipo debió tomarlos en cuenta a la hora de planear las iteraciones y distribuir la carga de trabajo. El llevar un registro de estos, permitió que se pudieran detectar y entender problemas en el desarrollo o procesos, pudiendo actuar en base a los mismos con el fin de solucionarlos y/o prevenirlos para el futuro.

Aprendizaje de tecnologías

Finalmente, otra lección que el equipo se lleva, es cómo ir aprendiendo las tecnologías conforme se desarrolla un producto de *software*, organizando los tiempos para dedicar al aprendizaje de las tecnologías sin sacrificar el desarrollo en sí. Este punto fue uno muy desafiante para el equipo, dado su desconocimiento con las tecnologías necesarias para el proyecto y la dificultad de ir entrelazando los tiempos para desarrollar, aprender, investigar y diseñar. En conjunto con el aprendizaje de *Dual Track Scrum*, el equipo logró dividir los tiempos entre las actividades necesarias para llevar a cabo un buen desarrollo del proyecto final de la carrera.

10.3 Próximos pasos

Una vez completado el proyecto, se decidieron un número de acciones a realizar para favorecer el legado del sistema a la empresa Qualabs. Los próximos pasos a seguir son:

- Entregar el acceso a JIRA: Si bien el sistema de Atlassian está dentro de la esfera del cliente, el acceso por el momento es únicamente del equipo de desarrollo, por lo que es necesario legar el acceso a la empresa.
- Entregar documentos al cliente: Como ya fue mencionado en la sección “Entregables adicionales” en el capítulo de “Ingeniería de Requerimientos”, el equipo debe entregar una serie de documentos solicitados por el cliente. Estos son documentos de la estructura de la arquitectura del sistema, instrucciones para levantarlo localmente, entre otros. Además de los documentos, se entregarán las credenciales de acceso al sistema.
- El sistema seguirá siendo utilizado por Qualabs, por lo que se les será entregado, dejando el mantenimiento futuro del producto a cargo de la empresa.

11 Referencias bibliográficas

[1] Asana, “Qué son los objetivos SMART y cómo crearlos con plantillas y ejemplos.” [Online]. Available: https://asana.com/es/resources/smart-goals. [Accessed: Aug. 18, 2022]

[2] E. Sánchez, “El mito Teseo y el hilo de Ariadna,” Oct. 2019. [Online]. Available: https://lamenteesmaravillosa.com/el-mito-teseo-y-el-hilo-de-ariadna/. [Accessed: Aug. 22, 2022]

[3] Scrum, “Como Integrar UX y Scrum - ¿Dual Track Scrum?” [Online]. Available: https://www.scrum.org/resources/blog/como-integrar-ux-y-scrum-dual-track-scrum. [Accessed: Aug. 11, 2022]

[4] Scrum, “Como Integrar UX y Scrum - Adaptar el Framework a UX.” [Online]. Available: https://www.scrum.org/resources/blog/como-integrar-ux-y-scrum-adaptar-el-framework-ux. [Accessed: Aug. 11, 2022]

[5] M. Cagan, “Dual-Track Agile,” Sep. 2017. [Online]. Available: https://www.svpg.com/dual-track-agile/. [Accessed: Aug. 15, 2022]

[6] Google Ventures, “The Design Sprint — GV.” [Online]. Available: http://www.gv.com/sprint. [Accessed: Aug. 12, 2022]

[7] K. S. Rubin, *Essential Scrum: A Practical Guide to the Most Popular Agile Process*. Upper Saddle River, NJ: Addison-Wesley Professional, 2012.

[8] Scrum, “Scrum from the Trenches - Product Backlog Refinement is a Scrum Team Responsibility.” [Online]. Available: https://www.scrum.org/resources/blog/scrum-trenches-product-backlog-refinement-scrum-team-responsibility. [Accessed: Aug. 21, 2022]

[9] Atlassian, “A brief overview of planning poker,” Oct. 2021. [Online]. Available: https://www.atlassian.com/blog/platform/a-brief-overview-of-planning-poker. [Accessed: Aug. 15, 2022]

[10] Atlassian, “Historias de usuario | Ejemplos y plantilla.” [Online]. Available:

https://www.atlassian.com/es/agile/project-management/user-stories. [Accessed: Aug. 25, 2022]

[11] D. Leffingwell, *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise*. Upper Saddle River, NJ: Addison-Wesley Professional, 2010.

[12] Amazon, "¿Qué es AWS?" [Online]. Available: https://aws.amazon.com/es/what-is-aws/. [Accessed: Aug. 27, 2022]

[13] Amazon, "Security groups." [Online]. Available: https://docs.aws.amazon.com/managedservices/latest/userguide/about-security-groups.html. [Accessed: Sep. 27, 2022]

[14] Amazon, "¿Qué es Amazon S3?" [Online]. Available: https://docs.aws.amazon.com/es_es/AmazonS3/latest/userguide/Welcome.html. [Accessed: Aug. 27, 2022]

[15] Amazon, "¿Qué es Amazon CloudFront?" [Online]. Available: https://docs.aws.amazon.com/es_es/AmazonCloudFront/latest/DeveloperGuide/Introduction.html. [Accessed: Aug. 27, 2022]

[16] Cloudflare, "What is an SSL certificate? | How to get a free SSL certificate." [Online]. Available: https://www.cloudflare.com/learning/ssl/what-is-an-ssl-certificate/. [Accessed: Aug. 27, 2022]

[17] Amazon, "AWS | Elastic compute cloud (EC2) de capacidad modificable en la nube." [Online]. Available: https://aws.amazon.com/es/ec2/. [Accessed: Aug. 27, 2022]

[18] Amazon, "AWS | Elastic beanstalk para aplicaciones web desarrolladas con Java." [Online]. Available: https://aws.amazon.com/es/elasticbeanstalk/. [Accessed: Aug. 27, 2022]

[19] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," *J. Res. Natl. Inst. Stand. Technol.*, p. 7, Sep. 2011. [Online]. Available: https://www.govinfo.gov/content/pkg/GOVPUB-C13-74cdc274b1109a7e1ead7185dfec2ada/pdf/GOVPUB-C13-

74cdc274b1109a7e1ead7185dfec2ada.pdf. [Accessed: Sep. 27, 2022]

[20] B. Butler, “PaaS Primer: What is platform as a service and why does it matter?,” Feb. 2013. [Online]. Available: https://www.networkworld.com/article/2163430/paas-primer--what-is-platform-as-a-service-and-why-does-it-matter-.html. [Accessed: Aug. 27, 2022]

[21] Debian, “Debian -- The Universal Operating System.” [Online]. Available: https://www.debian.org/. [Accessed: Aug. 27, 2022]

[22] Nginx, “What is NGINX?” [Online]. Available: https://www.nginx.com/resources/glossary/nginx/. [Accessed: Aug. 27, 2022]

[23] Apache, “Apache Module mod_proxy.” [Online]. Available: https://httpd.apache.org/docs/current/mod/mod_proxy.html. [Accessed: Aug. 27, 2022]

[24] A. Alakeel, “A Guide to Dynamic Load Balancing in Distributed Computer Systems,” *Int. J. Comput. Sci. Netw. Secur.*, vol. 10, Nov. 2009.

[25] GeeksforGeeks, “Horizontal and Vertical Scaling In Databases.” [Online]. Available: https://www.geeksforgeeks.org/horizontal-and-vertical-scaling-in-databases/. [Accessed: Aug. 28, 2022]

[26] Docker, “Docker overview.” [Online]. Available: https://docs.docker.com/get-started/overview/. [Accessed: Aug. 28, 2022]

[27] Docker, “Use Docker Compose.” [Online]. Available: https://docs.docker.com/get-started/08_using_compose/. [Accessed: Aug. 28, 2022]

[28] Linux Commands, “systemctl linux.” [Online]. Available: https://www.commandlinux.com/man-page/man1/systemctl.1.html. [Accessed: Aug. 27, 2022]

[29] PostgreSQL, “What is PostgreSQL?” [Online]. Available: https://www.postgresql.org/about/. [Accessed: Aug. 28, 2022]

[30] JSON Organization, “Introducing JSON.” [Online]. Available: https://www.json.org/json-en.html. [Accessed: Aug. 28, 2022]

[31] PostgreSQL, “8.14. JSON Types.” [Online]. Available: https://www.postgresql.org/docs/14/datatype-json.html. [Accessed: Aug. 28, 2022]

[32] Amazon, “Fully Managed Relational Database - Amazon RDS.” [Online]. Available: https://aws.amazon.com/rds/. [Accessed: Aug. 28, 2022]

[33] Oracle, “Database Backup and Recovery Basics.” [Online]. Available: https://docs.oracle.com/cd/B19306_01/backup.102/b14192/intro001.htm. [Accessed: Aug. 28, 2022]

[34] C. Richardson, “Monolithic Architecture pattern.” [Online]. Available: http://microservices.io/patterns/monolithic.html. [Accessed: Sep. 01, 2022]

[35] M. Fowler, “Micro Frontends,” Jun. 2019. [Online]. Available: https://martinfowler.com/articles/micro-frontends.html. [Accessed: Sep. 03, 2022]

[36] Mozilla Developer, “What is CSS? - Learn web development.” [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS. [Accessed: Sep. 03, 2022]

[37] J. Polacek, “Let’s Define Exactly What Atomic CSS is,” Apr. 2017. [Online]. Available: https://css-tricks.com/lets-define-exactly-atomic-css/. [Accessed: Sep. 03, 2022]

[38] V. Starkov and V. Strukchinsky, “BEM — Block Element Modifier.” [Online]. Available: http://getbem.com/. [Accessed: Sep. 03, 2022]

[39] Styled Components, “styled-components: Basics.” [Online]. Available: https://www.styled-components.com. [Accessed: Sep. 03, 2022]

[40] R. Rendle, “What are CSS Modules and why do we need them?,” Apr. 2016. [Online]. Available: https://css-tricks.com/css-modules-part-1-need/. [Accessed: Sep. 03, 2022]

[41] Sass, “Sass Basics.” [Online]. Available: https://sass-lang.com/guide. [Accessed: Sep. 03, 2022]

[42] M. Fowler, “Microservices.” [Online]. Available: https://martinfowler.com/articles/microservices.html. [Accessed: Sep. 01, 2022]

[43] M. Fowler, “MonolithFirst,” Jun. 2015. [Online]. Available: https://martinfowler.com/bliki/MonolithFirst.html. [Accessed: Sep. 02, 2022]

[44] The New Stack, “Six Strategies for Application Deployment,” Nov. 2017. [Online]. Available: https://thenewstack.io/deployment-strategies/. [Accessed: Sep. 04, 2022]

[45] Amazon, “Deployment methods - Practicing Continuous Integration and Continuous Delivery on AWS.” [Online]. Available: https://docs.aws.amazon.com/whitepapers/latest/practicing-continuous-integration-continuous-delivery/deployment-methods.html. [Accessed: Sep. 04, 2022]

[46] React, “React – A JavaScript library for building user interfaces.” [Online]. Available: https://reactjs.org/. [Accessed: Sep. 06, 2022]

[47] Digital Ocean, “SOLID: The First 5 Principles of Object Oriented Design.” [Online]. Available: https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design. [Accessed: Sep. 16, 2022]

[48] Express, “Express - Node.js web application framework.” [Online]. Available: https://expressjs.com/. [Accessed: Sep. 08, 2022]

[49] REST, “What is REST.” [Online]. Available: https://restfulapi.net/. [Accessed: Sep. 08, 2022]

[50] M. Hoyos, “What is an ORM and Why You Should Use it,” Dec. 2018. [Online]. Available: https://blog.bitsrc.io/what-is-an-orm-and-why-you-should-use-it-b2b6f75f5e2a. [Accessed: Sep. 08, 2022]

[51] TypeORM, “TypeORM.” [Online]. Available: https://typeorm.io/. [Accessed: Sep. 08, 2022]

[52] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 3rd ed. Reading, MA: Addison Wesley, 2012.

[53] Redux, “Redux Essentials, Part 1: Redux Overview and Concepts.” [Online]. Available: https://redux.js.org/tutorials/essentials/part-1-overview-concepts. [Accessed: Sep. 11, 2022]

[54] Intelequia, “Ciclo de vida del software: todo lo que necesitas saber,” Nov. 2020. [Online]. Available: https://intelequia.com/blog/post/2083/ciclo-de-vida-del-software-todo-lo-que-necesitas-saber. [Accessed: Sep. 17, 2022]

[55] Microsoft, “Federated Identity pattern.” [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/patterns/federated-identity. [Accessed: Sep. 14, 2022]

[56] R. Martin, “The Principles of OOD.” [Online]. Available: http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod. [Accessed: Sep. 14, 2022]

[57] npm, “react-beautiful-dnd.” [Online]. Available: https://www.npmjs.com/package/react-beautiful-dnd. [Accessed: Sep. 15, 2022]

[58] Atlassian, “Development and Collaboration Software Company.” [Online]. Available: https://www.atlassian.com/company. [Accessed: Sep. 15, 2022]

[59] Scrum Organization, “What is Scrum?” [Online]. Available: https://www.scrum.org/resources/what-is-scrum. [Accessed: Sep. 12, 2022]

[60] Atlassian, “¿Qué son los puntos de historia y cómo se estiman?” [Online]. Available: https://www.atlassian.com/es/agile/project-management/estimation. [Accessed: Sep. 15, 2022]

[61] R. Toledo and O. Prado, “Planificación de riesgos desmitificada: un enfoque práctico. Paper presented at PMI® Global Congress 2007—Latin America, Cancún, Mexico. Newtown Square, PA: Project Management Institute.,” Nov. 2007. [Online]. Available: https://www.pmi.org/learning/library/es-desmitificando-el-enfoque-practico-de-la-planificacion-de-riesgos-7331. [Accessed: Aug. 26, 2022]

[62] R. C. Martin, Ed., *Clean code: a handbook of agile software craftsmanship*. Upper Saddle River, NJ: Prentice Hall, 2009.

[63] Prettier, “Prettier · Opinionated Code Formatter.” [Online]. Available: https://prettier.io/index.html. [Accessed: Aug. 28, 2022]

[64] ESLint, “Find and fix problems in your JavaScript code - ESLint - Pluggable JavaScript Linter.” [Online]. Available: https://eslint.org/. [Accessed: Aug. 28, 2022]

[65] Universidad ORT Uruguay, “Documento 302.” [Online]. Available: https://www.ort.edu.uy/innovaportal/file/95484/1/normas-especificas-para-la-presentacion-de-trabajos-finales-de-carrera-facultad-de-ingenieria-excepto-biotecnologia__documento-302.pdf. [Accessed: Aug. 24, 2022]

[66] Universidad ORT Uruguay, “Documento 303.” [Online]. Available: https://www.ort.edu.uy/innovaportal/file/95484/1/hoja-de-verificacion-de-pautas-de-presentacion-de-trabajos-finales-de-carreras-excepto-biotecnologia__documento-303.pdf. [Accessed: Sep. 04, 2022]

[67] Universidad ORT Uruguay, “Documento 304.” [Online]. Available: https://www.ort.edu.uy/innovaportal/file/95484/1/normas-para-el-desarrollo-de-trabajos-finales-de-carrera__documento-304.pdf. [Accessed: Sep. 04, 2022]

[68] Universidad ORT Uruguay, “Documento 306.” [Online]. Available: https://www.ort.edu.uy/innovaportal/file/95484/1/orientacion-para-titulos-resumenes-o-abstracts-e-informes-de-correccion-de-trabajos-finales-de-carrera__documento-306.pdf. [Accessed: Sep. 04, 2022]

[69] J. Nielsen, “10 Usability Heuristics for User Interface Design,” abril 1994. [Online]. Available: https://www.nngroup.com/articles/ten-usability-heuristics/. [Accessed: Sep. 04, 2022]

[70] React, “React – Una biblioteca de JavaScript para construir interfaces de usuario.” [Online]. Available: https://es.reactjs.org/. [Accessed: Aug. 28, 2022]

[71] Typescript, “The starting point for learning TypeScript.” [Online]. Available: https://www.typescriptlang.org/docs/. [Accessed: Aug. 28, 2022]

[72] typicode, “husky.” Aug. 2022 [Online]. Available: https://github.com/typicode/husky. [Accessed: Aug. 24, 2022]

[73] J. Unadkat, “What is Automated UI testing?,” Diciembre 2020. [Online]. Available: https://www.browserstack.com/guide/what-is-automated-ui-testing. [Accessed: Sep. 15, 2022]

[74] M. Fowler, “Exploratory Testing,” Nov. 2019. [Online]. Available:

https://martinfowler.com/bliki/ExploratoryTesting.html. [Accessed: Sep. 16, 2022]

[75] J. Nielsen, “Thinking Aloud: The #1 Usability Tool,” enero 2012. [Online]. Available: https://www.nngroup.com/articles/thinking-aloud-the-1-usability-tool/. [Accessed: Sep. 04, 2022]

[76] Sonarqube, “Code Quality and Code Security | SonarQube.” [Online]. Available: https://www.sonarqube.org/. [Accessed: Aug. 28, 2022]

[77] Sonarqube, “Metric Definitions | SonarQube Docs.” [Online]. Available: https://docs.sonarqube.org/latest/user-guide/metric-definitions/. [Accessed: Sep. 26, 2022]

[78] Mocha, “Mocha - the fun, simple, flexible JavaScript test framework.” [Online]. Available: https://mochajs.org/. [Accessed: Aug. 28, 2022]

[79] Chai, “Chai.” [Online]. Available: https://www.chaijs.com/. [Accessed: Aug. 28, 2022]

[80] V. Khorikov, *Unit testing: principles, practices, and patterns*. Shelter Island, NY: Manning, 2020.

[81] Sdlcmetrics, “Mean Time to Fix - Encyclopedia of Software Development Life Cycle Metrics.” [Online]. Available: http://www.sdlcmetrics.org/metrics/mean-time-to-fix. [Accessed: Aug. 28, 2022]

[82] Netpromoter, “What Is Net Promoter?” [Online]. Available: https://www.netpromoter.com/know/. [Accessed: Sep. 04, 2022]

[83] M. Rosala, “Rating Scales in UX Research: Likert or Semantic Differential?,” Jun. 2020. [Online]. Available: https://www.nngroup.com/articles/rating-scales/. [Accessed: Sep. 04, 2022]

[84] T. Fessenden, “Net Promoter Score: What a Customer-Relations Metric Can Tell You About Your User Experience.” [Online]. Available: https://www.nngroup.com/articles/nps-ux/. [Accessed: Sep. 16, 2022]

[85] C. Kenessey, “How To Achieve (And Maintain) An Outstanding NPS,” Mar. 2019. [Online]. Available: https://www.forbes.com/sites/forbestechcouncil/2019/03/08/how-to-achieve-and-maintain-an-outstanding-nps/. [Accessed: Sep. 16, 2022]

[86] Git Kraken, “What is Git Flow | How to use Git Flow | Learn Git.” [Online]. Available: https://www.gitkraken.com/learn/git/git-flow. [Accessed: Aug. 26, 2022]

[87] T. Preston-Werner, “Semantic Versioning 2.0.0.” [Online]. Available: https://semver.org/. [Accessed: Aug. 24, 2022]

[88] M. Fowler, “Continuous Delivery.” [Online]. Available: https://martinfowler.com/books/continuousDelivery.html. [Accessed: Aug. 24, 2022]

[89] R. Krause, “Storyboards Help Visualize UX Ideas,” Jul. 2018. [Online]. Available: https://www.nngroup.com/articles/storyboards-visualize-ideas/. [Accessed: Sep. 22, 2022]

[90] Belbin, “Metodología Roles Belbin.” [Online]. Available: https://www.belbin.es/. [Accessed: Sep. 15, 2022]

12 Anexos

12.1 *Design Sprint*

A continuación se presentarán los prototipos de una funcionalidad, producto del proceso de *Design Sprint*.

En la figuras 12.1 y 12.2 se pueden ver los prototipos realizados para la creación de un empleado, que fueron creados con la herramienta Draw.io. Primeramente, cada integrante del equipo realizó un bosquejo básico y presentó sus ideas a los demás. Luego, se realizaron discusiones y en conjunto se seleccionaron las partes a mantener y descartar.

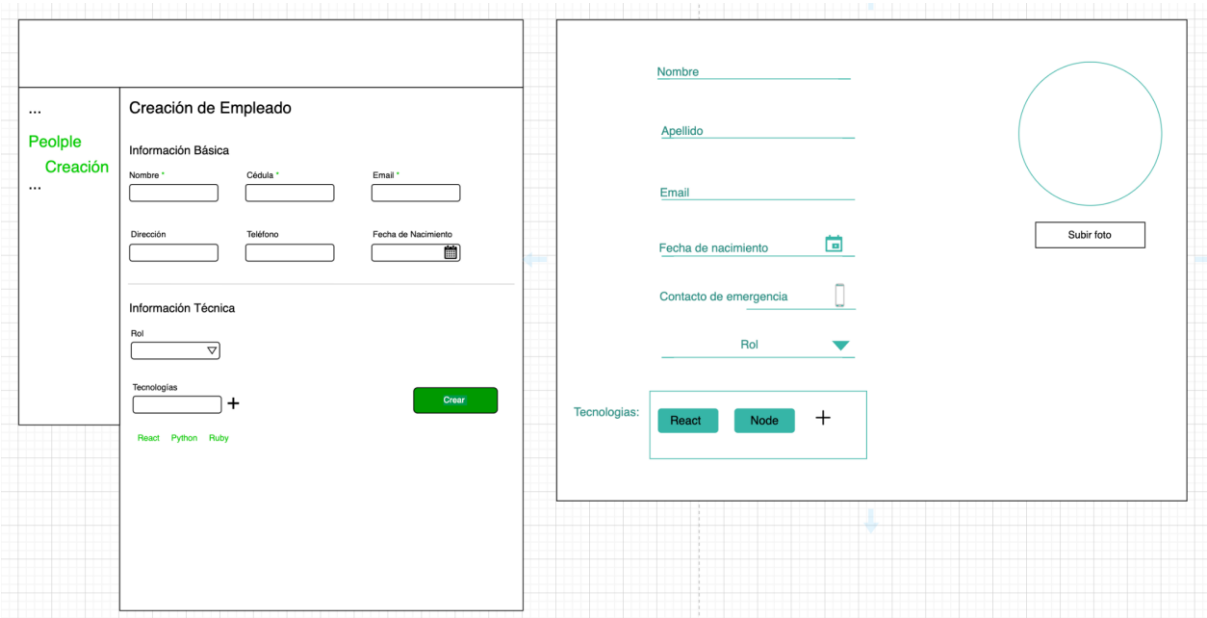


Figura 12.1: Prototipos iniciales que se realizaron en la primera reunión del *Design Sprint*

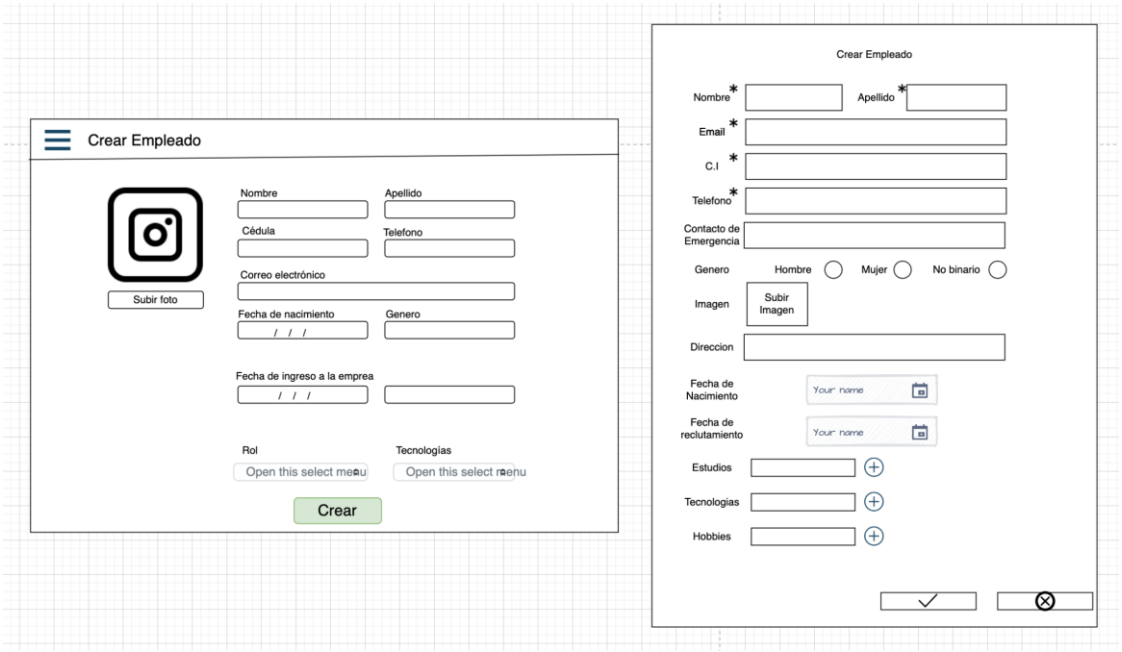


Figura 12.2: Prototipos iniciales que se realizaron en la primera reunión del *Design Sprint*

En la segunda reunión, se tomaron los bosquejos seleccionados anteriormente y se generó un *storyboard* [89]. Éste luego se transformó en un prototipo de alta fidelidad para utilizarlo en las pruebas de usuario. En la figura 12.3 se observa uno de estos prototipos realizado en Figma.

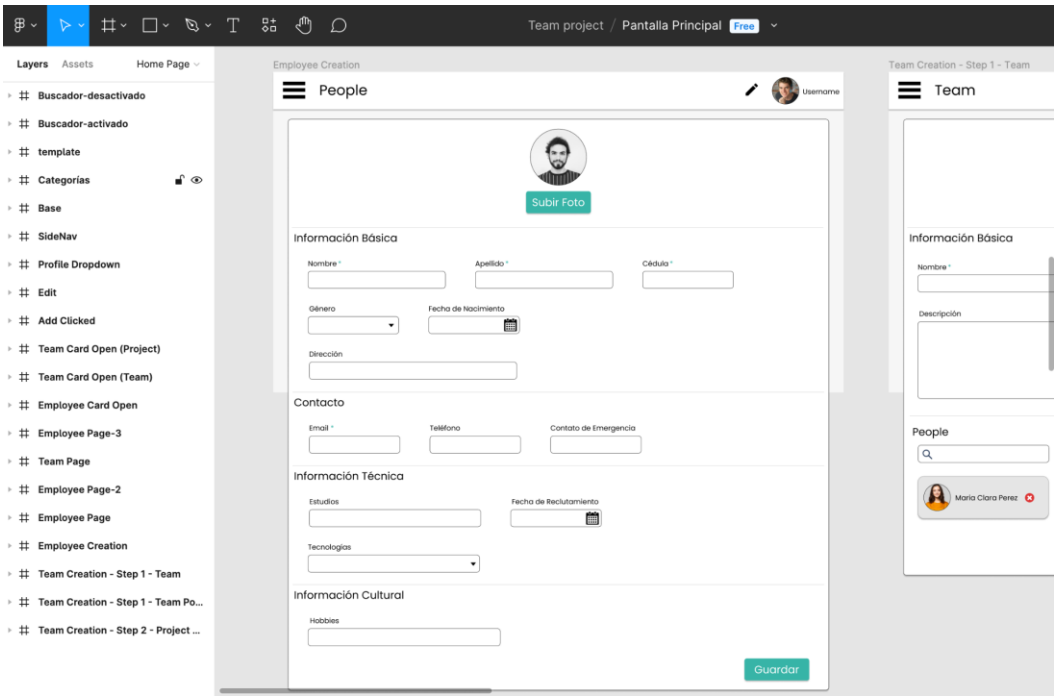


Figura 12.3: Prototipo de alta fidelidad realizando en Figma

Posteriormente se realizó una reunión intermedia, en la cual el equipo revisó la correctitud del prototipo y definió el procedimiento a seguir para las pruebas de usuario.

Por último, se llevaron a cabo las pruebas con el cliente. En éstas, el equipo observó cómo los usuarios utilizaban el prototipo, tomando notas y realizando preguntas. El objetivo fue entender si el diseño les fue de agrado y si lo encontraban sencillo de usar. En la figura 12.4 se puede ver una de las pruebas realizadas con el cliente.

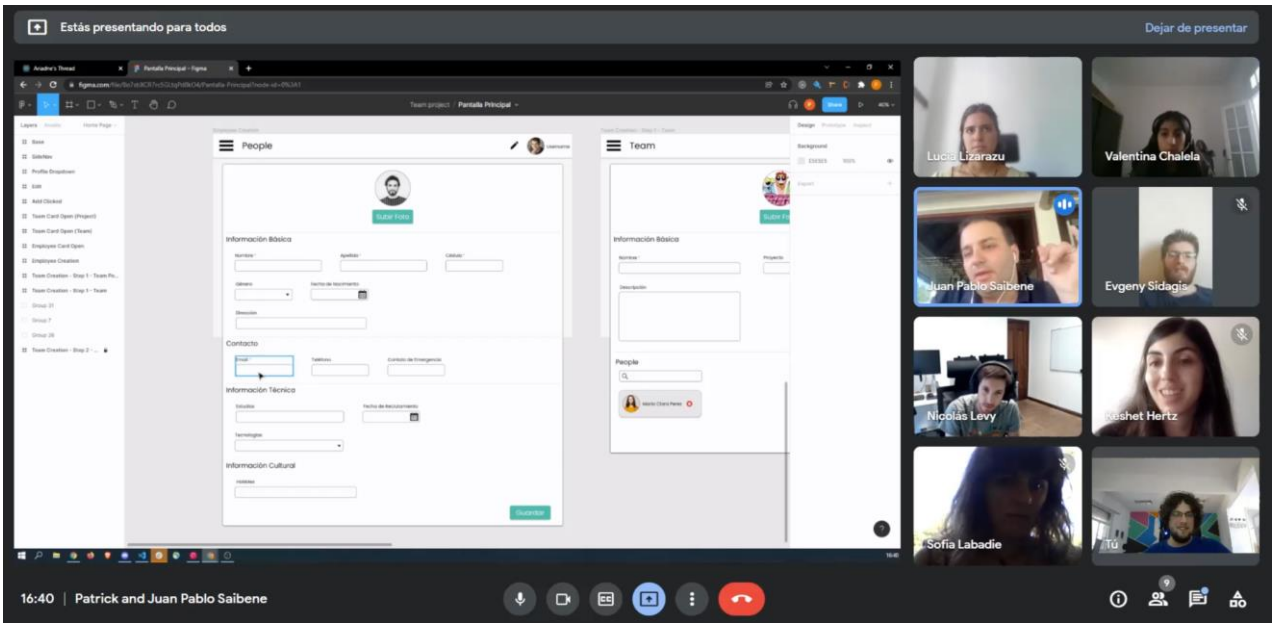


Figura 12.4: Prueba del prototipo con el cliente en Figma

12.2 Investigación de ingeniería inversa

Antes de adentrarse en el desarrollo de la solución, el equipo consideró necesario realizar una investigación sobre los sistemas existentes para ver los productos que se encuentran en el mercado actual, las funcionalidades que tienen y el aspecto visual de estos. A raíz de esta actividad, se logró tener una noción de los productos existentes y dónde estaba posicionada la solución, y en caso de ser posible, poder realizar ingeniería inversa en alguno de ellos.

Para este procedimiento el equipo fue dividido y cada integrante realizó una investigación sobre los productos existentes, donde debían obtener información detallada, ver sus funcionalidades, y de ser accesible probar o, en caso contrario, ver videos acerca de sus funcionalidades, dado que muchos de estos sistemas necesitaban ser parte de una empresa o abonar una cuota para su uso. Finalmente, se debía realizar un breve informe acerca de lo encontrado, mencionando los puntos en común y diferencias con la solución pensada.

Una vez finalizada la búsqueda, el equipo se reunió para compartir sus investigaciones, en donde cada uno realizó una pequeña demostración y explicación de sus hallazgos. Esto fue de mucha utilidad ya que permitió hacer una investigación mucho más rica y extensa en dónde se revisaron más de 10 productos del mercado.

Una vez expuesta la información, se filtró la que era de utilidad y se consolidó en un archivo en el que se ingresaron los sistemas encontrados, una breve descripción de ellos, *links* de acceso y una conclusión general de lo que presentaban estos sistemas.

Finalmente, lo investigado se usó para generar una tabla comparativa en donde se pueden observar ciertas funcionalidades principales (las diagramadas al principio del proyecto para el producto) y la diferenciación que presenta Ariadne’s Thread frente a los existentes, tomando en cuenta los deseos de Qualabs y el problema que tiene la empresa.

	Nuestro producto	Hi Bob	BambooHR	Zenefits	Workday
Enfoca en las personas individuales	●	●	●	●	●
Enfoca en los equipos	●				
Gestión de equipos	●	●			
Autorizaciones basadas en roles	●	●	●	●	●
Enfocado en la cultura	●				
Perfiles de empleados	●	●	●	●	●
Arrastrar y soltar	●				

Figura 12.5: Comparación entre soluciones existentes puntuales y Ariadne's Thread

Para esta comparación, se tomaron en cuenta los sistemas que son gestores de Recursos Humanos por sobre aquellos que eran de gestión de proyectos, dado que el producto a crear debía estar enfocado en las personas, los equipos y la cultura de la empresa Qualabs.

Como se ve en la tabla, se puede concluir que el sistema en desarrollo es un sistema diferenciado, debido a que tiene un enfoque "más humano" a la hora del manejo tanto de los empleados, como de los equipos dentro de la organización. El enfoque de hacerlo más humano implica contemplar a las personas como tal, con sus aspiraciones, gustos y preocupaciones, en contraposición a los otros sistemas que suelen tener una perspectiva más fría/numérica, enfocándose más en las ganancias que los recursos generan que en el bienestar del personal. El sistema permite que se enfoque primero en los conocimientos y otros atributos de los empleados, para luego asignar o rehacer equipos de la forma necesaria, pudiendo el usuario, contrastar pros y contras del cambio, logrando complementar a los integrantes de cada uno de estos.

Ninguna de las soluciones existentes se enfoca a nivel de equipos de la forma que la empresa deseaba y consideraba necesaria, y tampoco poseen la visualización y manejo ideal para el cliente. Además, los sistemas existentes poseen varias funcionalidades adicionales que no resultan de interés a la empresa, y carecen de algunas que la empresa sí necesita. Sumado que muchas veces estos sistemas se venden completos (sin posibilidad de comprar solo ciertos módulos) lo que aumenta los costos generales de la empresa, sin cubrir todas sus necesidades.

Otro detalle no menor a mencionar, es que la empresa buscaba un sistema lo más similar posible a lo que estaban utilizando, evitando incurrir mucho tiempo en el aprendizaje del sistema a utilizar. Por lo expresado anteriormente, se ve necesario un nuevo sistema adaptado a las necesidades y procesos de la empresa.

12.3 *Backlog* del Proyecto

A continuación se detallarán las épicas e historias de usuario principales del proyecto, constituyendo el *backlog* del mismo.

Para definir las historias de usuario se siguió el formato de **Como-Quiero-Para**. Con el fin de facilitar la lectura, se decidió no mostrar los criterios de aceptación dentro de esta lista. En las definiciones de las épicas, se decidió tomar un enfoque genérico, mencionando en todos los casos, el usuario de Ariadne's Thread, dejando la especificación de los actores que participan en las mismas, a cargo de las historias.

12.3.1 *Release* 1

Épica E-1: Gestión de entidades empresariales

Como usuario de Ariadne's Thread
Quiero gestionar las entidades del sistema
Para tener su información actualizada y poder visualizarla fácilmente al gestionar la empresa

HU-1: Gestión de equipos

- **Como** usuario administrador o asistente
- **quiero** crear y mantener un equipo
- **para** poder tener la información de mis equipos actualizada y visualizarla fácilmente al gestionar la empresa

HU-2: Gestión de personas

- **Como** usuario administrador o asistente
- **quiero** crear y mantener una persona
- **para** poder tener la información de los recursos humanos de la empresa actualizada y visualizarla fácilmente al gestionar la empresa

HU-3: Gestión de proyectos

- **Como** usuario administrador o asistente
- **Quiero** crear y mantener un proyecto

- **Para** poder tener la información de mis proyectos actualizada y visualizarla fácilmente al gestionar la empresa

Épica E-2: Visualización organizacional

Como usuario de Ariadne's Thread
Quiero ver, mover y cambiar las entidades del sistema
Para poder visualizar, mover o reorganizar estos elementos, en base a los cambios de la organización

HU-4: Movimientos de Equipos en el *Dashboard*

- **Como** usuario administrador o asistente
- **Quiero** mover y cambiar el orden de los equipos
- **Para** poder reorganizar la información en base surjan cambios en la organización

HU-5: Visualización del equipo en el *Dashboard*

- **Como** usuario de Ariadne's Thread
- **Quiero** ver los datos del equipo y sus empleados en el *Dashboard*
- **Para** tener una visión clara de la estructura actual de la organización

HU-6: Tarjetas desplegables con información de Equipos

- **Como** usuario de Ariadne's Thread
- **Quiero** ver cierta información extra del equipo al clickear su tarjeta en el *Dashboard*
- **Para** tener rápido acceso a información clave de los equipos para tomar decisiones del negocio

HU-7: Movimientos de Empleados en el *Dashboard*

- **Como** usuario administrador o asistente
- **Quiero** mover empleados de un equipo al otro
- **Para** cambiar la diagramación actual de los empleados y realizar cambios dentro de los equipos

HU-8: Gestión de rol de empleados en el *Dashboard*

- **Como** usuario administrador o asistente

- **Quiero** poder ver y cambiar el rol de los empleados dentro de la tarjeta del *Dashboard*
- **Para** poder actualizar rápidamente su rol, para reflejar los cambios en la organización

HU-9: Visualización de no asignados en el *Dashboard*

- **Como** usuario administrador o asistente
- **Quiero** ver las personas no asignadas a un equipo
- **Para** tomar decisiones respecto al equipo al que podrían pertenecer

HU-10: Agregado de personas no asignadas al equipo

- **Como** usuario administrador o asistente
- **Quiero** mover personas sin equipo a un equipo
- **Para** que puedan pertenecer a un equipo y reflejar en el sistema las incorporaciones en los equipos

HU-11: Tarjetas desplegables con información de Empleados

- **Como** usuario administrador o asistente
- **Quiero** ver cierta información adicional de un empleado al clickearlo en el *Dashboard*
- **Para** observar rápidamente las características clave del empleado y poder tomar decisiones de negocio en base a ellas

HU-12 a 14: **Aclaración**: por simplicidad y para no repetir, a continuación se pondrá un único ejemplo con la historia de la lista de empleados, pero la misma fue también realizada para las entidades de proyecto y equipo.

HU-12: Visualización de lista de empleados

- **Como** usuario administrador o asistente
- **Quiero** ver a mis empleados en un formato de lista con nombre, apellido. rol y equipo al que pertenecen
- **Para** ver y poder acceder rápidamente a la información de cada uno de estos.

Épica E-3: Gestión de usuario

Como usuario de Ariadne’s Thread
Quiero poder gestionar usuarios y permitir que estos ingresen al sistema
Para poder tener un registro de los usuarios y que estos puedan utilizar el sistema

HU-15: Login mediante cuentas de Google

- **Como** usuario de Ariadne's Thread
- **Quiero** loguearme al sistema con mi cuenta de Google
- **Para** utilizar las mismas credenciales que utilizo en todos los sistemas referentes a lo laboral

12.3.2 *Release* 2

Épica E-9: Cambios de *Feedback*

Como usuario de Ariadne's Thread
Quiero que se realicen ciertos cambios solicitados
Para que el sistema se adapte a las necesidades del negocio

HU-16: Creación de Proyectos desde Gestión de Equipos

- **Como** usuario administrador o asistente
- **Quiero** poder crear un proyecto nuevo mientras gestiono los datos de un equipo
- **Para** maximizar la eficiencia de trabajo

HU-17: Creación de Equipos desde Gestión de Proyectos

- **Como** usuario administrador o asistente
- **Quiero** poder crear un equipo nuevo mientras gestiono los datos de un proyecto
- **Para** maximizar la eficiencia de trabajo

HU-18: Gestión de Personas en el Equipo

- **Como** usuario administrador o asistente
- **Quiero** poder agregar, cambiar rol y sacar personas del equipo, desde su ficha de equipo
- **Para** mejorar la eficiencia de trabajo, teniendo un punto de acceso distinto para modificar las personas del equipo.

HU-19: Agregado de personas al equipo desde ficha de Equipos

- **Como** usuario administrador o asistente
- **Quiero** Poder tener una sección para agregar personas, pudiendo buscarlas por nombre, filtrarlas por categoría o de la lista de sin equipos

- **Para** poder tener otro punto de acceso para modificar las personas en el equipo mejorando la eficiencia del trabajo

Épica E-1: Gestión de entidades empresariales

HU-20: Creación de Clientes

- **Como** usuario administrador o asistente
- **Quiero** poder registrar y gestionar a los clientes de los proyectos
- **Para** poder tener una lista de mis clientes activos, asignarlos a nuevos proyectos y saber fácilmente sus proyectos y los equipos que están trabajando con ellos al gestionar la empresa

Épica E-4: Gestión de Categorías

Como usuario de Ariadne's Thread
Quiero poder crear y gestionar categorías, y asignarlas a las diferentes entidades del sistema
Para poder representar ciertos aspectos de interés para la organización, pudiendo etiquetar las diferentes entidades con estas y así tener esta información extra disponible para tomar mejores decisiones de negocio

HU-21: Manejo de Categorías

- **Como** usuario administrador
- **Quiero** crear y manejar categorías y subcategorías
- **Para** poder fácilmente añadir información extra a diferentes entidades (equipos, proyectos, personas) según las necesidades que vayan surgiendo de forma dinámica y extensible

HU-22: Asignación de categorías a entidades

- **Como** usuario administrador
- **Quiero** asignar categorías a proyectos, personas o equipos
- **Para** representar ciertos aspectos de interés que estos presentan

Épica E-2: Visualización organizacional

HU-23: Visualización de los puestos vacantes en un equipo

- **Como** usuario administrador o asistente

- **Quiero** ver los puestos vacantes de un equipo clasificados en “por contrato” o “deseables” con colores distintos de acuerdo a su prioridad
- **Para** ver rápidamente cuánta gente falta en un equipo ya sea para cumplir con el contrato, o con los que desea la empresa para el proyecto.

HU-24: Visualización de categorías en las tarjetas desplegables de empleado

- **Como** usuario administrador o asistente
- **Quiero** ver rápidamente las categorías de un empleado en el *Dashboard*
- **Para** hacer más eficiente la toma de decisiones

HU-25: Visualización de categorías en las tarjetas desplegables de equipo

- **Como** usuario administrador o asistente
- **Quiero** ver rápidamente las categorías de un equipo en el *Dashboard*
- **Para** hacer más eficiente la toma de decisiones

Épica E-5: Filtros

Como usuario de Ariadne’s Thread
Quiero poder filtrar en base a las diferentes categorías y sobre algunas entidades del sistema
Para poder acceder o visualizar rápidamente a información de mi interés, para la toma de decisiones de negocio

HU-26: Buscador Genérico

- **Como** usuario administrador o asistente
- **Quiero** poder buscar mediante texto libre en el sistema
- **Para** ver rápidamente los resultados (de proyectos, equipos o personas) que coincidan con el texto ingresado, y cuáles son sus datos

HU-27: Filtro por categorías y subcategorías

- **Como** usuario administrador o asistente
- **Quiero** filtrar los resultados por categorías y subcategorías (de equipos o personas), y marcar las personas que coinciden con el icono de filtro
- **Para** ver que equipos y personas que poseen dichos resultados

HU-28: Filtro por cliente

- **Como** usuario administrador o asistente
- **Quiero** ver los equipos que trabajan con un cliente en particular
- **Para** visualizar fácilmente los equipos que contienen proyectos de uno o más clientes en particular

HU-29: Filtro de equipos incompletos

- **Como** usuario administrador o asistente
- **Quiero** poder filtrar los equipos del *Dashboard* por un estatus de completitud, ya sea por necesidades del cliente o de la empresa
- **Para** poder analizar el estado de los proyectos y equipos que requieren más integrantes de los que cuentan actualmente

Épica E-7: Escenarios

Como usuario de Ariadne's Thread
Quiero gestionar escenarios
Para trabajar sobre la estructura actual de la organización, realizando cambios en la misma sin impactar en la estructura real de la empresa, y poder generar posibles cambios estratégicos.

HU-30: Gestión de escenarios hipotéticos

- **Como** usuario administrador o asistente
- **Quiero** crear y mantener diferentes posibles diagramaciones de empleados y equipos
- **Para** trabajar sobre la estructura actual de la organización, realizando cambios en la misma sin impactar en la estructura real de la empresa, y poder generar posibilidades de cambios estratégicos en el negocio

12.3.3 *Release* 3

Épica E-8: Links personalizables

Esta épica se usó para poner links personalizables en las entidades de empleado, equipo y proyecto. Para promover la lectura, se decidió no listarlas todas y dejar a modo de ejemplo detallado solo el caso de las historias de la entidad de equipo, listando una para los links en la ficha, y otra para la tarjeta desplegable de equipo en el *Dashboard*.

Como usuario de Ariadne’s Thread

Quiero agregar y gestionar links en las entidades del sistema

Para poder almacenar información de referencias a otros sistemas o documentos

HU-31 a 33: Gestión de links personalizables en equipo

- **Como** usuario administrador o asistente
- **Quiero** agregar y gestionar links sobre los equipos
- **Para** poder almacenar información de referencia a otros sistemas o documentos

HU-34 a 36: Tarjetas desplegables con links en el *Dashboard* para equipos

- **Como** usuario de Ariadne’s Thread
- **Quiero** ver los links en la tarjeta desplegable de equipo en el *Dashboard*
- **Para** ver y acceder rápidamente a la información asociada en otro lugar

HU-37: Gestión de links propios

- **Como** usuario empleado
- **Quiero** ver y modificar mis links
- **Para** poder almacenar información de referencia a otros sistemas o documentos.

Épica E-3: Gestión de usuario

HU-38: Gestión de usuarios con roles

- **Como** usuario administrador
- **Quiero** crear y manejar usuarios con diferentes roles
- **Para** crear usuarios y poder controlar el acceso a ciertas funcionalidades del sistema

HU-39: Gestión de ingreso al sistema para empleados

- **Como** usuario administrador
- **Quiero** permitir o denegar el acceso al sistema a los empleados
- **Para** que puedan loguearse, ver sus datos personales, y modificarlos

HU-40: Lista de usuarios

- **Como** usuario administrador
- **Quiero** ver a mis usuarios en un formato de lista

- **Para** ver y poder acceder rápidamente a la información de cada uno de estos.

HU-41: Filtrado de usuarios

- **Como** usuario administrador
- **Quiero** filtrar la lista de usuarios por roles o estado
- **Para** eficientizar el uso de la lista

Épica E-5: Filtros

HU-42: Filtro por selección manual de equipos en el *Dashboard*

- **Como** usuario administrador o asistente
- **Quiero** filtrar los equipos que quiero ver en el *Dashboard*
- **Para** trabajar con un número limitado de estos, viendo los equipos que me interesan en ese momento

Épica E-6: Reportes

Como usuario de Ariadne's Thread
Quiero poder ver visualmente la historia de equipos, empleados y cambios en estos
Para poder analizar los diferentes cambios que sucedieron a lo largo del tiempo y tomar decisiones estratégicas en base a estos.

HU-43: Reporte de línea de vida de equipo

- **Como** usuario administrador o asistente
- **Quiero** poder ver visualmente la historia de un equipo en la empresa, con cambios de entrada y salida de personas, y cambios de roles
- **Para** poder analizar los diferentes cambios que sucedieron en el equipo y tomar decisiones estratégicas en base a ellos

HU-44: Reporte de línea de vida de Empleado

- **Como** usuario administrador o asistente
- **Quiero** poder ver visualmente la historia de un empleado en la empresa, viendo los diferentes equipos y roles que ha tenido

- **Para** ver los cambios que este tuvo a lo largo del tiempo y tomar decisiones estratégicas en base a ellos

HU-45: Reporte de línea de vida propio

- **Como** usuario empleado
- **Quiero** poder ver visualmente mi historia en la empresa, con los diferentes equipos y roles que he tenido
- **Para** ver los cambios que tuve a lo largo de mi tiempo en la empresa

HU-46: Obtención de movimientos en los equipos y cambios de roles entre fechas

- **Como** usuario administrador o asistente
- **Quiero** obtener los datos sobre los movimientos de equipos y cambios de roles, pudiendo elegir entre un rango de fechas
- **Para** visualizar los cambios en la organización realizados entre dichas fechas y realizar operaciones de negocio

HU-47: Obtención de movimientos de personas y cambios de roles en equipos

- **Como** usuario administrador o asistente
- **Quiero** poder obtener un reporte de todos los movimientos y cambios de roles de las personas en los equipos
- **Para** visualizar todos los cambios en el tiempo que se han realizado en la organización hasta la fecha

HU-48: Descarga de reportes a Excel

- **Como** usuario administrador o asistente
- **Quiero** poder descargar los reportes de movimientos y cambios de roles en los equipos en formato Excel
- **Para** poder tener esta información y usarla por fuera del sistema -en herramientas de análisis de datos - y para tomar decisiones estratégicas de negocio.

Épica E-7: Escenarios

HU-49: Guardado de Escenarios

- **Como** usuario administrador o asistente
- **Quiero** poder guardar las nuevas diagramaciones de personas y equipos generadas
- **Para** poder ver posteriormente los diferentes cambios realizados en este y tomar decisiones estratégicas de negocio

HU-50: Visualización de cambios en los escenarios

- **Como** usuario administrador o asistente
- **Quiero** visualizar los cambios entre un escenario y el *Dashboard* actual
- **Para** comparar los cambios realizados contra el *Dashboard* y poder tomar decisiones

Épica E-9: Cambios de *Feedback*

HU-51: Botón de Cancelar

- **Como** usuario de Ariadne's Thread
- **Quiero** poder cancelar la edición en progreso y volver a la página donde me encontraba, alertando si tengo cambios pendientes
- **Para** tener la posibilidad de deshacer los cambios y volver a donde me encontraba fácilmente, sin tener que utilizar la navegación del navegador

12.4 Evolución del plan de *release*

En este anexo se mostrarán los diferentes cambios que fue teniendo el plan de *releases* a lo largo del tiempo, junto con el motivo de los cambios. En las imágenes que se presentarán a continuación se muestra el plan de *release* con sus funcionalidades principales, para dar una idea de lo que es el sistema y hacer amena la lectura. Sin embargo, estas funcionalidades listadas, luego a la hora de desarrollarlas, se desglosan en funcionalidades más pequeñas, que van conformando la totalidad del sistema.

12.4.1 Primera versión

En la siguiente imagen se presenta la primera versión del plan de *releases*, fruto de la primera etapa de descubrimiento e interacciones con el cliente. Se encuentran separados los *releases* por fechas tentativas de inicio y fin, junto con las principales funcionalidades que tendrían. Cabe destacar que, al inicio, estas eran más que nada una serie de ideas ya que no se contaba con una definición asegurada a ciencia cierta y se sabía que iban a haber cambios a medida que se fueran entendiendo más las necesidades del cliente, o mismo cambios por evolución de la empresa. Si bien se realizaba un plan general para tener una guía estructurada a seguir, se iba refinando cada *release*, más en detalle, conforme se acercaba su fecha, de modo de siempre tener seguro lo que se iba a desarrollar e ideas, pero sin tanto nivel de detalle de lo siguiente.

Figura 12.6: Primera versión del plan de *release* acordada con el cliente

12.4.2 Segunda versión

Agregado de funcionalidades tentativas

A inicios de febrero, refinando el plan se agregó la idea de una funcionalidad de "Formularios dinámicos", sin embargo, todavía no se tenía mucha idea de lo que serían, y algunos usuarios querían esta funcionalidad, otros no. El *Product Owner* no estaba muy de acuerdo, pero como faltaba tiempo se decidió dejar como *placeholder* dentro del plan. Se añadió también la idea de poder crear y gestionar categorías, haciendo que los anteriormente pensados "*tags*" pasen a ser en realidad categorías. Definiendo una categoría mediante su nombre y una lista de elementos dentro de la misma, estos últimos funcionarían como "etiquetas" de la categoría. En adición, se aseguró que iba a existir una lógica de asignación de categorías para las diferentes entidades. También se agregó la idea de que puedan existir diferentes roles y vistas con el fin de que esta nueva funcionalidad solo pueda ser utilizada por ciertos usuarios.

Cambio de fecha de primer *release* por petición del equipo

Avanzado febrero, por la segunda quincena, el equipo se da cuenta que no iba a ser posible entregar el primer *release* para fines de febrero o comienzos de marzo, por lo que el equipo le notifica al cliente y se cambia la fecha del primer *release* a marzo (lográndose entregar a fines de marzo).

En la siguiente imagen se ven los dos cambios mencionados, el agregado de funcionalidades y cambio de fecha del primer *release*.

Figura 12.7: Segunda versión del plan de *release* acordada con el cliente

12.4.3 Tercera versión

Con las revisiones y refinamientos del *release*, por la primera quincena de marzo, se eliminaron tres funcionalidades por diferentes motivos. Se muestran en la figura 12.8 las eliminaciones en rojo y a continuación se explicará la motivación de dichas eliminaciones. Además de las eliminaciones, se definieron los primeros filtros que tendría el sistema y dónde se aplicarían estos. También se adiciona la funcionalidad del "Buscador".

Figura 12.8: Tercera versión del plan de release acordada con el cliente

La idea detrás del "Algoritmo de Hipotetización" era hacer una sugerencia de posibles movimientos de personas en base a un algoritmo automatizado, se eliminó debido a que se vio que no encajaba con la cultura de la empresa, ya que pretendía ser un algoritmo automático de sugerencia de equipos. Luego de pensarlo vieron que no querían caer en eso, ya que no consideraban que fuera correcto que ese proceso lo llevara a cabo una máquina, les parecía antinatural y en contra de su cultura y por ende no se iba a utilizar.

"Datos Belbin" al inicio pretendía ser una categoría especial, con indicadores adicionales en base a ciertas reglas de la metodología Belbin [90], posteriormente se decidió eliminarla porque avanzado el desarrollo la empresa dejó de poner tanto énfasis en esta metodología, y por tanto se vio que no iban a ser necesarios estos indicadores ya que no estaban utilizando más esa medida. Además, en caso de que luego se quisiera etiquetar a las personas para mayor información, con su rol Belbin podrían realizarlo mediante la funcionalidad de "Categorías" que iba a ser implementada.

La “Vista para otros roles” suponía la creación de diferentes configuraciones visuales de los elementos en pantalla para los distintos tipos de usuarios (dependiendo de su jerarquía), sin embargo, en este momento el manejo de roles no estaba muy claro (ya que todavía faltaba tiempo para el tercer *release*) y no tenían claro si iban a necesitar muchos roles o si tenían la necesidad de tener visuales separadas para cada uno, por tanto, en el momento no se consideró necesario y se decidió eliminar.

12.4.4 Cuarta versión

Reordenamiento de funcionalidades entre *releases*

Ya entregado el primer *release* y avanzado el desarrollo del segundo, a inicios de mayo, se vió que no se iba a llegar a tiempo con todas las funcionalidades planeadas para la entrega de mayo, por lo tanto, se decidió correr la funcionalidad de los “Formularios Dinámicos” (que todavía no se lograba una definición definitiva de estos) para el tercer *release*.

Agregado de cambios del *feedback*, eliminación y reordenamiento de funcionalidades

En desarrollo del segundo *release*, a mediados de mayo, luego de otra iteración de refinamiento y elicitación con el cliente se decidió eliminar la funcionalidad de “Formularios Dinámicos”, ya que luego de pensarlo no se le vio relevancia o uso suficiente como para incorporarla, y además podía ser cubierta con otras funcionalidades del sistema. En este tiempo, habían llegado al equipo muchas solicitudes de cambios provenientes del *feedback* del cliente sobre el uso del sistema, por lo que se decidió priorizarlos e incorporar algunos al *release*, ya que brindarían valor al sistema y mejorarían su uso (terminan siendo 11 funcionalidades a incorporar y algunos cambios menores). Con el fin de que estos cambios provenientes del *feedback* pudieran terminarse a tiempo, sumado al hecho de que se estaba empezando a bajar a tierra el “Hipotetizador Manual”, y no se había empezado su desarrollo, por lo que podría demorar gran tiempo en quedar listo, se decidió mover la fecha de entrega del segundo *release* a junio y la funcionalidad del “Hipotetizador Manual” al tercer *release*.

Figura 12.9: Cuarta versión del plan de release acordada con el cliente

12.4.5 Quinta versión

Ya entregado el segundo *release*, y en desarrollo del tercero, el cliente decidió eliminar el "Hipotetizador Manual" (luego de que el equipo ya había incurrido tiempo en él, prototipando y validando, esto puede verse más en detalle en el capítulo de "Gestión del proyecto"), debido a que el grupo del cliente se dio cuenta que no tenía definido el proceso de armado de equipos, y por tanto no podía establecer algo fijo que pudiera bajarse a tierra y convertirse en funcionalidad. Además de que dicho proceso estaba en plena evolución, haciendo que lo que fuera que se definiese, tuviera alta probabilidad de quedar obsoleto en el tiempo, porque el flujo ya no iba a ser así y por tanto no se iba a usar. Esto hizo que se tuviera que llevar un proceso profundo de relevamiento para definir más funcionalidades (que aportaran valor al sistema) para sopesar esta pérdida. En total se definieron 3 nuevas funcionalidades principales, junto con otras funcionalidades secundarias, también se decidió incorporar cambios de *feedback*, sumando 3 funcionalidades y algunos cambios pequeños.

Agregado de nueva funcionalidad

Siguiendo con el desarrollo del último *release*, si bien el "Hipotetizador Manual" había sido eliminado, después de varias conversaciones con el cliente, el equipo ve que igualmente el cliente necesitaba un lugar en donde pueda mover a los empleados libremente, sin generar cambios reales en la organización. Es así como surgió la funcionalidad de "Escenarios interactivos", que es un lienzo que tiene como base la conformación de empleados y equipos actual de la empresa y se pueden diagramar diferentes posibles cambios empresariales,

moviendo empleados de equipos, sin generar cambios reales en el sistema y pueden guardar estas ideas para verlas luego, con el fin de poder visualizar fácilmente cambios de negocios y dar lugar a una mejor toma de decisiones futura.

En la siguiente imagen se observa los dos grandes sucesos mencionados anteriormente, que conforman la versión final del plan de *releases*.

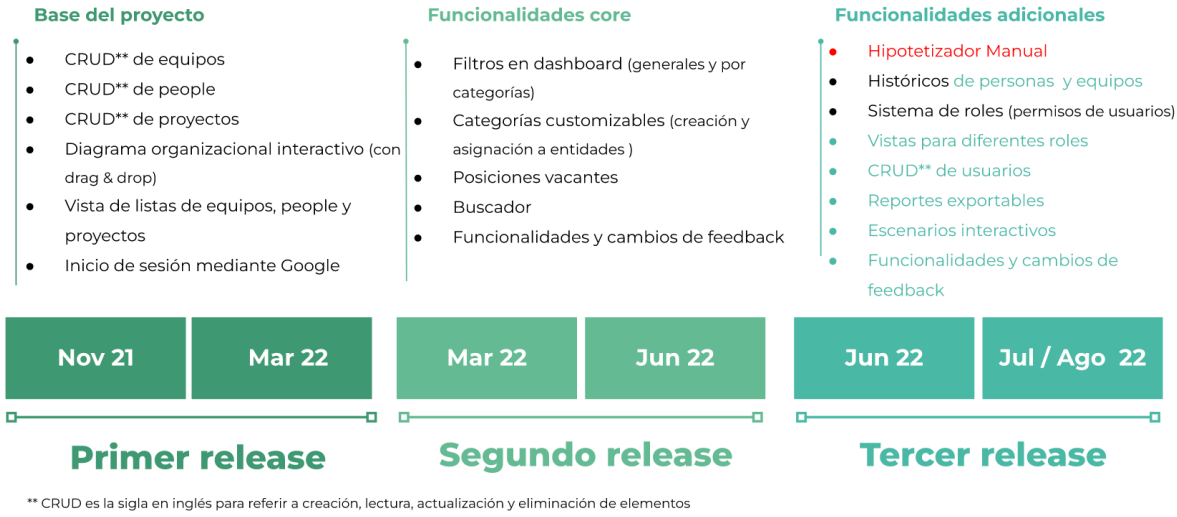


Figura 12.10: Quinta versión del plan de *release* acordada con el cliente

12.5 Registro de riesgos

12.5.1 Riesgos por *sprint*

En este anexo se listarán algunos de los riesgos considerados y que ocurrieron en los diferentes *sprints* a lo largo del proyecto. Como se mencionó en el capítulo de "Gestión del proyecto", en la sección de "Gestión de riesgos", antes de iniciar cada *sprint* se analizaban los posibles riesgos que podrían ocurrir en ese período, se anotaban los nuevos y se actualizaban, si correspondía, los valores de los ya conocidos. Esto permitió generar y mantener actualizado el registro de riesgos en el tiempo.

En la lista, se encuentran los riesgos con su ID, tipo (familia a la que corresponde), una breve descripción del riesgo (mencionando en algunos casos su principal consecuencia), y los valores de probabilidad e impacto que se habían considerado en su análisis, previo a comenzar el *sprint* (hubo casos en que estos valores fueron distintos cuando se manifestaron). Con el fin de sintetizar la lista y hacer más ágil la lectura se omitieron las respuestas, pero cabe destacar que estas en la mayoría de los casos fueron acciones cuyo objetivo era intentar minimizar el impacto o probabilidad del riesgo o intentar evitarlo.

Para facilitar la lectura y evitar hacer más larga la lista, en la mayoría de los casos se omitirán de la lista los riesgos que estuvieron presentes en todos los *sprints*, salvo que sean el único riesgo del *sprint* o su categorización sea alta y sea relevante mencionarlo. Estos riesgos tendrán lugar al final de la presente sección, en donde se mostrará un pequeño resumen de su evolución a lo largo del tiempo. Sin embargo, no debe olvidarse que estos siempre estuvieron presentes, impactando en mayor o menor medida en el transcurso del proyecto.

12.5.1.1 Riesgos *Sprint* 1 - 22/11/2021

ID	Tipo	Descripción	Probabilidad	Impacto	Magnitud
R6	Académico	Entregas finales de cursos de facultad de Keshet y Evgeny. Puede impactar sobre el rendimiento de los integrantes y la cantidad de esfuerzo dedicado al proyecto.	Bastante probable	Moderado	Riesgo medio

12.5.1.2 Riesgos *Sprint* 2 - 06/12/2021

ID	Tipo	Descripción	Probabilidad	Impacto	Magnitud
R6	Académico	Entregas finales y parciales de cursos de facultad de Keshet, Patrick y Valentina. Puede impactar sobre el rendimiento de los integrantes y la cantidad de esfuerzo dedicado al proyecto.	Muy probable	Alto	Riesgo alto
R7	Académico	Armado y presentación de revisión uno. Puede afectar el esfuerzo a dedicar en las otras áreas del proyecto.	Bastante probable	Moderado	Riesgo medio

12.5.1.3 Riesgos *Sprint* 3 - 20/12/2021

ID	Tipo	Descripción	Probabilidad	Impacto	Magnitud
R8	Social	Tiempo de fiestas de fin de año, los integrantes se quieren tomar un tiempo para pasar con sus familias. Por lo que se reducirá en general el esfuerzo dedicado de los mismos.	Muy probable	Alto	Riesgo alto

12.5.1.4 Riesgos *Sprint* 4 - 03/01/2022

ID	Tipo	Descripción	Probabilidad	Impacto	Magnitud
R3	Comunicación	Los cuatro integrantes del equipo del cliente, con los que	Muy probable	Alto	Riesgo alto

		se tenía comunicación en ese momento, estaban de licencia, y por lo tanto no iban a tomar reuniones con el equipo durante ese tiempo.			

12.5.1.5 Riesgos *Sprint* 5 - 17/01/2022

ID	Tipo	Descripción	Probabilidad	Impacto	Magnitud
R1	Técnico	Se tenían desconocimientos de tecnologías en dos grandes implementaciones a realizar. Estos eran: los movimientos de *Drag&Drop* en el tablero (que conllevaba además consideraciones de lógica y comunicaciones con el servidor). Y la reestructuración de todos los modelos de bases de datos para utilizar las relaciones (mediante *annotations*) provistas por TypeORM, lo que llevaría a un pienso para definir las relaciones entre las entidades, entender cómo funciona esa funcionalidad de la herramienta y aplicar los cambios.	Muy probable	Moderado	Riesgo alto
R12	Externo	Se rompió el monitor de Evgeny (siendo esta su pantalla principal por tener un computador de escritorio).	Poco probable	Alto	Riesgo medio

12.5.1.6 Riesgos *Sprint* 6 - 31/01/2022

ID	Tipo	Descripción	Probabilidad	Impacto	Magnitud
R1	Técnico	Creación y configuración del entorno de desarrollo en AWS. Creación y configuración del *pipeline* de bitbucket para la integración continua. Procesos los cuales no se contaba con experiencia ni conocimientos (en cuanto a AWS si bien se había visto algo de forma académica, no todos eran los mismos servicios).	Muy probable	Alto	Riesgo alto

12.5.1.7 Riesgos *Sprint* 7 - 14/02/2022

ID	Tipo	Descripción	Probabilidad	Impacto	Magnitud
R8	Social	Evgeny se va para afuera, algunos días del *sprint*. Lo que puede bajar el rendimiento o dedicación al proyecto.	Muy probable	Moderado	Riesgo alto
R10	Salud	Patrick se encuentra realizando suplencia de jefe de equipo en su trabajo, por lo que a veces puede quedarse más tiempo de su horario, afecta su rendimiento general y provoca agotamiento.	Muy probable	Moderado	Riesgo alto
R13	Externo	Error en *eslint* que provocaba fallas en el *build* de *frontend*, haciendo que por un tiempo no se pudieran impactar los cambios *mergeados* en la instancia de AWS ya que el pipeline no podía	Poco probable	Alto	Riesgo medio

		completar su camino y deployar la versión.			
R9	Gestión	Se detecta que todavía quedan varias tareas por hacer y por tanto es muy probable que no se llegue a tiempo con la fecha estimada de entrega del primer *release*.	Muy probable	Moderado	Riesgo alto

12.5.1.8 Riesgos *Sprint* 8 - 28/02/2022

ID	Tipo	Descripción	Probabilidad	Impacto	Magnitud
R8	Social	Vienen los padres de Keshet de visita desde Israel por dos semanas, y se va de vacaciones con ellos al interior. Por lo que no va a estar disponible (o muy poca disponibilidad) para el proyecto o reuniones.	Muy probable	Moderado	Riesgo alto
R2	Técnico	El equipo de *backend* va a realizar algunas historias de *frontend* porque se necesitan más recursos. Realizaron una breve capacitación, pero no tienen experiencia o mucho conocimiento con esta tecnología.	Muy probable	Moderado	Riesgo alto
R4	Comunicación	El cliente había solicitado varios cambios en la *review* y se tenían que realizar y validar diseños. El equipo necesita entender bien los procesos para poder realizar correctamente	Muy probable	Alto	Riesgo alto

		los cambios y diseños. Además de que se acerca el primer *release*.			

12.5.1.9 Riesgos *Sprint* 9 - 14/03/2022

ID	Tipo	Descripción	Probabilidad	Impacto	Magnitud
R10	Salud	Patrick se encuentra realizando suplencia de jefe de equipo en su trabajo, por lo que a veces puede quedarse más tiempo de su horario, afecta su rendimiento general y tiempo a dedicar al proyecto.	Muy probable	Alto	Riesgo alto
R8	Social	El padre de Patrick viene de Brasil a visitarlo, se toma unos días para estar con él, y luego se va a Artigas (su ciudad natal) para votar (debido a una votación obligatoria del país). Por lo que estará menos disponible para el proyecto.	Muy probable	Alto	Riesgo alto
R7	Académico	Se debe empezar con el armado del informe de avance. Esto puede afectar en el tiempo dedicado a otras áreas del proyecto.	Probable	Moderado	Riesgo medio
R14	Salud	Keshet con COVID-19, lo que provoca grandes malestares y afecta en gran medida su rendimiento.	Probable	Alto	Riesgo medio

12.5.1.10 Riesgos *Sprint* 10 - 28/03/2022

ID	Tipo	Descripción	Probabilidad	Impacto	Magnitud
R7	Académico	Se debe preparar el informe de avance y empezar a preparar la presentación para segunda revisión.	Probable	Moderado	Riesgo medio
R14	Salud	Keshet con COVID-19, la recuperación está llevando más tiempo del esperado, siguen los síntomas y afecta al rendimiento.	Bastante probable	Alto	Riesgo medio
R8	Social	Patrick se encuentra por unos días en Artigas (que tuvo que ir por la votación), allí no tiene muchos elementos para poder trabajar de manera eficiente, lo que dificulta en gran medida sus tareas. Luego debe volver en ómnibus, que es un viaje largo y agotador. Además, con los casos de COVID-19 activos implica un riesgo sanitario no menor.	Bastante probable	Alto	Riesgo alto
R15	Salud	Patrick y Valentina deben darse la tercera dosis de la vacuna del COVID-19 con Pfizer (que antes se habían dado Sinovac). Esto puede ocasionar malestar y decaimiento, pudiendo provocar un bajo rendimiento en las tareas.	Probable	Alto	Riesgo medio
R1	Técnico	Se debe crear y establecer la instancia de producción en la nube y	Muy probable	Muy alto	Riesgo alto

		configurar el *pipeline* para realizar el *deploy* automático en la rama *master* a este ambiente. Si bien se había realizado, hace un tiempo, para la instancia de desarrollo este procedimiento era un poco distinto y no se contaba con el conocimiento. El no poder realizarlo o que llevara mucho más tiempo del previsto repercutía en los tiempos en los que el cliente podía acceder y tener una instancia privada para su uso.			

12.5.1.11 Riesgos *Sprint* 11 - 11/04/2022

ID	Tipo	Descripción	Probabilidad	Impacto	Magnitud
R7	Académico	Se debe terminar el informe de avance y preparar la presentación para la segunda revisión.	Muy probable	Moderado	Riesgo alto
R8	Social	Evgeny se va unos días para afuera, y Keshet se va para Atlántida unos días, por lo que estarán ausentes hacia el proyecto en ese tiempo.	Muy probable	Moderado	Riesgo alto
R3	Comunicación	Juan Pablo, Nicolás y varias personas del área de People están de licencia la primera semana del *sprint*, y luego a fines de la segunda semana Juan Pablo (PO) se iría de viaje de negocios. Dificultando la	Bastante probable	Alto	Riesgo alto

		realización de reuniones necesarias de requerimientos.			

12.5.1.12 Riesgos *Sprint* 12 - 25/04/2022

ID	Tipo	Descripción	Probabilidad	Impacto	Magnitud
R7	Académico	Se debe practicar y tener la segunda revisión. Lo que puede afectar en los tiempos dedicados al resto del proyecto.	Muy probable	Alto	Riesgo alto
R2	Técnico	Hay demasiadas historias de *frontend* para realizar, los integrantes de *backend* todavía no se sienten seguros en esta área y un integrante del equipo de *frontend* no está del todo disponible.	Muy probable	Moderado	Riesgo alto

12.5.1.13 Riesgos *Sprint* 13 - 09/05/2022

ID	Tipo	Descripción	Probabilidad	Impacto	Magnitud
R6	Académico	Keshet tiene parcial a fines de la primera semana del *sprint*, por lo que va a estar unos días dedicando menos esfuerzo hacia el proyecto.	Muy probable	Moderado	Riesgo alto
R16	Académico	Evgeny tiene prueba de Japonés, por lo que puede disminuir un poco el tiempo dedicado al proyecto.	Bastante probable	Bajo	Riesgo medio
R3	Comunicación	Se deben realizar los diseños y validar el hipotetizador, pero Sofía (directora de P&C) no se encuentra disponible hasta finales de mayo.	Bastante probable	Alto	Riesgo alto

		Por lo que se debe realizar este procedimiento con el resto del equipo del cliente, que en estos tiempos tienen las agendas muy apretadas. Dificultando en gran medida la posibilidad de conseguir reuniones.			
R11	Gestión	Patrick y Valentina tienen unos días de licencia del trabajo, por lo que pueden incorporar un poco más de esfuerzo hacia el proyecto.	Bastante probable	Moderado	Riesgo medio (**positivo**)
R9	Gestión	Debido a atrasos en las historias en los *sprints* anteriores, el equipo detecta que puede no llegar con todo lo previsto para la fecha de entrega de *release* planeada.	Bastante probable	Alto	Riesgo alto

12.5.1.14 Riesgos *Sprint* 14 - 23/05/2022

ID	Tipo	Descripción	Probabilidad	Impacto	Magnitud
R3	Comunicación	Sofía sigue de vacaciones y estará ausente hasta inicios de la segunda semana del *sprint*. Por lo que se debe seguir el proceso de elicitación y validación del hipotetizador sin ella. Además de que siguen en general muy apretadas las agendas del equipo	Bastante Probable	Alto	Riesgo alto

		del cliente, dificultando la comunicación.			
R8	Social	Valentina y Patrick estarán para afuera unos días, por lo que estarán ausentes en el proyecto ese tiempo.	Muy probable	Alto	Riesgo alto
R6	Académico	Evgeny tiene un parcial que si bien es fuera del *sprint*, es cerca del final de este. Puede que incurra tiempo para estudiar hacia el final del *sprint*, disminuyendo el tiempo dedicado al proyecto en ese lapso.	Bastante probable	Moderado	Riesgo medio
R2	Técnico	Más del 85% de las historias del *sprint* son de *frontend* y no son sencillas. Esto puede ocasionar retrasos por la falta de experiencia de los integrantes en esta área.	Bastante probable	Alto	Riesgo alto

12.5.1.15 Riesgos *Sprint* 15 - 06/06/2022

ID	Tipo	Descripción	Probabilidad	Impacto	Magnitud
R6	Académico	Evgeny tiene un parcial en la primera semana del *sprint*, por lo que va a dedicar un tiempo para la preparación y estudio del mismo.	Muy probable	Alto	Riesgo alto
R8	Social	Keshet se va unos días al interior del país por un casamiento. Va a estar ausente hacia el proyecto en ese tiempo.	Muy probable	Moderado	Riesgo alto

12.5.1.16 Riesgos *Sprint* 16 - 20/06/2022

ID	Tipo	Descripción	Probabilidad	Impacto	Magnitud
R6	Académico	Evgeny tiene entrega de obligatorio en facultad y Keshet tiene parcial. Ambos se tomarán tiempo para la preparación y estudios de los mismos. Disminuyendo el tiempo dedicado al proyecto.	Muy probable	Moderado	Riesgo alto
R3	Comunicación	Las agendas del equipo del cliente están muy difíciles de conseguir horarios y se debe realizar un gran esfuerzo de ingeniería de requerimientos para definir nuevos requerimientos y cambios en el plan de *release* debido a la eliminación del hipotetizador.	Muy probable	Alto	Riesgo alto
R11	Gestión	La segunda semana del *sprint* Valentina estará de licencia del trabajo, por lo que puede dedicar un mayor esfuerzo hacia el proyecto.	Bastante probable	Alto	Riesgo alto (**positivo**)

12.5.1.17 Riesgos *Sprint* 17 - 04/07/2022

ID	Tipo	Descripción	Probabilidad	Impacto	Magnitud
R6	Académico	Keshet tiene entrega de obligatorio en la facultad. Se tomará un tiempo para la preparación y estudios de este. Disminuyendo el tiempo dedicado al proyecto.	Muy probable	Moderado	Riesgo alto

12.5.1.18 Riesgos *Sprint* 18 - 18/07/2022

ID	Tipo	Descripción	Probabilidad	Impacto	Magnitud
R6	Académico	Keshet tiene entrega de obligatorio y defensa de materias de facultad. Evgeny tiene defensa de una materia de facultad. Ambos se tomarán tiempo para la preparación y estudios de los mismos. Disminuyendo el tiempo dedicado al proyecto.	Muy probable	Alto	Riesgo alto
R16	Académico	Evgeny tiene pruebas de japonés en el tiempo del *sprint*, dedicará tiempo para la preparación de estas. Disminuyendo el tiempo dedicado al proyecto.	Bastante probable	Alto	Riesgo alto
R7	Académico	Se debe armar, practicar y realizar la tercera revisión. Lo que probablemente lleve tiempo y puede afectar el esfuerzo a dedicar en las otras áreas del proyecto.	Muy probable	Alto	Riesgo alto
R11	Gestión	Patrick tiene la segunda semana del *sprint* libre por licencia en el trabajo. Puede brindar un mayor esfuerzo hacia el proyecto.	Bastante probable	Moderado	Riesgo medio (**positivo**)

12.5.1.19 Riesgos *Sprint* 19 - 18/07/2022

ID	Tipo	Descripción	Probabilidad	Impacto	Magnitud
R8	Social	Keshet se va por todo el *sprint* a Estados Unidos a diferentes zonas, lo que puede generar un impacto en las comunicaciones con el equipo por la diferencias horarias y también el tiempo dedicado al proyecto del integrante.	Muy probable	Alto	Riesgo alto
R3	Comunicación	Las agendas del equipo del cliente están bastante apretadas, incluso algunos integrantes se encuentran de licencia y se necesitan algunas reuniones para definir algunos arreglos y cambios finales.	Muy probable	Alto	Riesgo alto

12.5.2 Evolución de los riesgos con seguimiento en todos los *sprints*

Como se mencionó en el capítulo de "Gestión del proyecto" hubo riesgos que estuvieron presentes en todo el transcurso del proyecto, y en cada *sprint* se los analizaba y actualizaba su categorización, según el contexto del proyecto. Si bien en algunas ocasiones estos riesgos no se materializaron (o al menos no con los valores previstos), el hecho de mantenerlos actualizados y pronosticar sus posibles magnitudes permitió ir llevando un control y mantener al equipo alerta para tomar decisiones dependiendo del momento. En total fueron 8 riesgos a los que se le llevó un seguimiento de cerca.

Para mostrar estos riesgos de forma más clara, se separarán en familias, y dentro de estas se verá una gráfica con los diferentes valores que fueron tomando en sus debidas evaluaciones a lo largo del tiempo.

12.5.2.1 Técnicos:

Esta familia está relacionada a las tecnologías, codificación y todo lo relacionado a esos asuntos (es decir: desarrollo, *hosting*, repositorios, entre otros).

Siendo los principales riesgos monitoreados:

- **Desconocimiento de tecnologías** (R1): como ha sido mencionado en capítulos de este documento, el equipo no contaba con experiencia o grandes conocimientos de las tecnologías a utilizar.
- **Cargas de *features* de *frontend*** (R2): el sistema a realizar tenía alto grado de desarrollo de *frontend*, más que nada en algunas etapas, esto implicó riesgos en varias ocasiones debido a la falta de recursos dentro del equipo con experiencia en esta área.

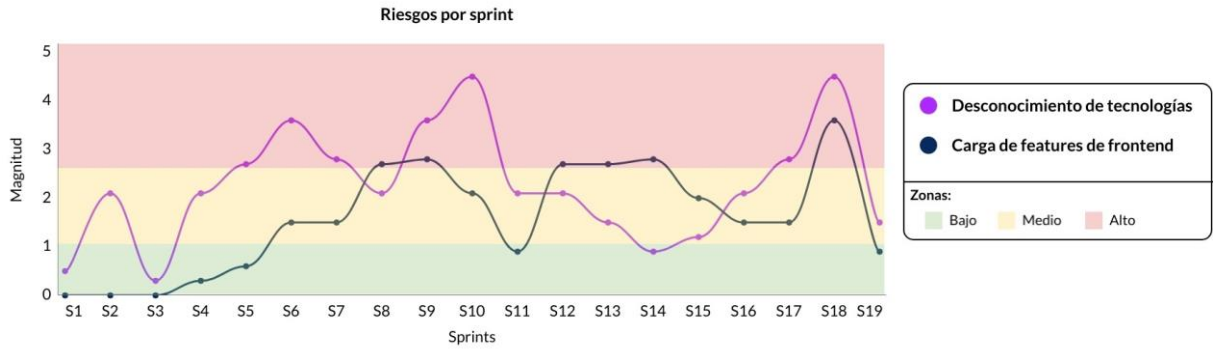


Figura 12.11: Evolución de los riesgos de la familia Técnicos

12.5.2.2 Comunicación:

Dentro de esta categoría se encuentran todos los riesgos asociados a comunicaciones, entendimiento de las solicitudes, procesos del cliente, y coordinación de reuniones (tanto con el cliente o con el equipo).

Siendo los principales riesgos monitoreados:

- **No conseguir reunirse con Qualabs** (R3): la comunicación con el equipo del cliente era importante, sin embargo, estos en general tenían las agendas muy apretadas, teniendo lapsos bastantes difíciles para conseguir reuniones. Para esto se debía tener un

alto grado de organización y anticipación. A lo largo del tiempo se fueron monitoreando las agendas y actualizando la categorización del riesgo, así como tomando medidas por si surgían contratiempos.

- **Entender los procesos de Qualabs** (R4): el objetivo era crear un sistema que se adapte a la empresa y permita realizar sus procesos de buena manera, por lo tanto, era necesario entenderlos. Así como también entender sus necesidades y los motivos detrás de los cambios pedidos para poder realizar las modificaciones correctamente e ir adaptando el sistema de una manera satisfactoria.

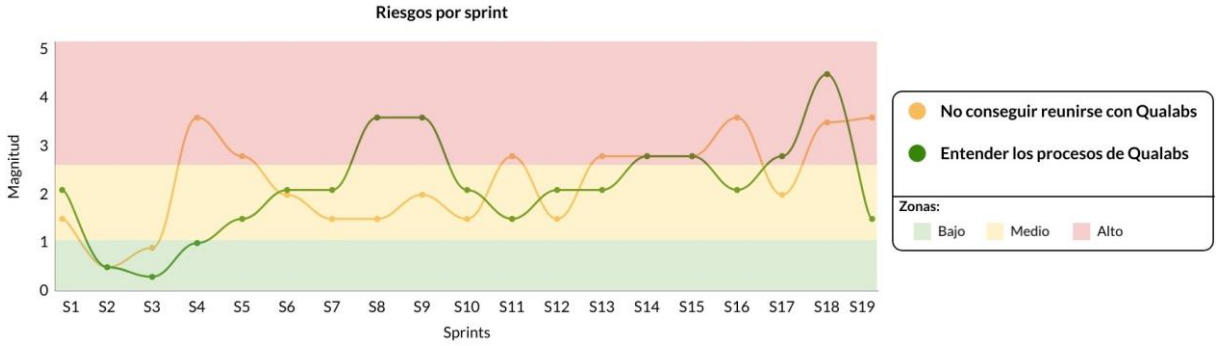


Figura 12.12: Evolución de los riesgos de la categoría Comunicación

12.5.2.3 Gestión:

Esta familia agrupa todos los riesgos relacionados con la gestión del proyecto en sí, alcance planificado, aplicaciones de metodologías, entre otros.

Los principales riesgos monitoreados fueron:

- **Estimación de tiempo mal realizada** (R5): este riesgo refiere tanto a errores de estimación tanto de historias como de fechas planeadas de plan de *release* u otras actividades. Se decidió llevar un control de esto porque en el inicio no fue fácil realizar buenas estimaciones de las historias debido al desconocimiento tanto de las tecnologías como del equipo en sí. Además, dado a que se tenía un tiempo acotado para el desarrollo, era importante que no hubiera grandes errores en esta área, ya que en algunos casos esto podría tener grandes consecuencias.

- **No llegar al alcance** (R9): este riesgo refiere tanto a no llegar con las funcionalidades previstas para el *release*, como el no poder llegar a realizar el alcance total planeado. Dado a que en algunos momentos hubo errores de estimaciones o funcionalidades que terminaron siendo más grandes de lo previsto en un inicio este riesgo era algo que se debía controlar para poder lograr tener un control correcto del proyecto y poder tomar acciones antes de que ocurran consecuencias. Llevar un control de esto permitía, además, el tener al cliente notificado y ser transparente con los tiempos, pudiendo informar con tiempo en caso de ver aproximarse un posible contratiempo.

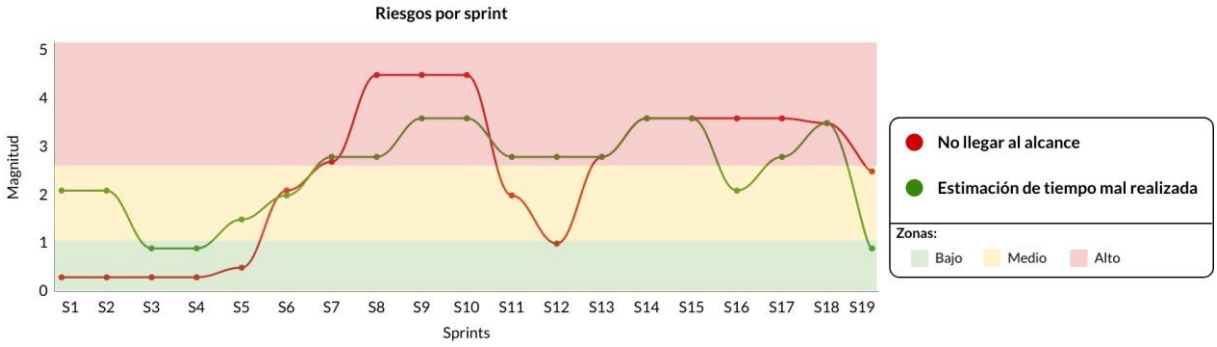


Figura 12.13: Evolución de los riesgos de la familia Gestión

12.5.2.4 Social:

Esta categoría engloba todo lo referente a compromisos, vacaciones, visitas de familiares, ambiente interno del equipo, entre otros.

Se vio que cada uno de estos riesgos resultaban en una no disponibilidad debido a una condición social, para su mejor entendimiento y control se decidió tomarlo como uno solo que podía generarse por diferentes motivos. De esta manera se tiene un único riesgo con nombre genérico que va tomando diferentes valores según sea instanciado en el tiempo.

El riesgo monitoreado es el siguiente:

- **Compromisos varios** (R8): Como se mencionó abarca todo lo que son vacaciones, visitas de familiares y compromisos sociales que hacen que los recursos no estén disponibles, o afectan el rendimiento de estos por un tiempo. Se decidió mantener el control de este riesgo dado que dentro del tiempo del proyecto se encontraban las fiestas de fin de año, período de vacaciones, y compromisos ya organizados de algunos integrantes. Por este motivo es que se decidió llevar el control de este riesgo, para poder

monitorear la cantidad de recursos disponibles en el tiempo o la posible implicancia del mismo, así poder prevenir grandes consecuencias y actuar acorde según el caso.

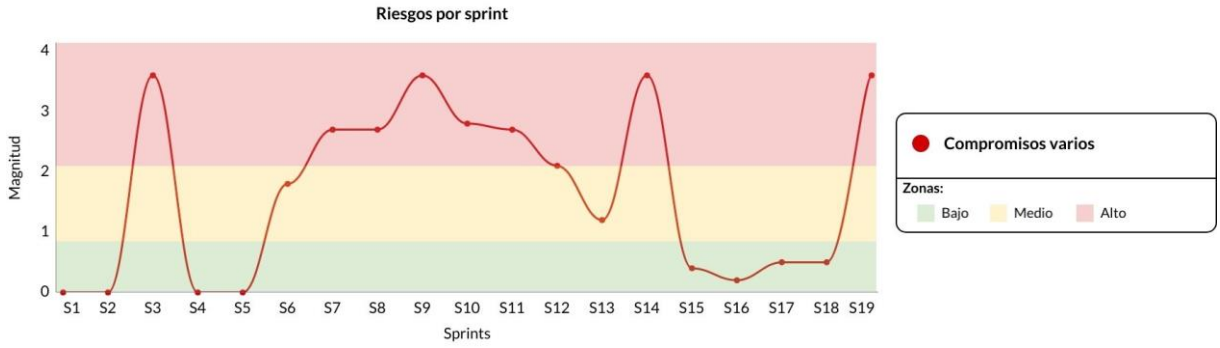


Figura 12.14: Evolución de los riesgos de la categoría Social

12.5.2.5 Académico:

Esta familia abarca todos los sucesos relacionados con las materias de facultad, entregables y eventos requeridos por la universidad referentes al proyecto, y eventos de cursos por fuera de la facultad.

El riesgo monitoreado fue el siguiente:

- **Obligaciones de materias de facultad** (R6): esto riesgo refiere a todas las actividades y obligaciones de materias de facultad, (parciales, obligatorios, entregas, clases en sí). Se decidió controlar este riesgo debido a que a lo largo del transcurso del proyecto los integrantes se encontraban cursando materias de facultad. Esto afectaba al hecho de la coordinación de horarios para reuniones entre el equipo o con el cliente. Además, era de vital importancia estar alertas y tener en cuenta el período de entregas y cierre de cursos, ya que en estos era muy probable que se disminuyera el esfuerzo dedicado al proyecto o el rendimiento en general de él o los integrantes en cuestión.

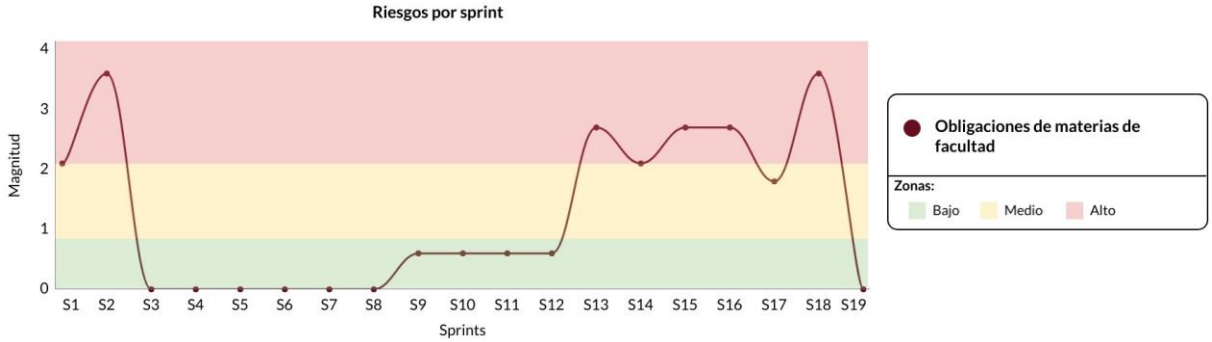


Figura 12.15: Evolución de los riesgos de la familia Académico

12.6 Heurísticas de Nielsen

En este anexo se mostrará cómo el equipo siguió las Heurísticas de Nielsen [69] al desarrollar el sistema realizando un análisis de cada heurística.

1. Visibilidad del estado del sistema

En todo momento, el usuario es informado sobre lo que sucede en el sistema. Para esto se le da un *feedback* constante en todas las páginas. En los formularios, por ejemplo, el usuario consta de mensajes de éxito o error al crear o editar una entidad. Gracias a esto, el usuario puede saber en qué estado quedó el sistema luego de enviar estos formularios. Los mensajes aparecen en forma inmediata y en un formato claro y sencillo de entender.

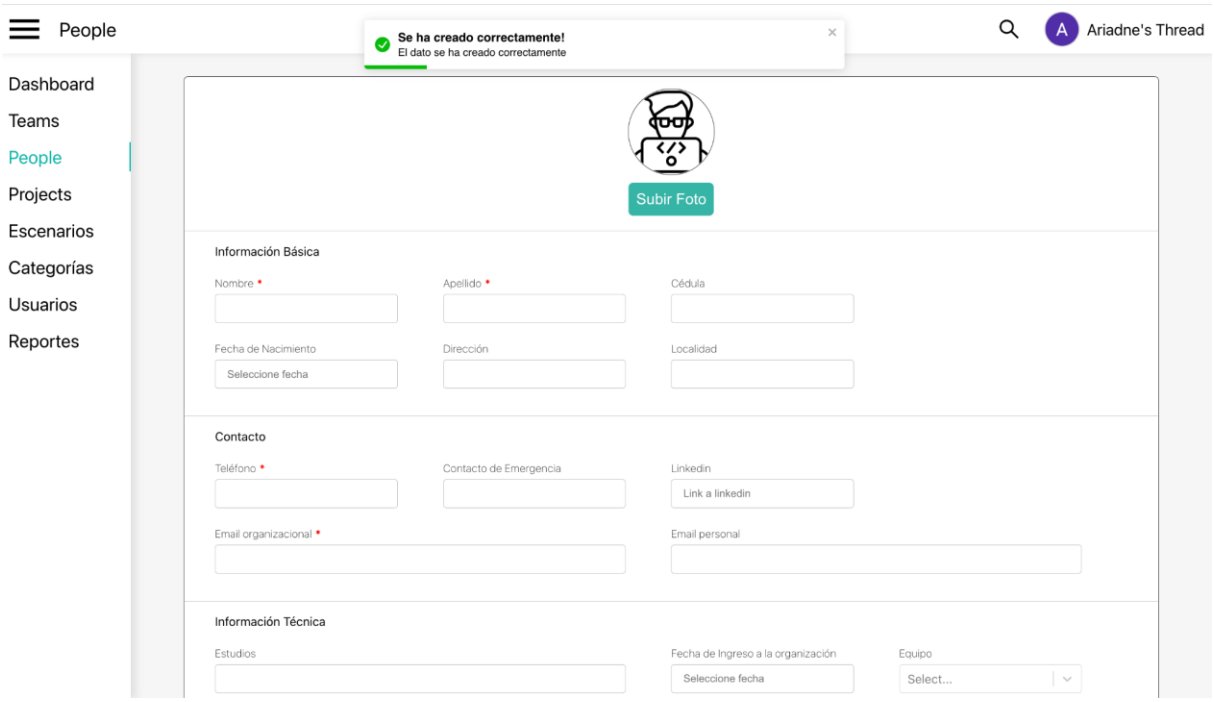


Figura 12.16: *Feedback* de creación correcta de una entidad

2. Coincidencia entre el sistema y el mundo real

El sistema emplea el mismo vocabulario de la empresa y del mundo real en todo momento. Varias veces, el cliente solicitó cambiar una palabra o una frase para que se ajuste a la cultura de la empresa. Un ejemplo fue cambiar la palabra Empleados a *People*, ya que es el término que utilizan en la empresa. El equipo siempre se mantuvo atento a las solicitudes del cliente e iba ajustando el sistema según éstas.

Además, los textos en el sistema son claros y sencillos de entender, teniendo clara coincidencia entre el sistema y el mundo real.

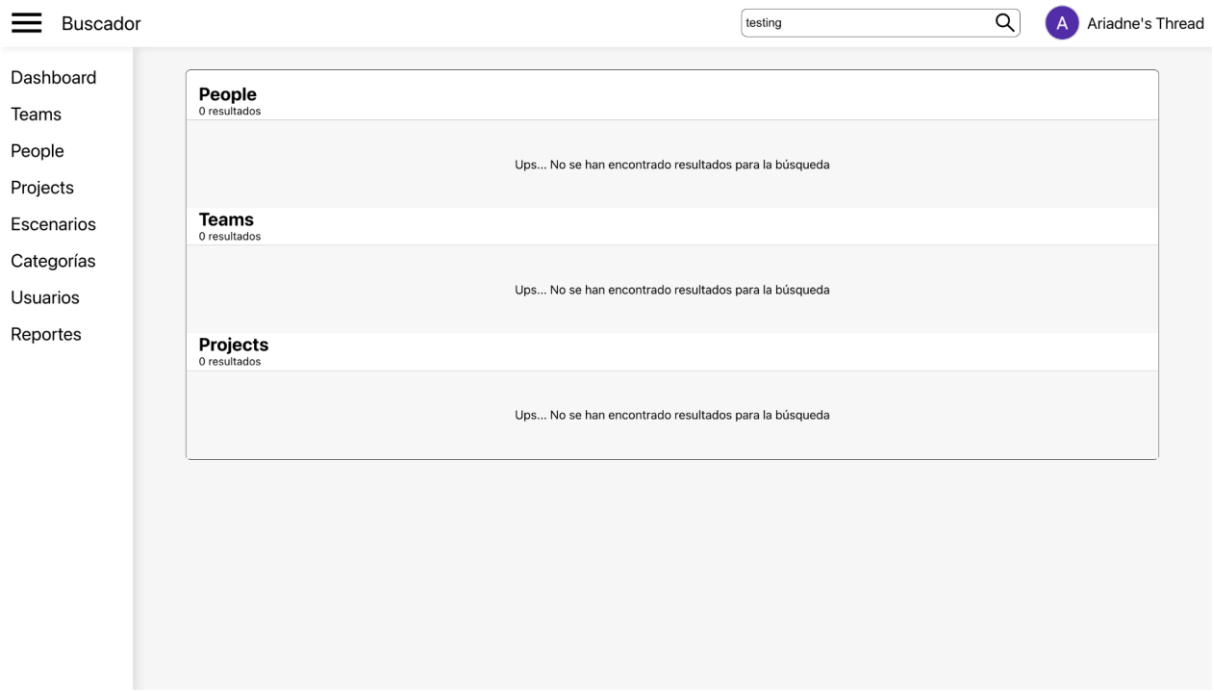


Figura 12.17: Texto claro cuando no se ha encontrado resultados

3. Dale al usuario el control y la libertad

El sistema no cuenta con la funcionalidad de deshacer, ya que no se vió como algo crítico en el uso del sistema. Pero, como es mencionado más adelante, el sistema cuenta con alertas de confirmación antes de realizar una acción.

4. Consistencia y estándares

El equipo siguió un estándar fijo durante todo el desarrollo del sistema. Los colores utilizados son consistentes al *branding* de la empresa. Además, como se puede ver en las imágenes, todos los componentes de interfaz gráfica son reutilizados a lo largo del sistema, manteniendo un estilo unificado y coherente. Por ejemplo, las listas utilizadas son iguales entre las diferentes páginas.

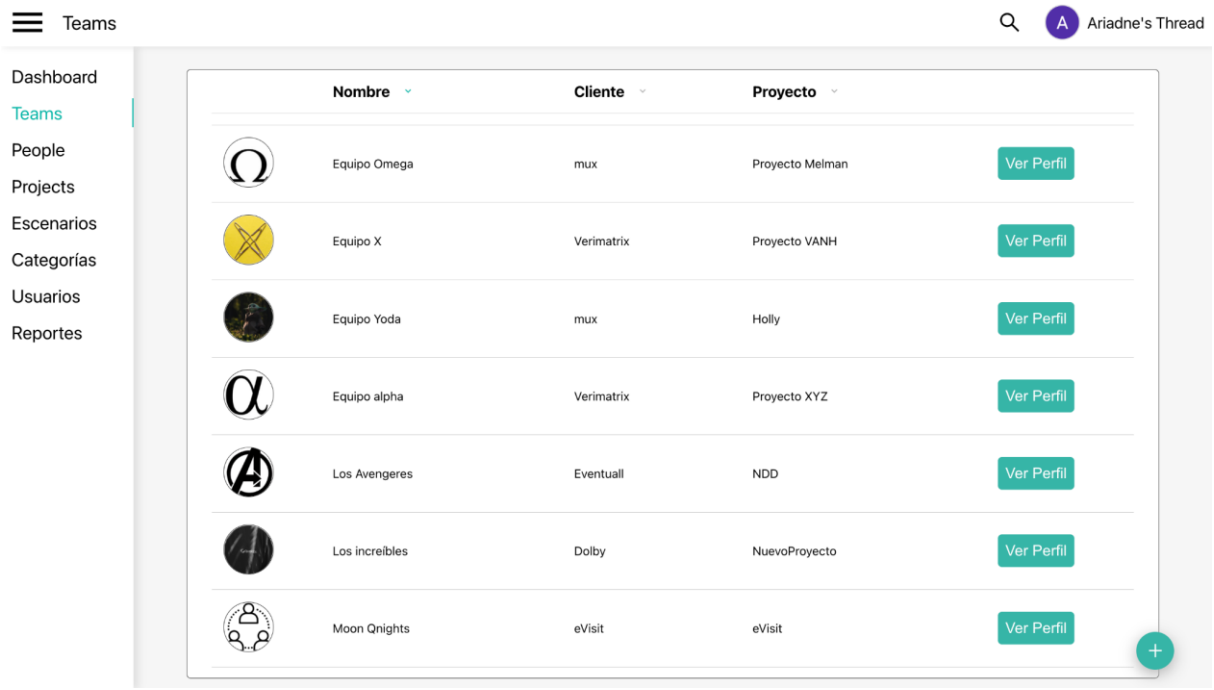


Figura 12.18: El diseño (colores, lenguaje, flujo de navegación) es coherente en todas las páginas

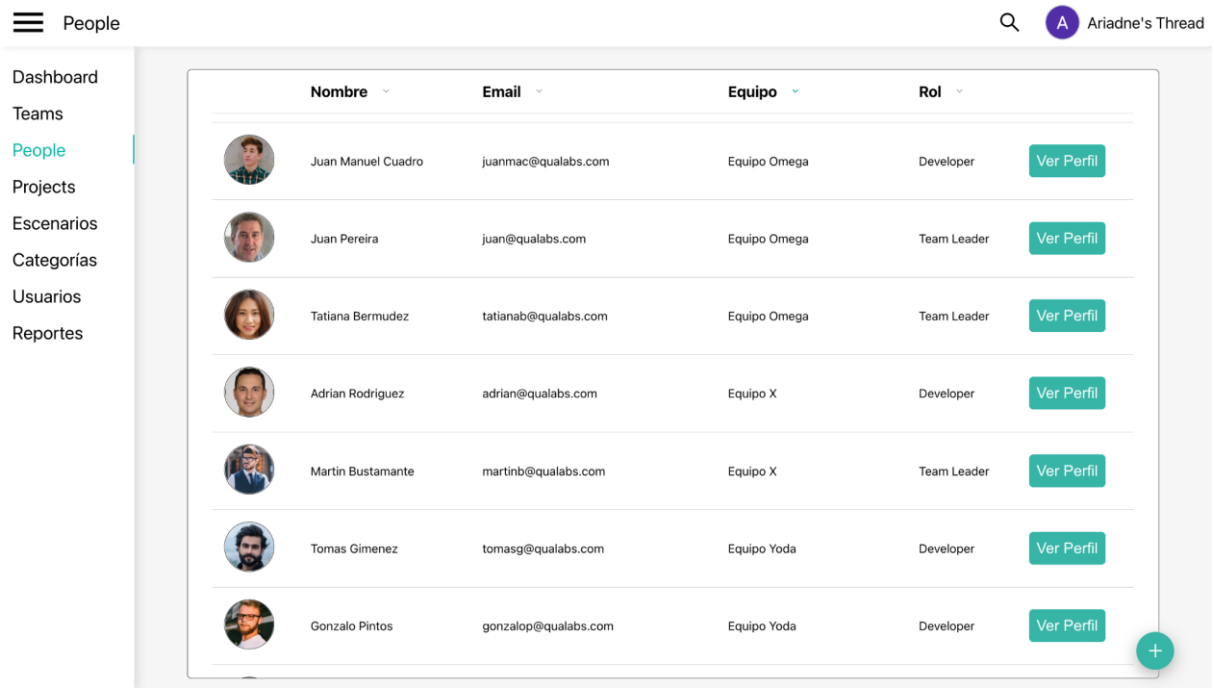


Figura 12.19: El diseño (colores, lenguaje, flujo de navegación) es coherente en todas las páginas

5. Prevención de errores

Antes de realizar una acción que afecte al sistema, se solicita confirmación. Esto fue implementado luego de notar que los usuarios pueden cometer errores involuntariamente, por lo cual se vio necesario tener una instancia de revisión.

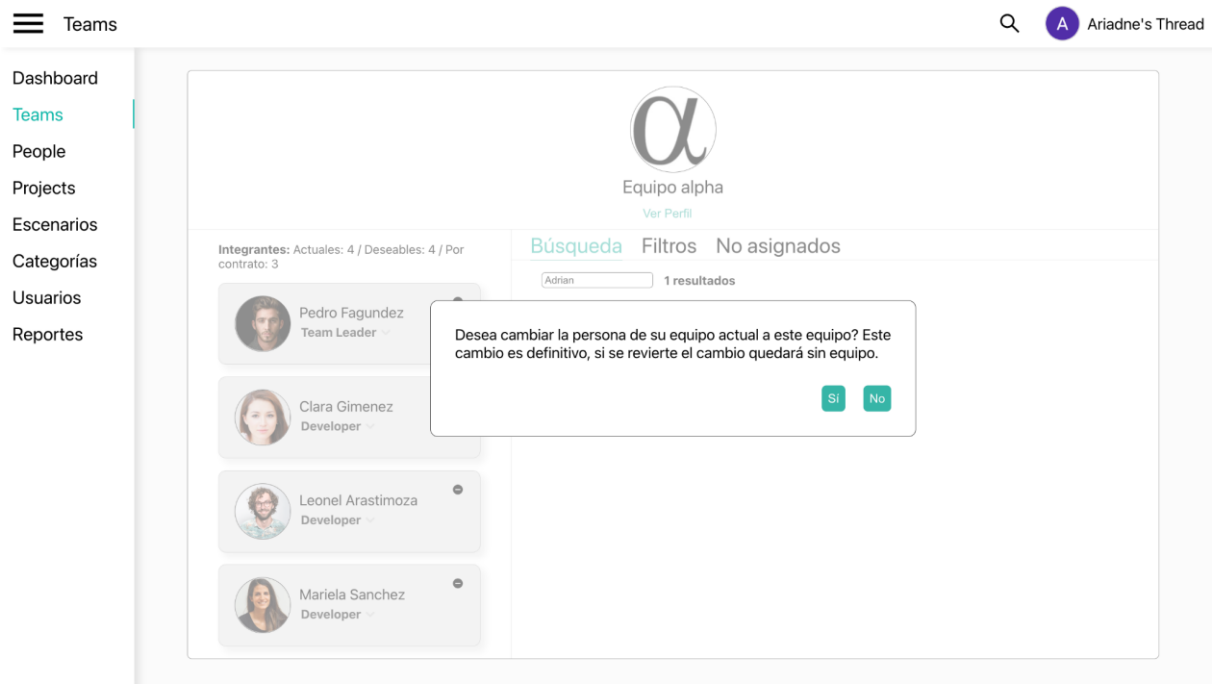


Figura 12.20: Mensajes de confirmación claros para que el cliente revise su acción

Otro claro ejemplo es en los formularios, donde se verifican los campos y se muestran mensajes de error, previniendo que se envíen datos incorrectos.

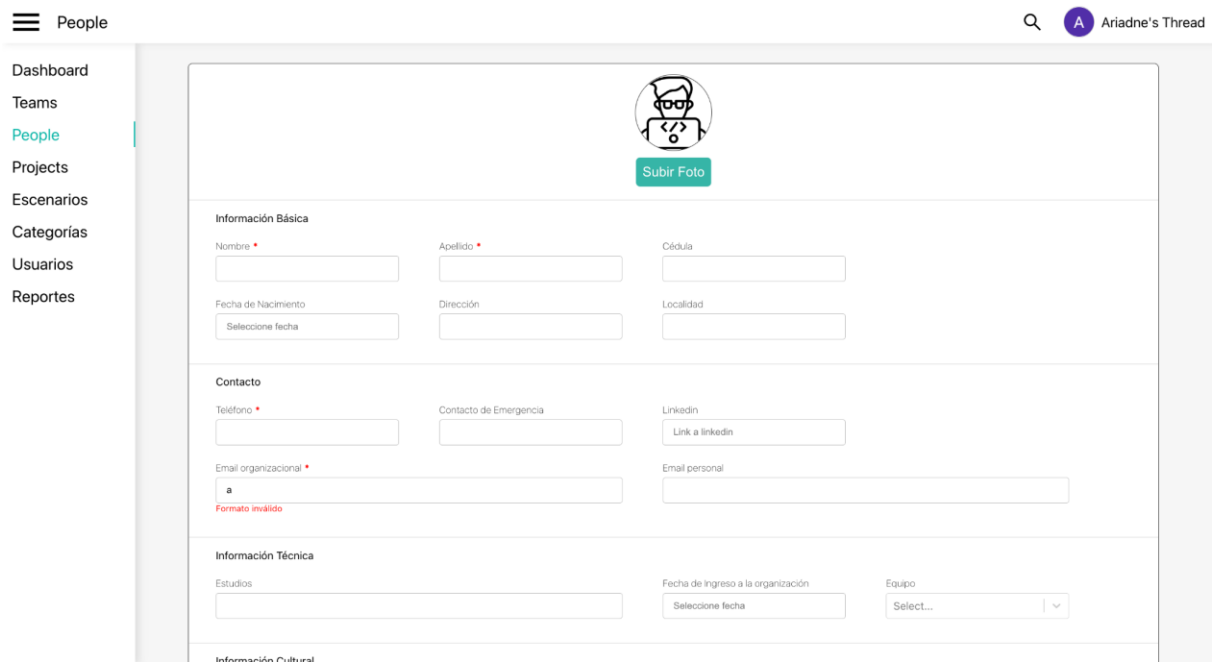


Figura 12.21: Mensaje de error en el campo al que corresponde

6. Reconocer en lugar de recordar

La pantalla principal del sistema (o *Dashboard*) es la que los usuarios van a utilizar de manera más frecuente. Debido a esto, se decidió acumular toda la información más importante para el usuario en esta pantalla, en vez de navegar entre muchas páginas para obtener la misma información. En un principio, la pantalla principal no constaba de toda esta información, y el cliente solicitó poder ver todo desde allí.

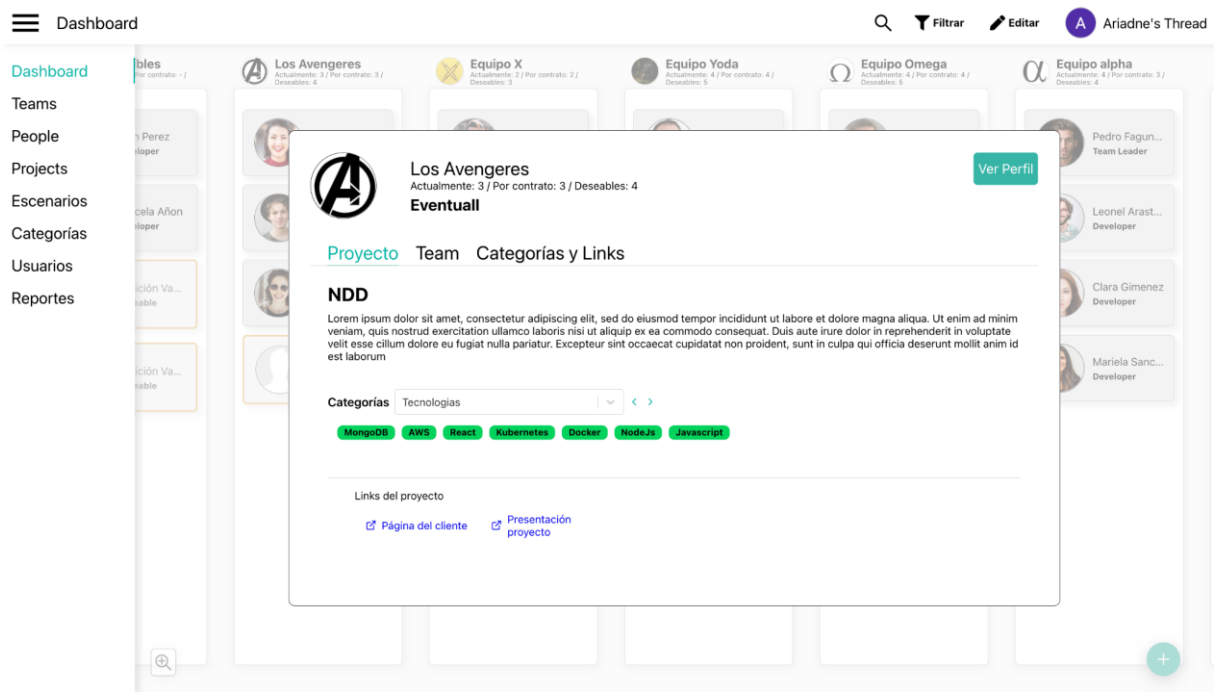


Figura 12.22: El usuario puede ver toda la información necesaria desde la pantalla principal

7. Flexibilidad y eficiencia de uso

La flexibilidad y eficiencia de uso fueron un factor importante para el cliente. Éste quería poder navegar de un lugar a otro sin tener que volver a otra página utilizando el menú lateral, sino a través de botones o links más directos. Es por esto, que cada vez que aparece información de una entidad del sistema, éste es clickeable y navegable, generando un *shortcut* rápido y flexible.

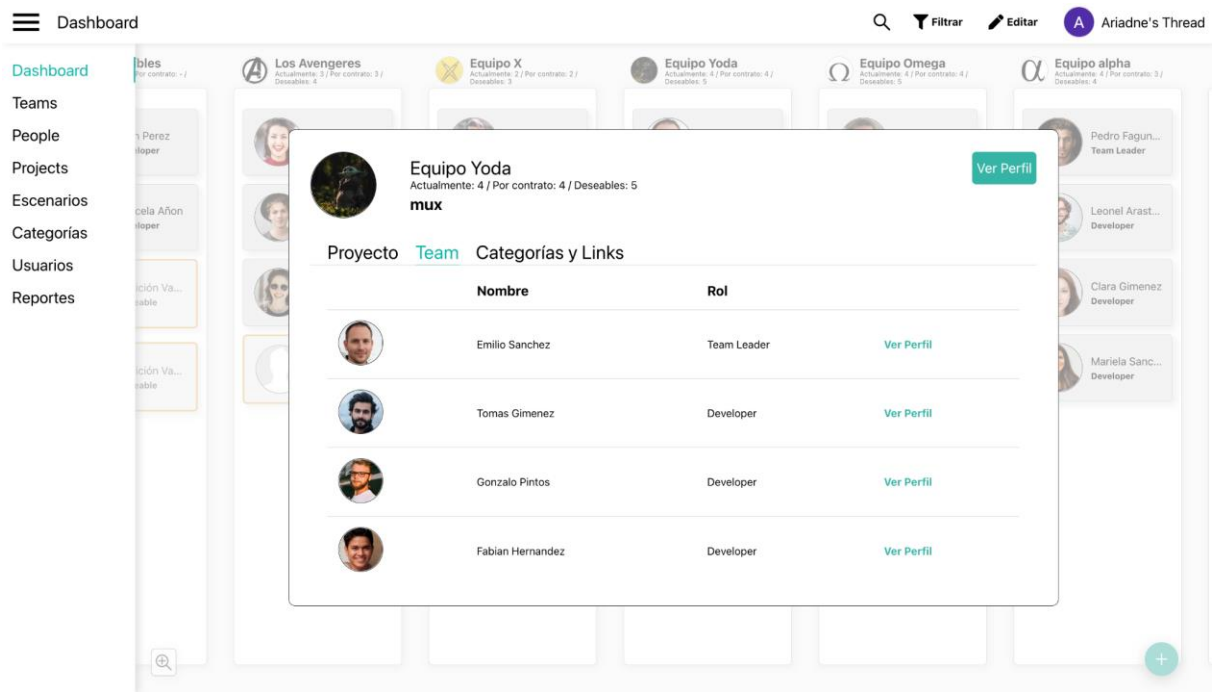


Figura 12.23: El usuario puede navegar fácilmente al perfil del equipo y al perfil de persona con tan solo un *click*

8. Estética y diseño minimalista

El diseño contiene sólo información relevante y necesaria. Por ejemplo, los botones incluyen un texto claro que explica lo que hacen.

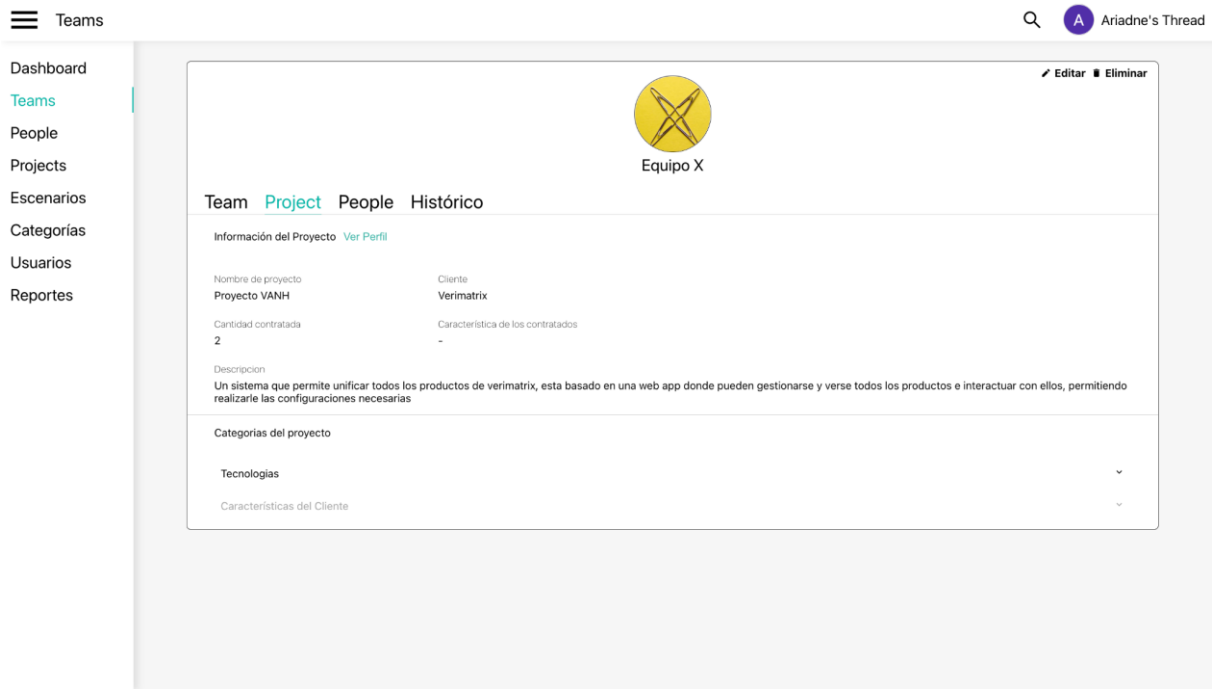


Figura 12.24: Se puede ver que todos los botones tienen un ícono y un texto para aclarar su acción

9. Ayuda al usuario a reconocer, diagnosticar y recuperarse de los errores

Tanto en el *backend* como en el *frontend*, se realizaron manejo de errores. En los formularios, por ejemplo, todos los campos con error contienen mensajes claros de error en rojo, para que el usuario lo identifique y entienda rápidamente y lo pueda arreglar.

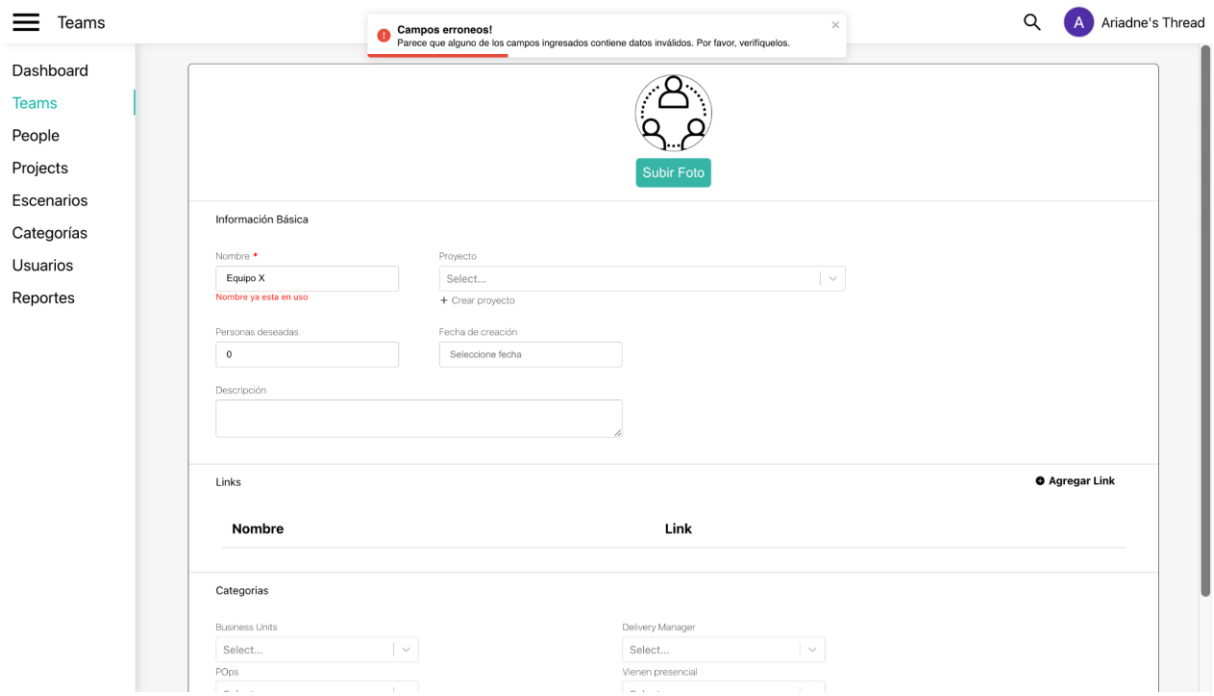


Figura 12.25: Mensaje de error y mensaje en el campo al que corresponde

10. Ayuda y documentación

El cliente no recibió un manual de uso del sistema, ya que no se vio necesario. Sin embargo, luego de cada *release*, el usuario recibió un listado de las nuevas funcionalidades detallando qué hace cada una.

12.7 *Tests* de usuarios

En el siguiente anexo se detallarán las pruebas que se realizaron con los usuarios. Se establecieron varias pruebas para poder analizar el comportamiento de los usuarios del sistema.

El objetivo de estas pruebas fue medir y observar:

- El tiempo que les llevó realizar cada tarea
- Si se logró terminar todas las tareas
- Dónde se encontraron frustraciones
- Preguntas y comentarios realizados

12.7.1 *Script*

1. Creación de proyecto
Objetivo de la tarea: Entender cómo se maneja el usuario al crear un proyecto en el sistema.
***Inputs*:** El usuario debe ubicarse en el *Dashboard* (página principal).
Terminada si: El usuario logra crear un proyecto con los datos correctos.
Instrucciones para el usuario: Surgió un nuevo proyecto de la empresa *Amazon* con el nombre *PrimeVideo*. El cliente quiere un equipo de 3 personas utilizando las tecnologías de Javascript, NodeJs, React, *MongoDB* y *hosting* en *AWS*.

2. Creación de equipo
Objetivo de la tarea: Entender si el usuario sabe crear un equipo y linkearlo al proyecto.
***Inputs*:** El usuario puede partir de donde se encuentra luego de la tarea anterior.

Terminada si: El usuario logra crear un equipo con los datos correctos y linkearlo al proyecto anteriormente creado.

Instrucciones para el usuario:

Para el proyecto anteriormente mencionado se desea crear el equipo *Quavengers* que tenga inicialmente 4 personas deseadas, para que se pueda asegurar siempre cumplir las expectativas del cliente. Sabes también que el equipo pertenecerá a la BU3.

3. Asignación de persona por búsqueda

Objetivo de la tarea: Entender si el usuario sabe utilizar el buscador de gestión de equipo.

***Inputs*:** El usuario debe encontrarse en la pantalla de gestionar el equipo *Quavengers*.

Terminada si: Se agrega a Joaquín Pereira al equipo Quavengers

Instrucciones para el usuario:

Sabes que "Joaquín Pereira" se encuentra actualmente sin equipo y que éste sería un buen *fit* para ser *Team Leader* para este proyecto. ¡Agrégalo al equipo!

4. Asignación de persona por filtro

Objetivo de la tarea: Entender si el usuario sabe utilizar el filtro de gestión de equipo.

***Inputs*:** El usuario debe encontrarse en la pantalla de gestionar el equipo Quavengers.

Terminada si: Se agrega a una persona que sepa React al equipo.

Instrucciones para el usuario:

Tienes que agregar al equipo una persona que sepa la tecnología de React. Puedes agregar a la persona que desees.

5. Creación de persona y asignación a equipo
Objetivo de la tarea: Entender si el usuario sabe crear una persona y asignarla a un equipo.
***Inputs*:** El usuario puede partir de donde se encuentra luego de la tarea anterior.
Terminada si: Se crea a Juan Esteban Hernández y se lo asigna a *Quavengers.*
Instrucciones para el usuario: Sabes que llega una nueva persona a la empresa llamada Juan Esteban Hernandez y que ya está destinado a este proyecto. Sabes que su número de teléfono es 099099099 y que su mail organizacional es juanh@qualabs.com. Añádelo al sistema y asígnalo al equipo.

6. Filtrar por categoría
Objetivo de la tarea: Entender si el usuario sabe utilizar el filtrado por categoría del *Dashboard*.
***Inputs*:** El usuario debe ubicarse en el *Dashboard* (página principal).
Terminada si: Llegar a una vista que se muestren todos los equipos que pertenecen a la BU3.
Instrucciones para el usuario: Quieres ver qué equipos conforman la BU actual del equipo *Quavengers*. Ve al *Dashboard* y filtra por la BU3 para que aparezcan solo sus equipos.

7. Filtrar por equipos incompletos por contrato
Objetivo de la tarea: Entender si el usuario sabe utilizar el filtro general del *Dashboard*.
***Inputs*:** El usuario debe ubicarse en el *Dashboard* (página principal).

Terminada si: Llegar a una vista que se muestren todos los equipos que tienen puestos vacantes por contrato.
Instrucciones para el usuario: Se está pensando en incorporar más personas a la empresa, y el área de negocios quiere tener una idea de qué equipos necesitan cubrir posiciones requeridas por contrato. Para esto, utiliza el filtro del *Dashboard* para poder visualizar los equipos.

12.7.2 Ejecución de pruebas

12.7.2.1 Usuario 1

Es un usuario con poca alfabetización digital, ha visto el sistema anteriormente pero no lo ha usado, ha participado en algunas demos y tiene poca noción de las primeras funcionalidades. Se podría decir que es un usuario inexperiente, con muy poco uso del sistema.

Tiempo total de la sesión: 15:40 minutos contando la introducción, lectura de pruebas y cierre con algunos comentarios.

	Prueba 1	Prueba 2	Prueba 3	Prueba 4	Prueba 5	Prueba 6	Prueba 7
Duración	4.46 min	1.25 min	28s	30s	3.05 min	25 s	22 s

Tabla 12.1: Tiempo que le tomó al usuario 1 realizar cada tarea

Pruebas no completadas: completó todas las pruebas.

Frustraciones: Le costó asignar una persona al equipo. Como es un campo que no está remarcado, fácilmente se puede pasar por alto. Además, se frustró con el Filtro de persona en la página de Equipo. Dijo que, al filtrar las personas, no se dio cuenta del botón de “Ver Más” y terminó viendo solamente las 4 primeras opciones que le aparecen en pantalla, en vez de ver todas.

Preguntas y comentarios del usuario:

Le gustó realizar la prueba. Comentó que algunas de las dificultades que presentó eran debidas a que no había usado el sistema anteriormente, y cree que luego de haber usado una vez el sistema se entendería mucho más rápido.

12.7.2.2 Usuario 2

Es un usuario con una alfabetización digital promedio. Tuvo una alta participación en las demos, por lo tanto estaba informado sobre los avances del desarrollo del sistema. Ha utilizado el sistema en varias ocasiones, realizando pruebas para familiarizarse con él, y hasta ingresó datos del sistema para poder empezar a utilizarlo.

Se podría decir que es un usuario experimentado del sistema.

Tiempo total de la sesión: 12:20 minutos aproximadamente, contando la introducción, lectura de pruebas y cierre con comentarios.

	Prueba 1	Prueba 2	Prueba 3	Prueba 4	Prueba 5	Prueba 6	Prueba 7
Duración	2.13 min	38 s	23 s	27.58 s	1.35 min	37.40 s	30.05 s

Tabla 12.2: Tiempo que le tomó al usuario 2 realizar cada tarea

Pruebas no completadas: completó todas las pruebas.

Frustraciones: ninguna.

Preguntas y comentarios del usuario: Le gustó la prueba y el sistema, se mostró contento usándolo.

12.7.2.3 Usuario 3

Es un usuario con una alfabetización digital promedio. Estuvo presente desde el inicio hasta la mitad del proyecto aproximadamente, siendo partícipe en demos y negociaciones, pero no estaba informado sobre los últimos avances del sistema. Tenía conocimientos básicos del

sistema, y hasta lo había utilizado, pero solamente en etapas tempranas del proyecto y no estaba al tanto de los últimos avances.

Se podría decir que es un usuario con conocimiento moderado del sistema; tiene idea de algunas de las funcionalidades básicas, pero no lo ha utilizado mucho.

Tiempo total de la sesión: 14 minutos aproximadamente contando introducción, lectura de pruebas y cierre con algunos comentarios.

	Prueba 1	Prueba 2	Prueba 3	Prueba 4	Prueba 5	Prueba 6	Prueba 7
Duración	2.09 min	48 s	2.17 min	1.10 min	1.20 min	15 s	22 s

Tabla 12.3: Tiempo que le tomó al usuario 3 realizar cada tarea

Pruebas no completadas: completó todas las pruebas.

Frustraciones: Le costó encontrar las personas que todavía no están asignados a un equipo en el *Dashboard*. También mencionó que en la creación de proyectos, no está clara la opción de crear un nuevo cliente mediante el selector.

Preguntas y comentarios del usuario: ninguno.

12.7.2.4 Usuario 4

Es un usuario con una alfabetización digital promedio. Estuvo presente desde el inicio hasta la mitad del proyecto aproximadamente, siendo partícipe en demos, pero no estaba informado sobre los últimos avances del sistema. Tenía conocimientos básicos del sistema, y hasta lo había utilizado, pero solamente llegó a utilizar las funcionalidades básicas, como registrar algunos usuarios reales, en etapas tempranas del proyecto.

Se podría decir que es un usuario con conocimiento moderado-alto del sistema, sobre todo de las funcionalidades iniciales y básicas del sistema, que fue las que utilizó. En cuanto a las nuevas funcionalidades y secciones no presenta experiencia.

Tiempo total de la sesión: Aproximadamente 10 minutos, contando introducción, lectura de las pruebas, ejecución y cierre con comentarios.

	Prueba 1	Prueba 2	Prueba 3	Prueba 4	Prueba 5	Prueba 6	Prueba 7
Duración	2.01 min	1.02 min	40 s	32 s	1.09 min	28 s	31 s

Tabla 12.4: Tiempo que le tomó al usuario 4 realizar cada tarea

Pruebas no completadas: completó todas las pruebas.

Frustraciones: El agregado de cliente mediante el selector le causó problemas, dado que no presionó *enter* dejando escrito el campo, por lo que no quedó creado. Luego de darse cuenta, presionó *enter* y pudo crearlo correctamente.

Preguntas y comentarios del usuario: Le gustó mucho el sistema. Mencionó que lo que más le costó fueron los filtros del *Dashboard*, dado que nunca había entrado ahí, pero que posiblemente no hubiera tenido problema si ya hubiera conocido el sistema.

12.8 Encuesta de satisfacción

A continuación se presentará la encuesta de satisfacción realizada para el cliente. Ésta fue hecha con el fin de poder medir la satisfacción del cliente con el producto. Las preguntas fueron escritas de tal manera que sean cuantificables, para luego poder obtener métricas y analizar los resultados. En la primera pregunta se utilizó la escala NPS (una puntuación del 1 al 10), y luego en el resto de las preguntas se empleó la escala Likert (una puntuación del 1 a 5).

12.8.1 Resultados de la encuesta

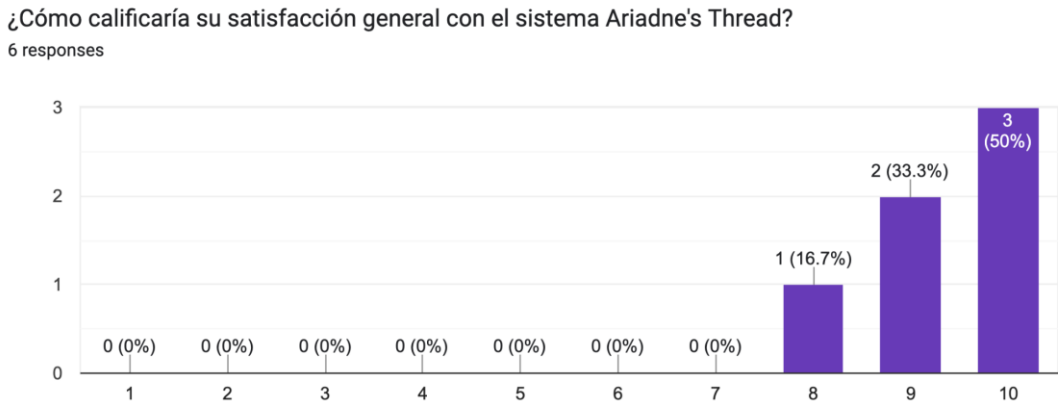


Figura 12.26: Resultado de la primera pregunta

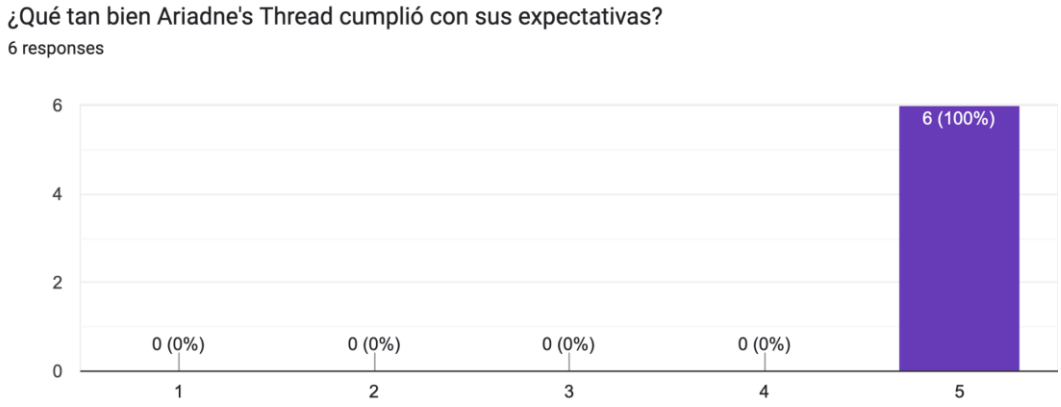


Figura 12.27: Resultado de la segunda pregunta

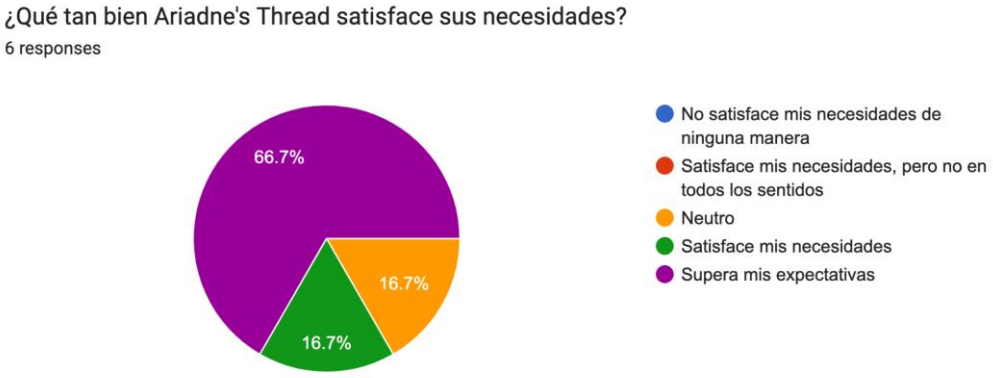


Figura 12.28: Resultado de la tercera pregunta

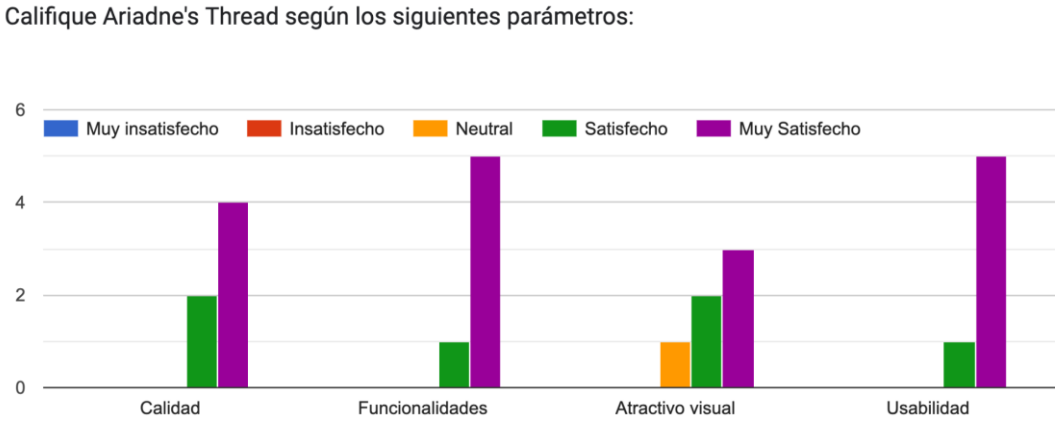


Figura 12.29: Resultado de la cuarta pregunta

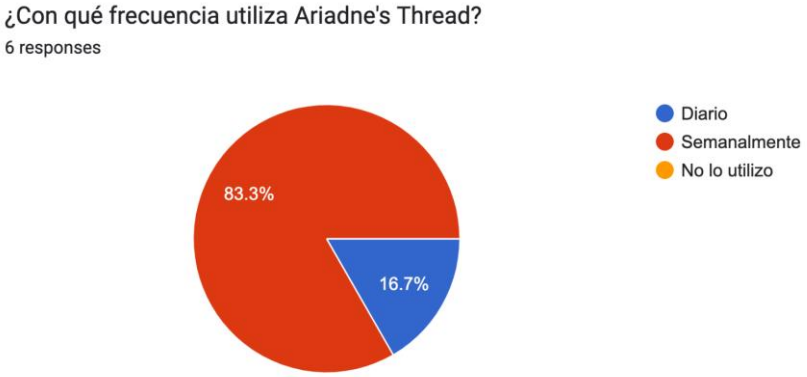


Figura 12.30: Resultado de la quinta pregunta

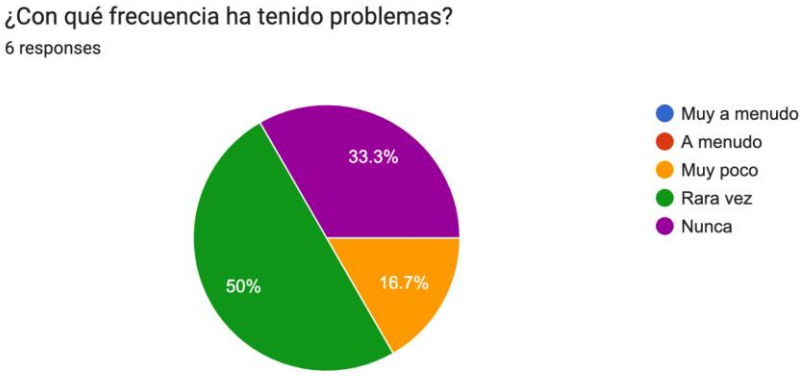


Figura 12.31: Resultado de la sexta pregunta

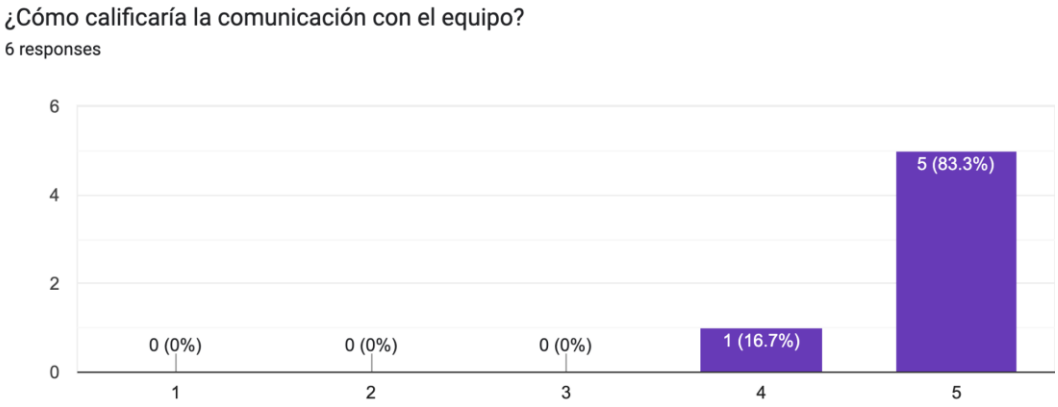


Figura 12.32: Resultado de la séptima pregunta

12.8.2 Cálculo de métricas

Para obtener las métricas de la encuesta se utilizó Excel para calcular de forma rápida el NPS y Likert.

12.8.2.1 NPS

En la figura 12.33 se muestra la hoja de Excel, en donde se pueden ver los promotores en verde (puntajes del 9 al 10), los pasivos en amarillo (del 7 al 8) y los detractores en rojo (del 1 al 6). Utilizando la fórmula mencionada en la sección de “Gestión de la calidad”, se pudo calcular el NPS basado en la primera pregunta de la encuesta obteniendo un resultado de 83.

RESULTADOS DE LA ENCUESTA: ¿Cómo calificaría su satisfacción general con el sistema Ariadne's Thread?	PUNTUACIÓN NPS	NÚMERO DE ENCUESTADOS	PORCENTAJE DE ENCUESTADOS	PROMOTORES (10, 9)	DETRACTORES (1-6)	PASIVOS (7, 8)	NPS:	83
9	10	3	50%	5	0	1		
10	9	2	33%	83%	0%	17%		
10	8	1	17%					
9	7	0	0%					
8	6	0	0%					
10	5	0	0%					
	4	0	0%					
	3	0	0%					
	2	0	0%					
	1	0	0%					
	Total de encuestado	6						

Figura 12.33: Hoja de Excel utilizada para calcular las métricas de la pregunta de tipo NPS

12.8.2.2 Likert

En las figuras 12.34 y 12.35 se muestra la hoja de Excel utilizada para calcular las métricas de las preguntas de tipo Likert [83]. Para cada pregunta, se multiplicó cada puntaje por la cantidad de respuestas correspondientes. Luego, se tomó la suma total de éstas y se dividió por la cantidad de encuestados.

Al terminar de calcular los datos para cada pregunta, se sumaron todos los puntajes de todas las preguntas para luego dividirlo por el número de preguntas, obteniendo finalmente un puntaje aproximado de 4.6 en la escala Likert.

Pregunta #	Pregunta		Escala	Respuestas	Total	
2	¿Qué tan bien Ariadne's Thread cumplió con sus expectativas?	1	1	0	0	
		2	2	0	0	Suma Total/Respuestas
		3	3	0	0	5
		4	4	0	0	
		5	5	6	30	
3	¿Qué tan bien Ariadne's Thread satisface sus necesidades?		Escala	Respuestas	Total	
		No satisface mis necesidades de ninguna manera	1	0	0	
		Satisface mis necesidades, pero no en todos los sentidos	2	0	0	Suma Total/Respuestas
		Neutro	3	1	3	4.5
		Satisface mis necesidades	4	1	4	
		Supera mis expectativas	5	4	20	
4	Califique Ariadne's Thread según los siguientes parámetros:					
	Calidad		Escala	Respuestas	Total	
		Muy insatisfecho	1	0	0	
		Insatisfecho	2	0	0	Suma Total/Respuestas
		Neutral	3	0	0	4.666666667
		Satisfecho	4	2	8	
		Muy Satisfecho	5	4	20	
	Funcionalidades		Escala	Respuestas	Total	
		Muy insatisfecho	1	0	0	
		Insatisfecho	2	0	0	Suma Total/Respuestas
		Neutral	3	0	0	4.833333333
		Satisfecho	4	1	4	
		Muy Satisfecho	5	5	25	

Figura 12.34: Hoja de Excel utilizada para calcular las métricas de las preguntas de tipo Likert

Atractivo visual		**Escala**	**Respuestas**	**Total**		
	Muy insatisfecho	1	0	0		
	Insatisfecho	2	0	0		Suma Total/Respuestas
	Neutral	3	1	3		**4.333333333**
	Satisfecho	4	2	8		
	Muy Satisfecho	5	3	15		
Usabilidad		**Escala**	**Respuestas**	**Total**		
	Muy insatisfecho	1	0	0		
	Insatisfecho	2	0	0		Suma Total/Respuestas
	Neutral	3	0	0		**4.833333333**
	Satisfecho	4	1	4		
	Muy Satisfecho	5	5	25		
5 ¿Con qué frecuencia ha tenido problemas?		**Escala**	**Respuestas**	**Total**		
	Muy a menudo	1	0	0		
	A menudo	2	0	0		Suma Total/Respuestas
	Muy poco	3	1	3		**4.166666667**
	Rara vez	4	3	12		
	Nunca	5	2	10		
					Puntaje final:	**4.619047619**

Figura 12.35: Hoja de Excel utilizada para calcular las métricas de las preguntas de tipo Likert

12.9 Comentarios del cliente

En el siguiente anexo se mostrarán mensajes y comentarios recibidos por parte del cliente. Éstos fueron recibidos a lo largo del proyecto en reuniones, demos, intercambios de mensajes y preguntas abiertas en la encuesta.

Comentarios de las reuniones y demos

Durante las reuniones y demos, el equipo tomó notas de los comentarios realizados por el cliente. Algunos de estos fueron:

“¡Me encanta! Ya lo quiero empezar a usar, ¡Gracias!”

“Estoy copada.”

“No saben el laburo que estoy teniendo ahora con el mural.”

Comentarios de la encuesta

En la encuesta de satisfacción, se incluyeron tres preguntas abiertas en el cual el cliente pudo realizar comentarios y dar su opinión con respecto al producto, equipo y proceso. Estos se pueden encontrar en las imágenes 12.36, 12.37 y 12.38.

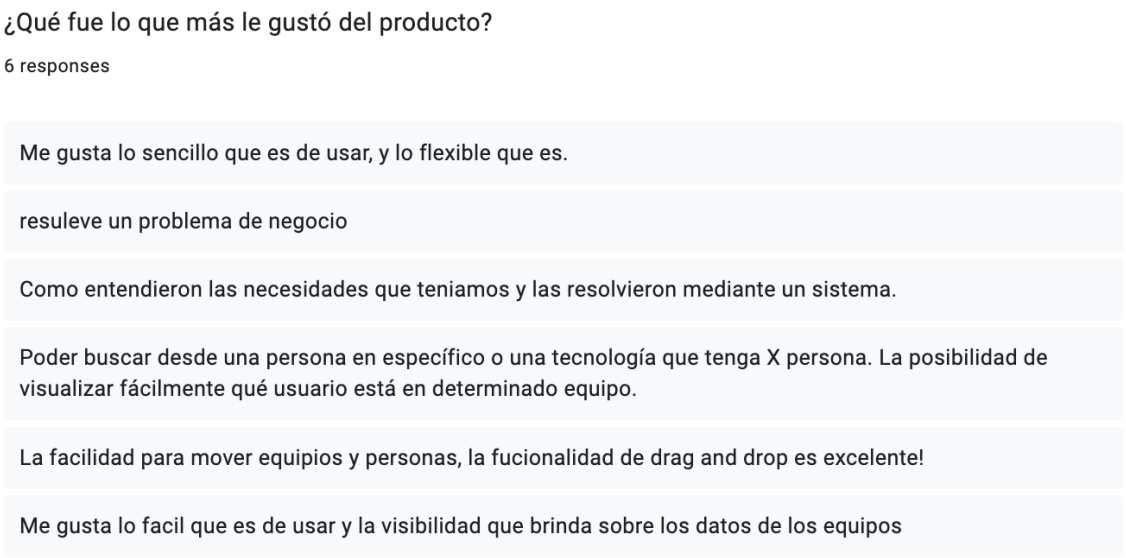


Figura 12.36: Comentarios recibidos en la encuesta de satisfacción

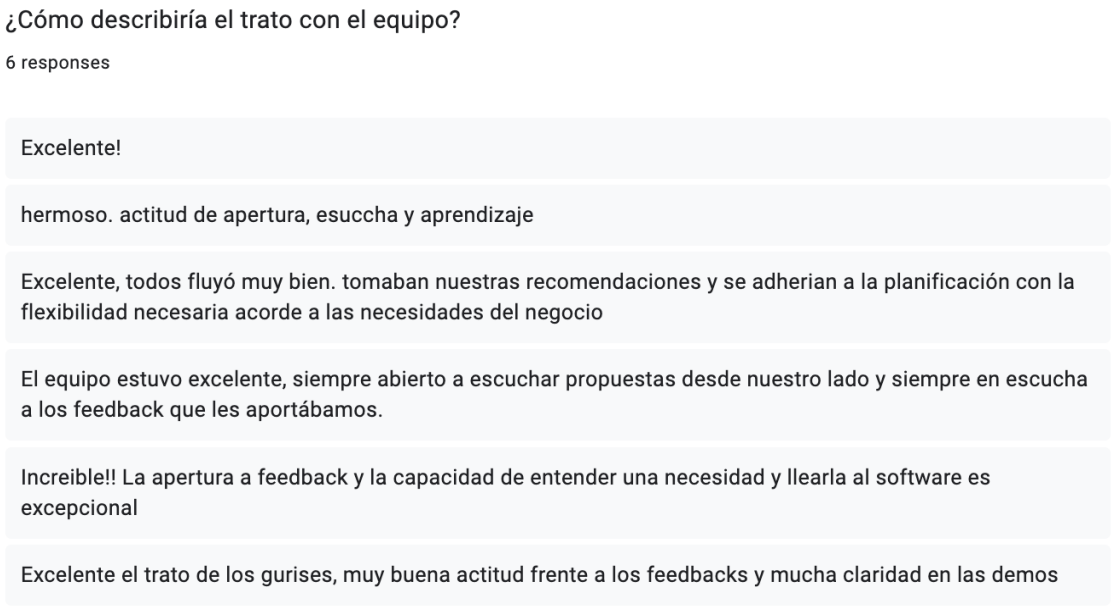


Figura 12.37: Comentarios recibidos en la encuesta de satisfacción

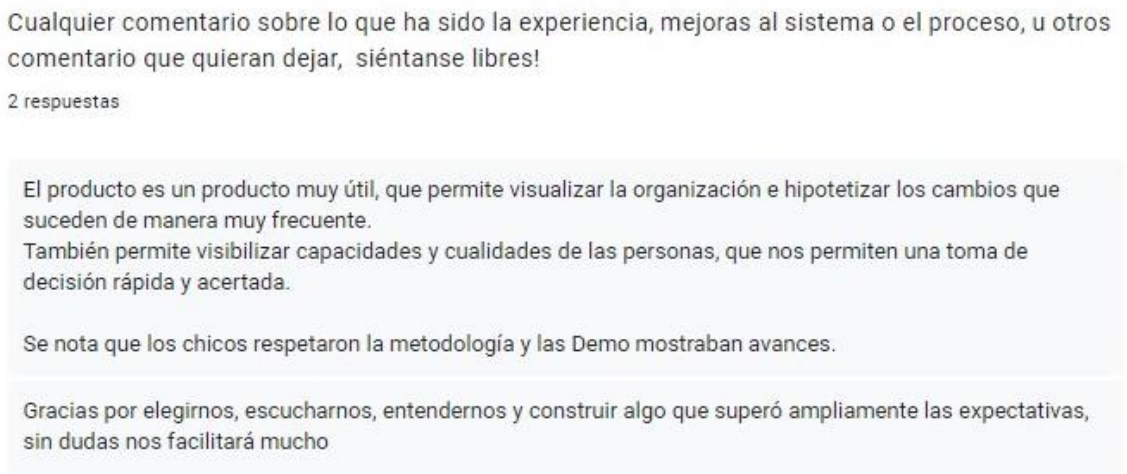


Figura 12.38: Comentarios recibidos en la encuesta de satisfacción

Comentarios del *chat* de slack

Como se ha mencionado anteriormente, Patrick y Valentina trabajan en Qualabs, lo que les ha permitido utilizar los canales internos de comunicación de la empresa. Unos de esos canales fue Slack, una herramienta que sirvió de comunicación adicional con el cliente.

Se creó un *channel* dentro de Slack específicamente para la tesis, para así poder coordinar reuniones o escuchar opiniones de forma más rápida y fácil con el cliente.

En la figura 12.39 se puede ver un intercambio de mensajes realizado mediante Slack.

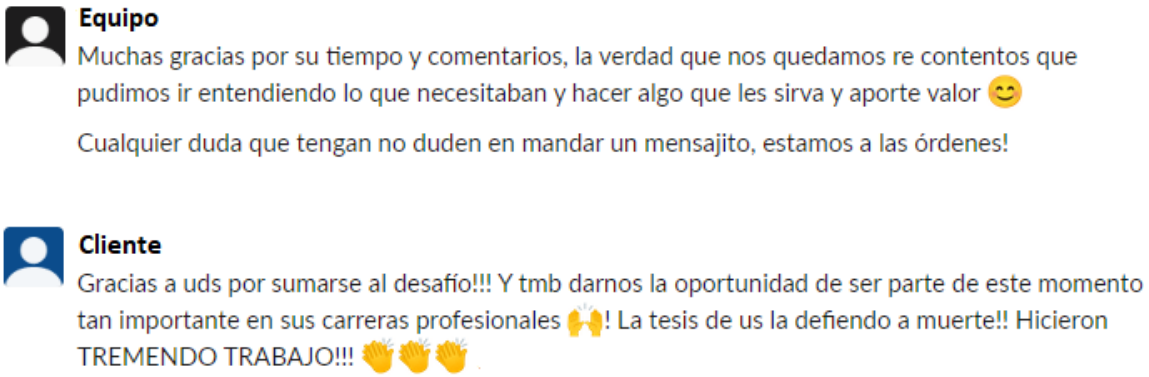


Figura 12.39: Intercambio de mensajes en Slack con el cliente

12.10 Diferentes incidencias por *sprint*

En el siguiente anexo, se mostrará un registro de los *bugs* y *feedbacks* completos cuando se entregó este documento, marcando su clave, el *sprint* en el que fue reportado, y el *sprint* en el que se resolvió la incidencia para ambas tablas. Además, se usará la severidad / prioridad y el *sprint* en el que fue agregado (*sprint* de inicio) en la tabla de *bugs.*

Ambas tablas fueron divididas por *sprints* marcados en color morado.

12.10.1 Bugs

Bug ID	Sprint de reporte	Sprint de Inicio	Sprint de finalizacion	Prioridad
Sprint 8				
Dev-101	8	8	8	Alta
Dev-124	8	8	8	Media
Sprint 9				
Dev-127	9	9	10	Media
Dev-122	9	9	9	Alta
Dev-121	9	9	9	Media
Dev-118	9	9	9	Media
Dev-117	9	9	16	Media
Dev-116	9	9	9	Baja
Dev-115	9	9	9	Baja
Dev-114	9	9	9	Media
Dev-113	9	9	9	Media
Dev-112	9	9	9	Minima
Dev-111	9	9	9	Alta
Sprint 10				
Dev-129	10	10	11	Media
Dev-131	10	10	10	Media
Dev-132	10	10	10	Alta

Figura 12.40: *Bugs* reportados desde el *sprint 8* al 10.

Bug ID	Sprint de reporte	Sprint de Inicio	Sprint de finalizacion	Prioridad
Sprint 12				
Dev-166	12	12	12	Alta
Dev-167	12	12	12	Maxima
Dev-168	12	12	12	Media
Dev-169	12	12	12	Alta
Dev-170	12	12	12	Alta
Dev-172	12	12	12	Alta
Dev-176	12	12	12	Media
Dev-177	12	12	13	Media
Dev-179	12	12	12	Media
Dev-181	12	12	13	Minima
Dev-182	12	0	0	Baja
Dev-183	12	13	13	Alta
Sprint 13				
Dev-180	13	13	14	Media
Dev-201	13	13	13	Alta
Dev-220	15	13	13	Media
Dev-221	13	15	15	Alta
Dev-223	13	15	16	Media

Figura 12.41: Bugs desde e el sprint 12 al 13.

Bug ID	Sprint de reporte	Sprint de Inicio	Sprint de finalizacion	Prioridad
Sprint 15				
Dev-242	15	15	15	Alta
Dev-243	15	15	15	Media
Dev-244	15	15	15	Media
Dev-245	15	15	16	Minima
Dev-246	15	15	15	Alta
Dev-247	15	15	15	Media
Dev-248	15	15	15	Alta
Sprint 16				
Dev-301	16	16	16	Media
Dev-290	16	16	16	Maxima
Sprint 17				
Dev-340	17	17	17	Alta
Dev-342	17	17	17	Alta
Dev-350	17	17	17	Media
Sprint 18				
Dev-338	18	18	18	Media
Dev-353	18	18	18	Media
Dev-357	18	18	18	Media
Dev-373	18	18	18	Media
Dev-352	18	19	19	Media
Sprint 19				
Dev-365	19	19	19	Media
Dev-364	19	19	19	Media

Figura 12.42: *Bugs* reportados desde el *sprint* 15 al 19.

12.10.2 Feedback

Reporte ID	Sprint de Reporte	Sprint de finalizacion
Sprint 12		
Dev-186	12	13
Dev-195	12	13
Dev-185	12	13
Dev-187	12	13
Dev-188	12	13
Dev-191	12	13
Dev-194	12	13
Dev-197	12	13
Dev-192	12	13
Dev-189	12	14
Dev-192	12	18
Sprint 13		
Dev-209	13	13
Dev-213	13	15
Dev-208	13	15
Dev-216	13	14
Dev-219	13	14
Dev-212	13	14
Dev-218	13	14
Dev-217	13	14
Dev-211	13	13
Dev-214	13	13
Dev-210	13	13
Dev-232	13	17

Figura 12.43: *Feedback* reportado entre el *sprint* 12 y el *sprint* 13.

Reporte ID	Sprint de Reporte	Sprint de finalizacion
Sprint 15		
Dev-239	15	15
Dev-240	15	15
Sprint 16		
Dev-289	16	16
Dev-288	16	16
Dev-334	16	17
Dev-326	16	17
Dev-304	16	17
Sprint 18		
dev-354	18	18
dev-358	18	18
Sprint 19		
dev-374	19	19
Dev-375	19	19

Figura 12.44: *Feedback* reportado entre el *sprint* 15 y el *sprint* 19.

12.11 Evolución del Sistema

En este anexo se mostrará como fue evolucionado el sistema a lo largo del tiempo, tomando en cuenta como *milestones* las versiones del mismo. Se tomarán en cuenta ciertas funcionalidades del sistema, pero no todas, con el fin de facilitar la visualización de los cambios considerados relevantes.

12.11.1 Principales funcionalidades en la versión 1.0.0

En esta sección, se mostrará el estado del sistema cuando se entregó el primer *release* al cliente, tomando en cuenta ciertas funcionalidades principales que fueron evolucionando desde este *release* hasta el final del proyecto.

Dashboard

En la siguiente imagen, se muestra el estado del *Dashboard* en el primer *release*.

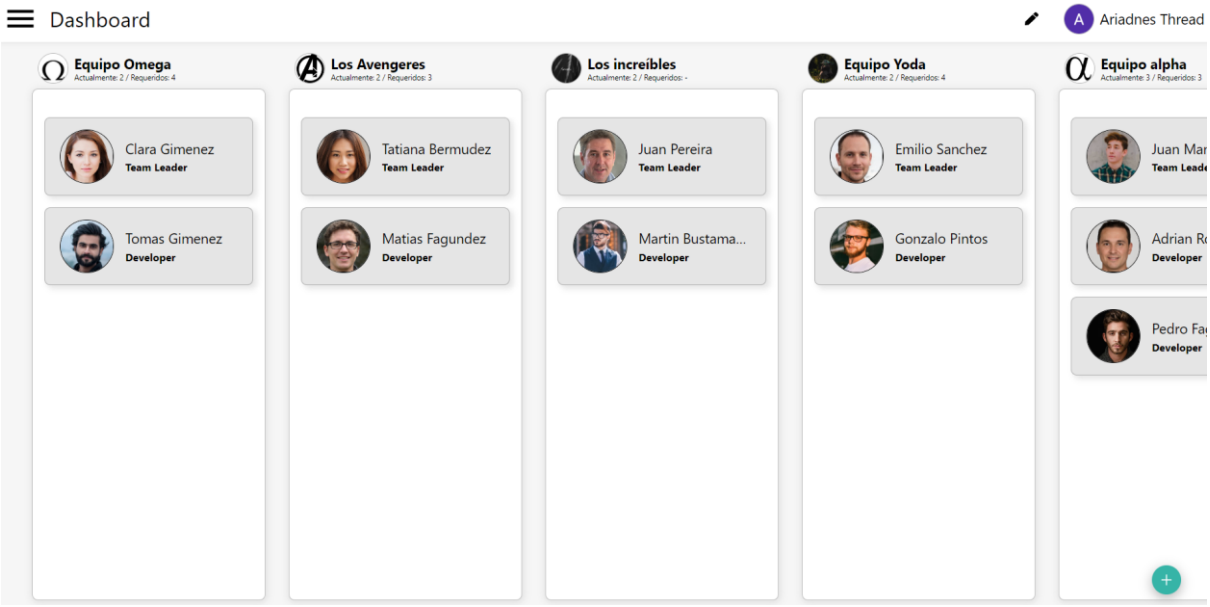


Figura 12.45 *Dashboard* en la versión 1.0.0

El *Dashboard* se mantiene simple, mostrando los equipos y sus empleados junto con su rol. Carece de visualización respecto a las vacantes, y también carece de funcionalidades agregadas posteriormente, como un *zoom*. Este diseño de diagramación en columnas y tarjetas fue realizado así con el fin de mantener el formato similar al esquema de la aplicación Mural, que era con la que los miembros de P&C estaban familiarizados y no querían que el nuevo sistema les implique un cambio radical frente a la estructura a la que estaban acostumbrados.

Modal de equipo en el *Dashboard*

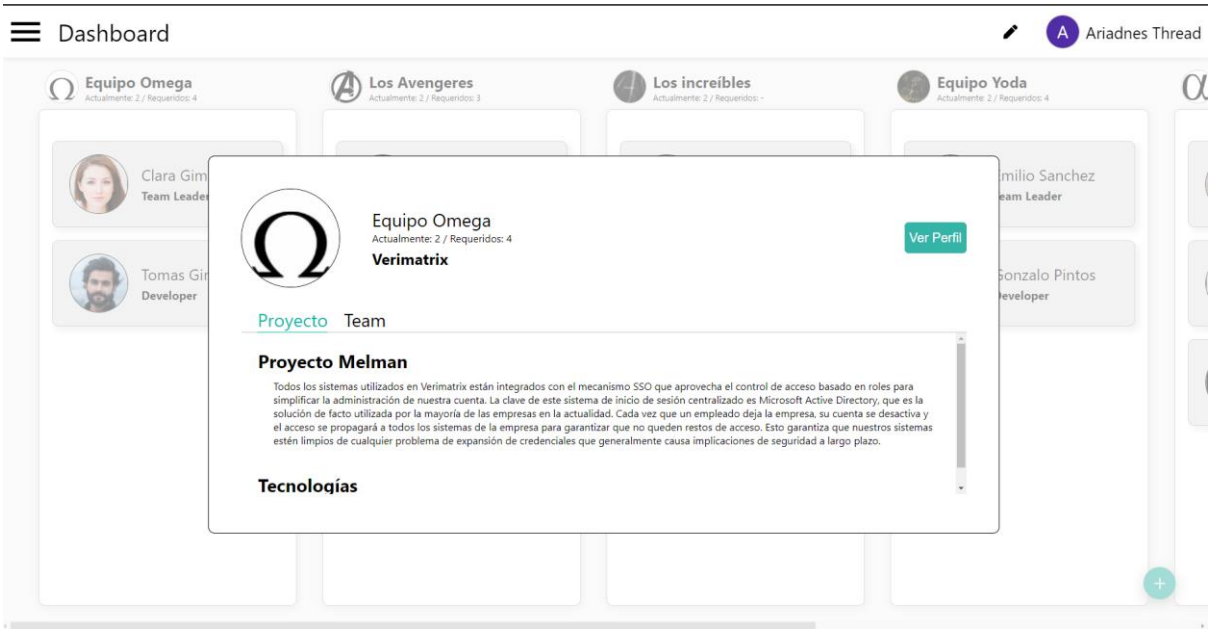


Figura 12.46: Modal de equipo en el *Dashbaord* en la versión 1.0.0

El modal de equipo contiene información del proyecto, el cliente marcado en la parte superior, el número actual de miembros y los miembros requeridos (solicitados por el cliente para el proyecto), y en la otra *tab* contiene la lista de los miembros del equipo.

Modal de empleado en el *Dashboard*

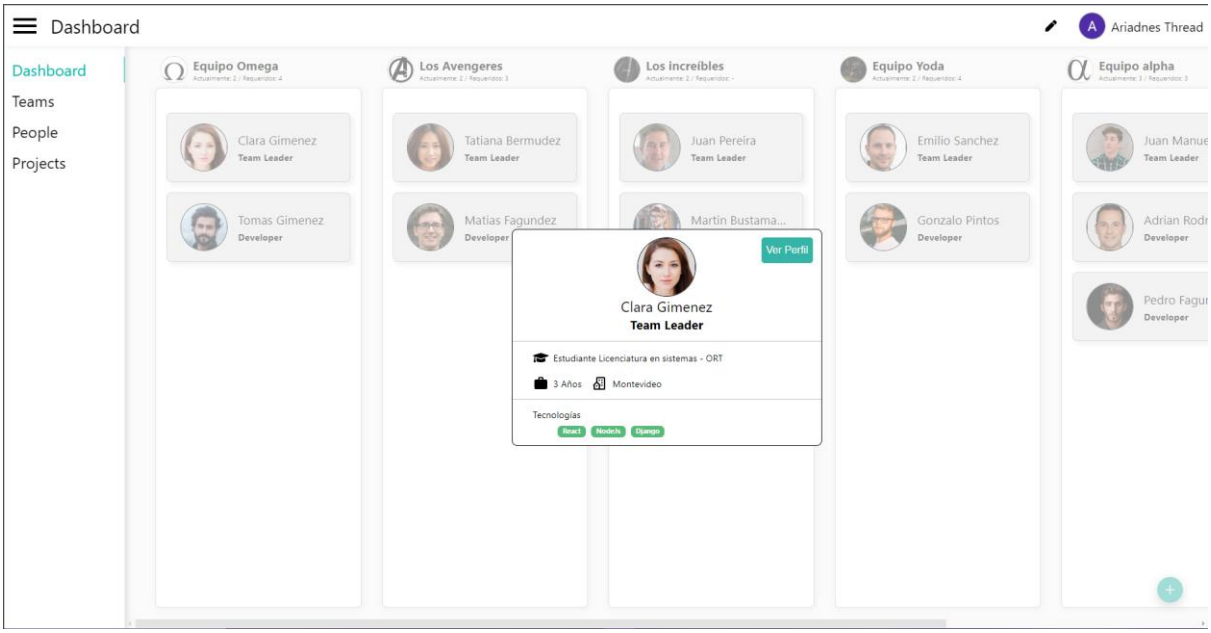


Figura 12.47: Modal de empleado en el *Dashboard* en la versión 1.0.0

El modal de empleado contiene información básica del empleado, como las tecnologías que conoce, donde estudió, su rol y hace cuanto trabaja en la empresa. A la izquierda se muestra el *sidebar* de la aplicación, que permite navegar hacia otras secciones del sistema.

Ficha de creación de equipo

A continuación, se muestra la ficha de creación de equipos. Los campos obligatorios se encuentran marcados con "*", esto aplica al resto de fichas de creación de entidades.

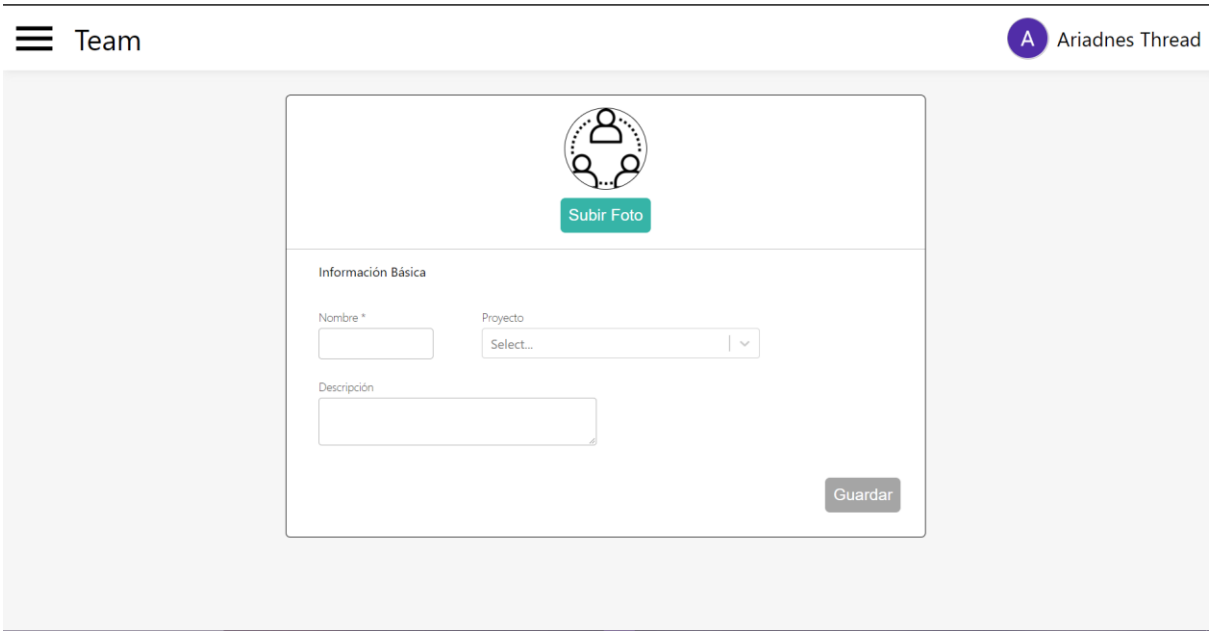


Figura 12.48: Ficha de creación de equipos en la versión 1.0.0

La ficha de creación de equipos solo posee tres campos: el nombre del equipo, el proyecto que le será asignado en forma de un *selector* con los proyectos disponibles (es decir, sin equipo), y una descripción. Solo el nombre es requerido.

Ficha de creación de proyecto

A continuación, se muestra la ficha de creación de proyectos.

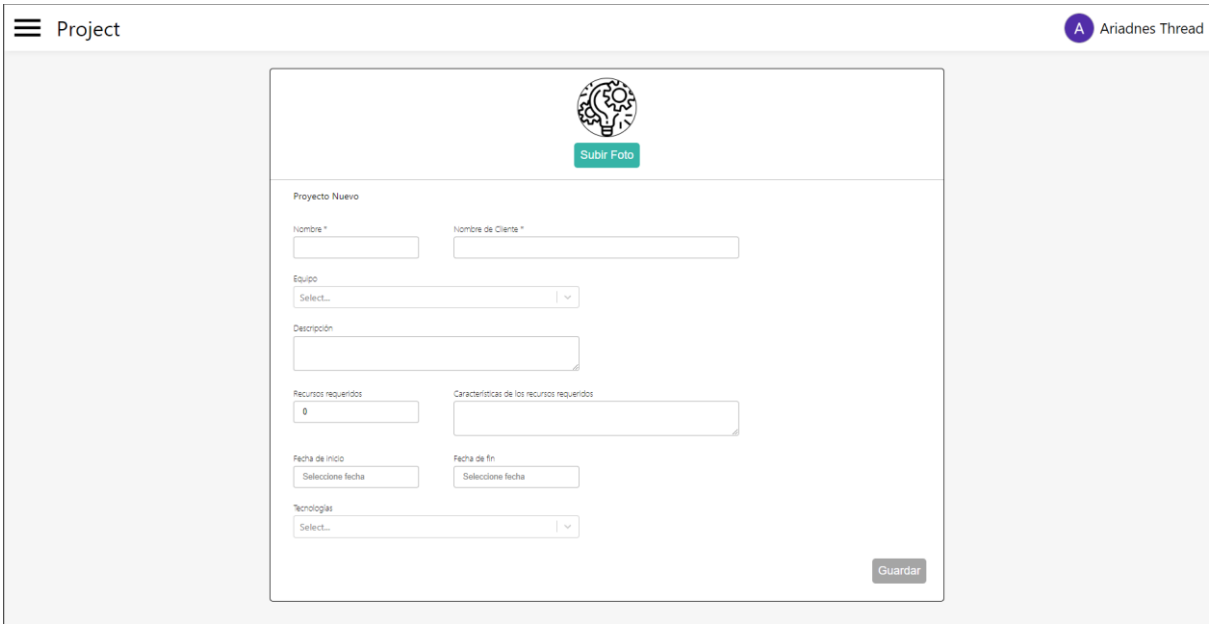


Figura 12.49: Ficha de creación de proyecto en la versión 1.0.0

La ficha de creación de proyecto contiene campos relacionados al proyecto en sí. En esta instancia, el cliente es únicamente un texto, existe el campo de requeridos (siendo este campo el que representa los recursos humanos pedidos por el cliente a su proyecto), y el agregado de tecnologías. El resto de los campos se mantuvieron parecidos a lo largo del tiempo.

Ficha de creación de empleado

A continuación se muestra la ficha de creación de empleado.

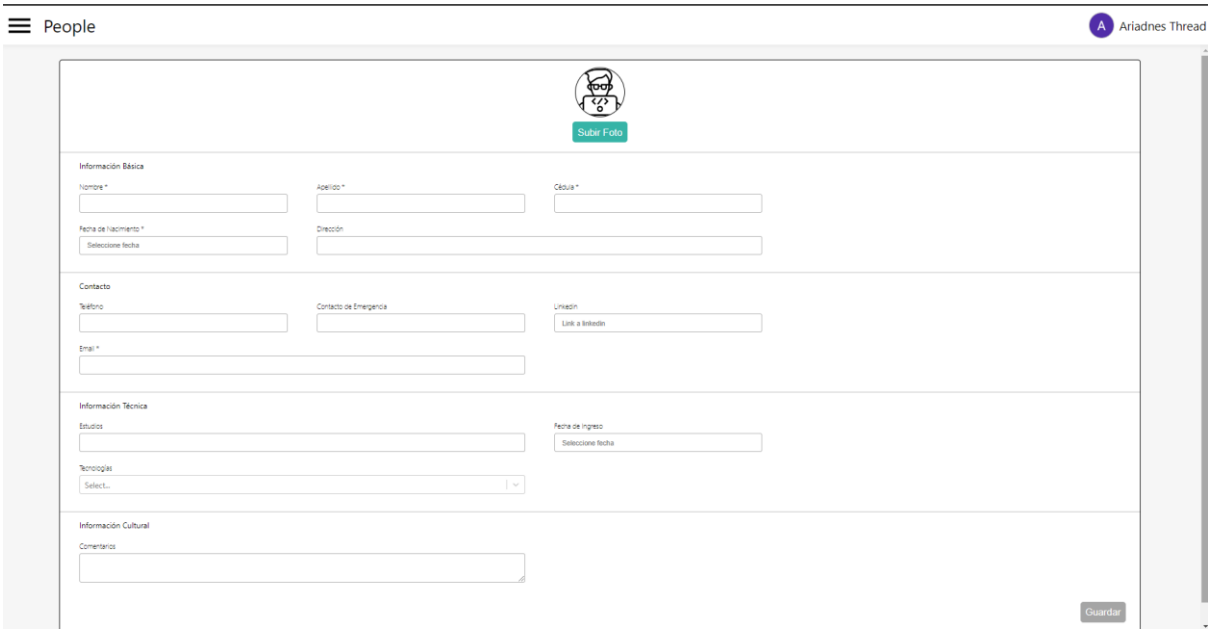


Figura 12.50: Ficha de creación de empleado en la versión 1.0.0

La ficha de creación de empleados contiene un buen número de campos requeridos, entre ellos la fecha de nacimiento, un único email requerido, y la cédula, que en su momento era requerida y también tenía que ser válida, es decir, el número correcto de dígitos y que el dígito verificador fuera el correcto. Además, Linkedin no era requerido, ni tenía que ser un *link*. Por último, el contacto de emergencia solo podía ser un número de teléfono válido (al igual que el número de teléfono corriente), sin dar lugar a comentarios o de quien era ese teléfono (madre, padre, pareja, entre otros). Al igual que la ficha de creación de proyectos, presenta un campo para las tecnologías.

Ficha de visualización de equipo

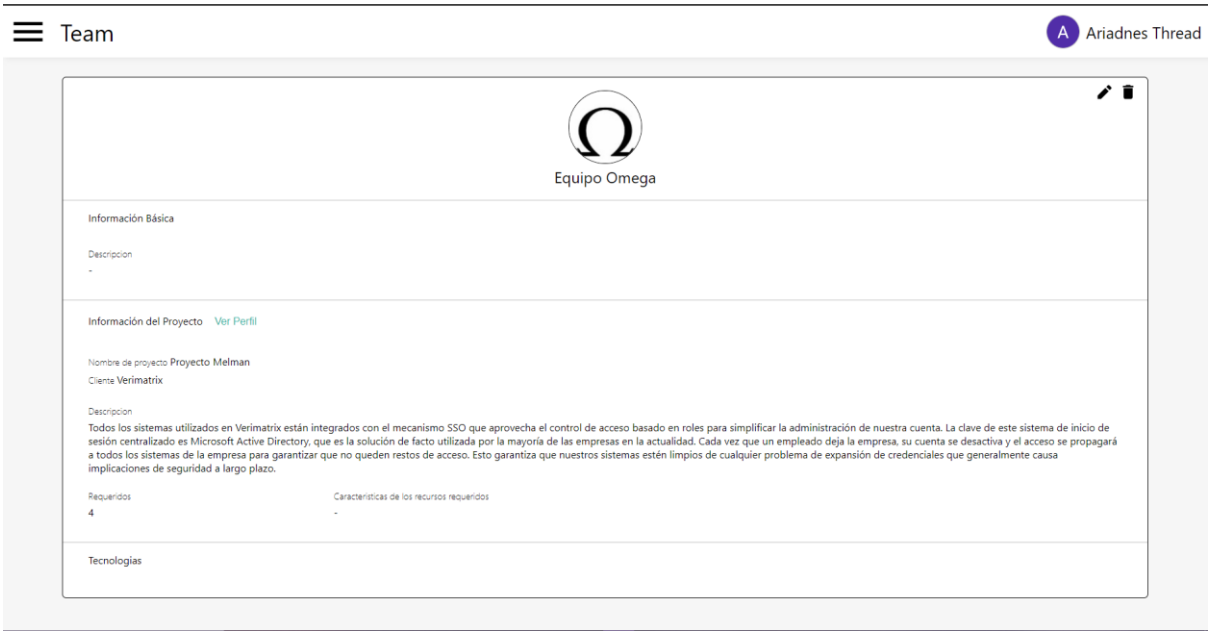


Figura 12.51: Ficha de visualización de equipo en la versión 1.0.0

La ficha de visualización de equipo se muestra como una ficha única, con información del equipo y una breve descripción del proyecto que posee dicho equipo, si lo tuviese.

Ficha de visualización de proyecto

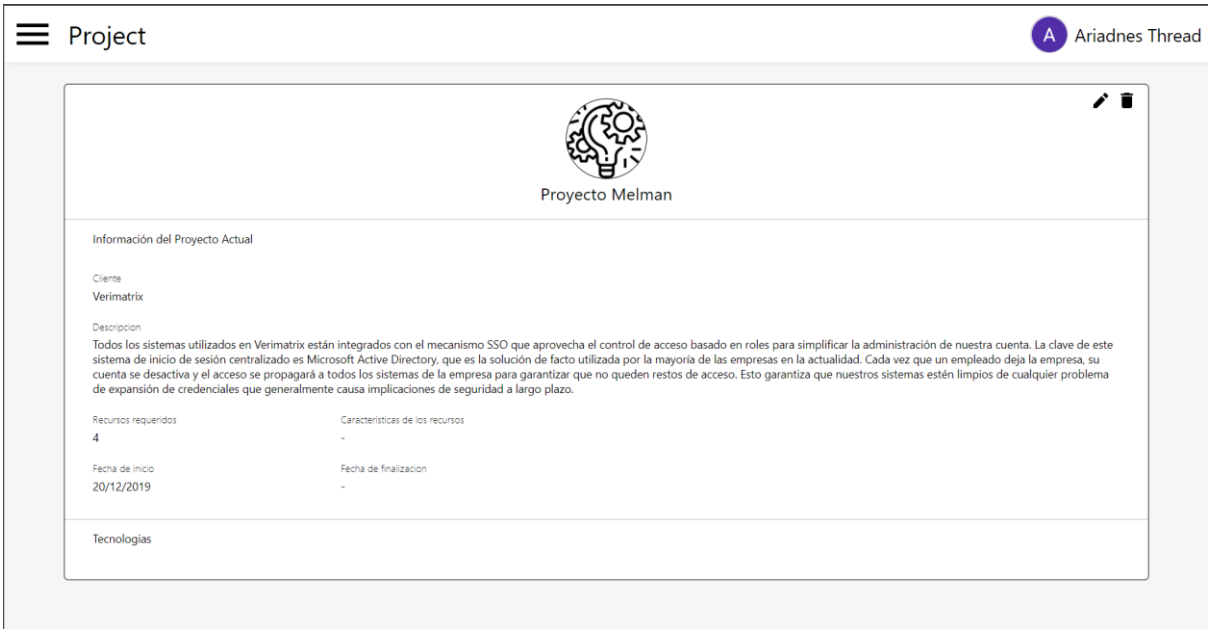


Figura 12.52: Ficha de visualización de proyecto en la versión 1.0.0

La ficha de visualización de proyecto no posee información del equipo al que pertenece, sino que simplemente muestra la información del mismo. En la sección inferior, se muestran las tecnologías que en el futuro pasarán a ser parte de las categorías del sistema.

Ficha de visualización de empleado

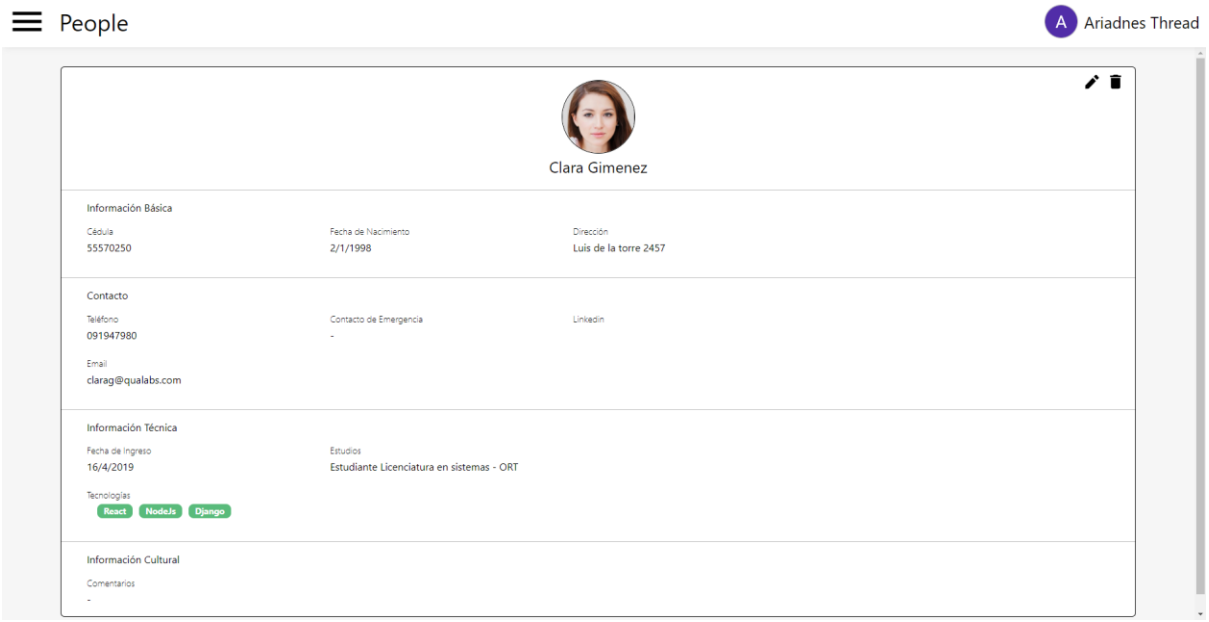


Figura 12.53: Ficha de visualización de empleado en la versión 1.0.0

La visualización del empleado contiene información relevante del empleado, es decir, la información que fue añadida en la creación y modificada en la edición.

12.11.2 Principales funcionalidades en la versión 2.0.0

En esta sección se verá el estado de las principales funcionalidades en la versión 2.0.0, tomando en cuenta funcionalidades antiguas de la versión 1.0.0, y también nuevas funcionalidades que fueron evolucionando a lo largo de las versiones del sistema.

Dashboard

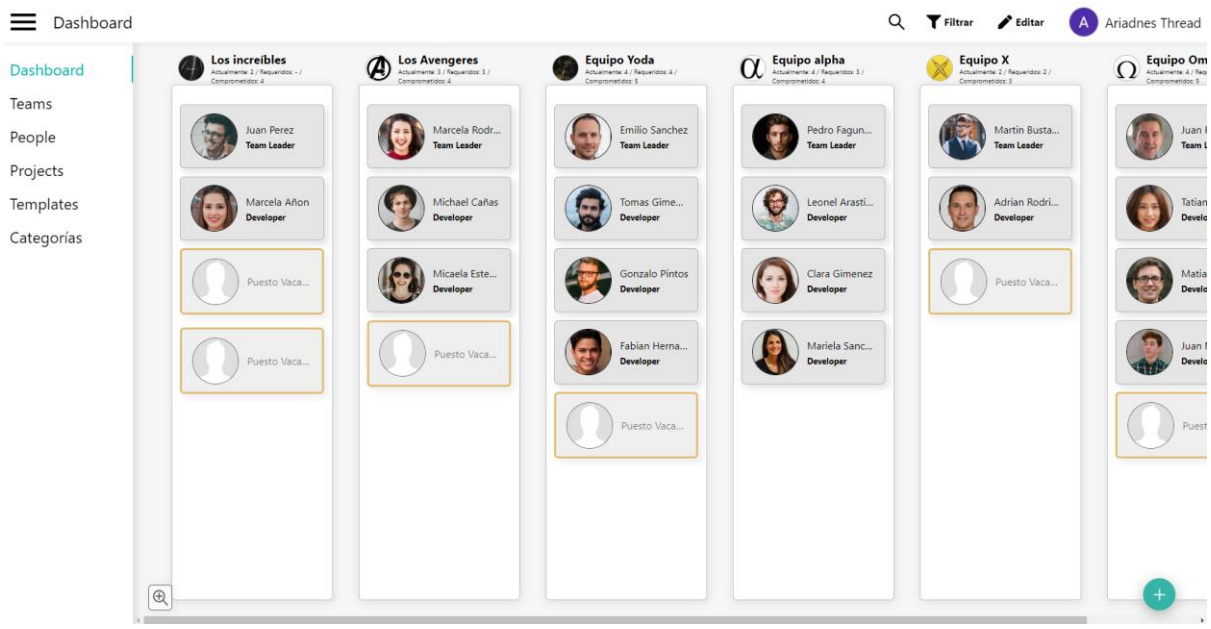


Figura 12.54: *Dashboard* en la versión 2.0.0

El *Dashboard* presenta varias evoluciones, siendo estas una funcionalidad de *zoom* integrada, el sistema ahora marca a los puestos vacantes del cliente en rojo, y puestos vacantes del equipo en amarillo (atributo comprometidos, que será explicado más adelante). Además, existen nuevas funcionalidades aparte de la edición del *Dashboard*, siendo esta los filtros y un buscador (que es global a todo el sistema, no solo el *Dashboard*).

Se eligió mostrar a los puestos vacantes del sistema de esta forma para que puedan ser visualizados y entendidos rápidamente, sin tener que buscar los números de actuales, comprometidos y requeridos manualmente, solicitado de esta forma por medio de *feedback*. La funcionalidad de *zoom* se agregó en base a *feedbacks* de uso en base a diferentes resoluciones de monitor y dado que la funcionalidad del navegador deformaba las columnas y cartas.

Modal de empleado

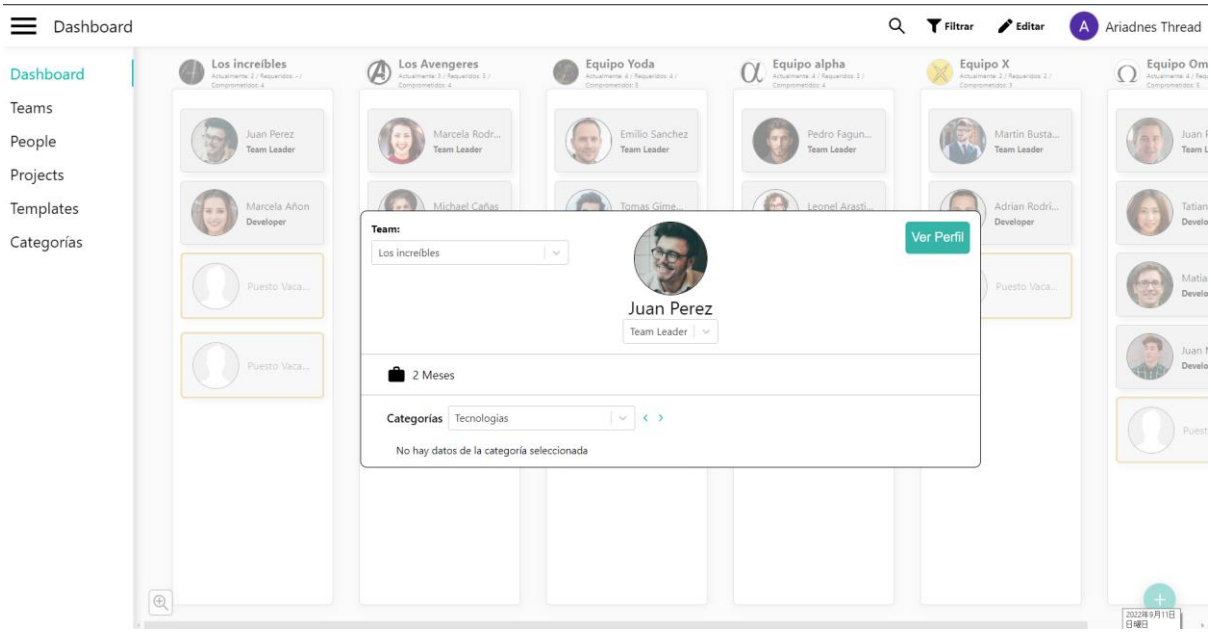


Figura 12.55: Modal de empleado en la versión 2.0.0

El modal de empleados presenta grandes cambios respecto a la versión anterior, viendo agregados las categorías al mismo, además se sacó información no relevante para el transcurso de la operación del sistema, dejando únicamente la antigüedad. El modal fue ampliado, y se agregaron dos selectores: uno para cambiar el rol, y otro para cambiar el equipo rápidamente del empleado.

Modal de equipo, *tab* de categorías

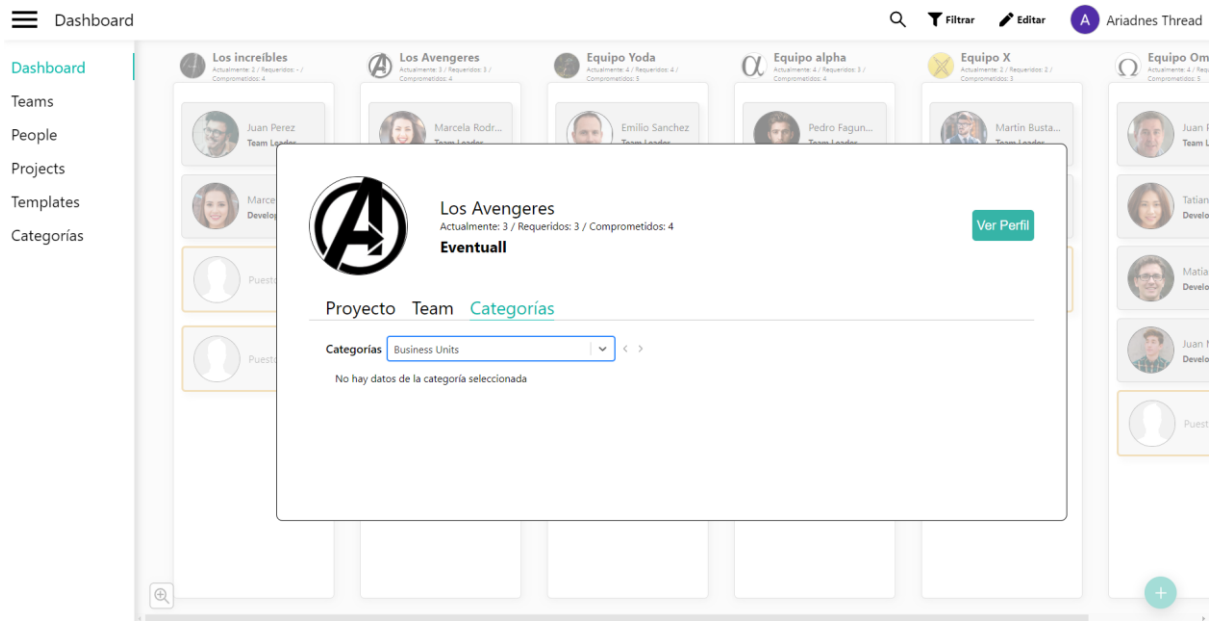


Figura 12.56: Modal de equipo, *tab* de categorías en la versión 2.0.0

El modal de equipos presenta el agregado de una nueva *tab*, referente a las categorías del mismo. Se decidió separarlo de otras *tabs* con el fin de mostrar fácilmente y de manera no compleja las categorías del mismo.

Modal de equipo, *tab* de proyectos

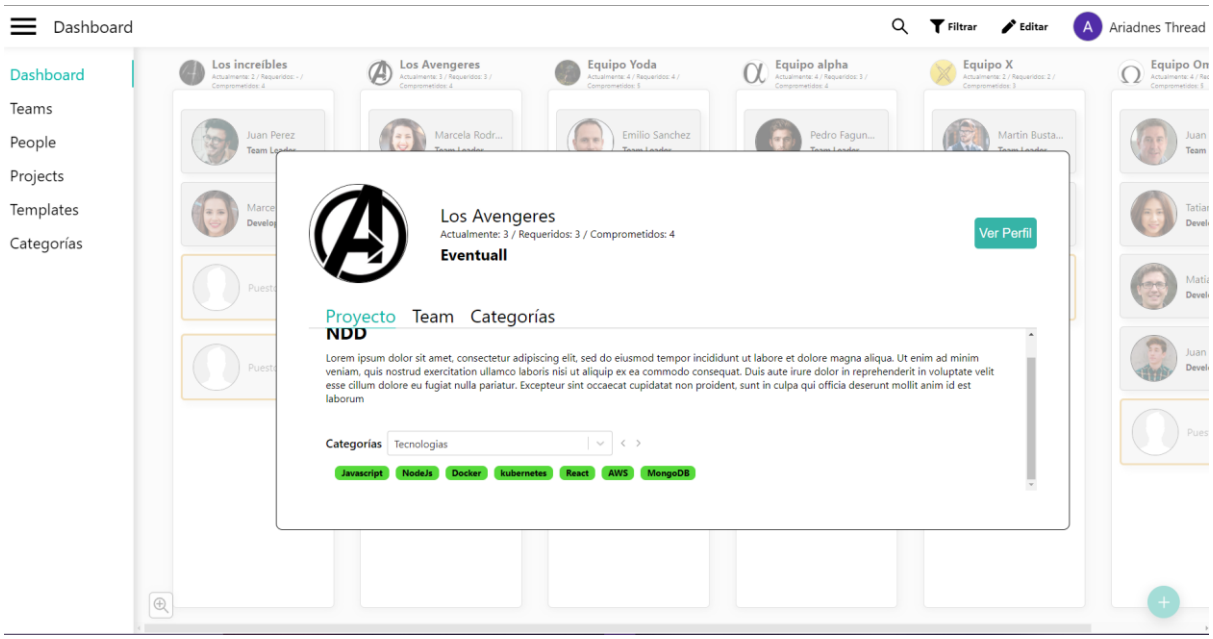


Figura 12.57: Modal de equipo, *tab* de proyectos en la versión 2.0.0

La *tab* de proyectos presenta un cambio, siendo este el reemplazo de la sección de tecnologías por la sección de categorías.

Filtros generales

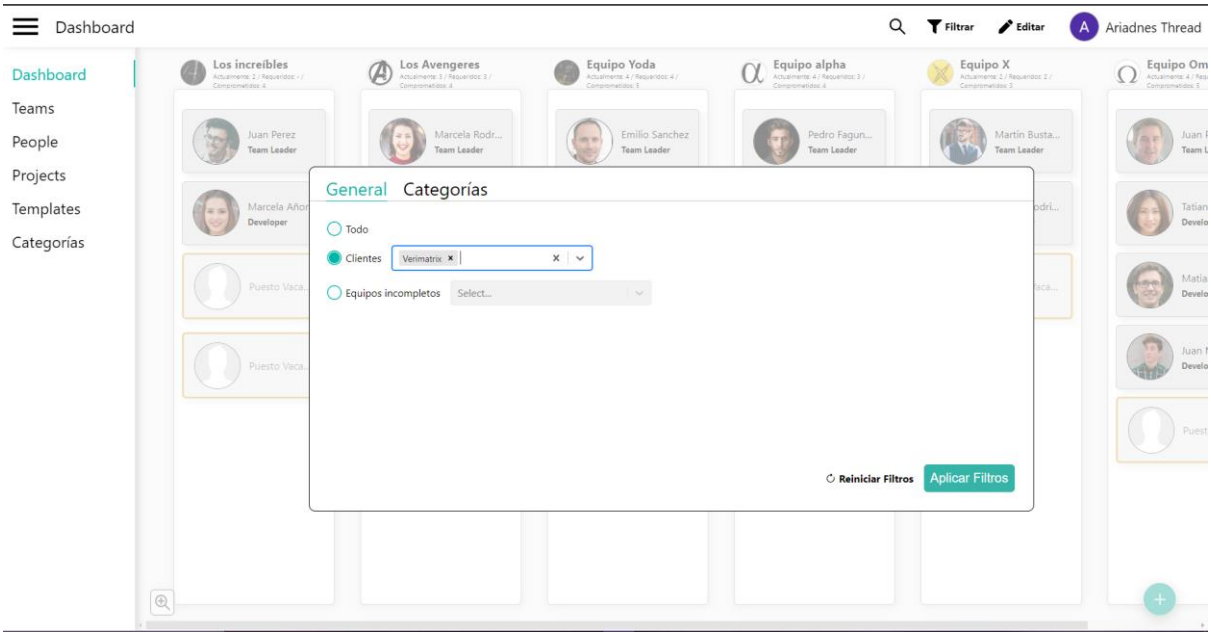


Figura 12.58: Filtros generales en la versión 2.0.0

Se agregó una funcionalidad de filtros al *Dashboard*, con el fin de filtrar lo que se muestra en base a ciertos parámetros. En el ámbito de los filtros generales, los filtros son por cliente o visualizar únicamente los equipos incompletos.

Filtros por categorías

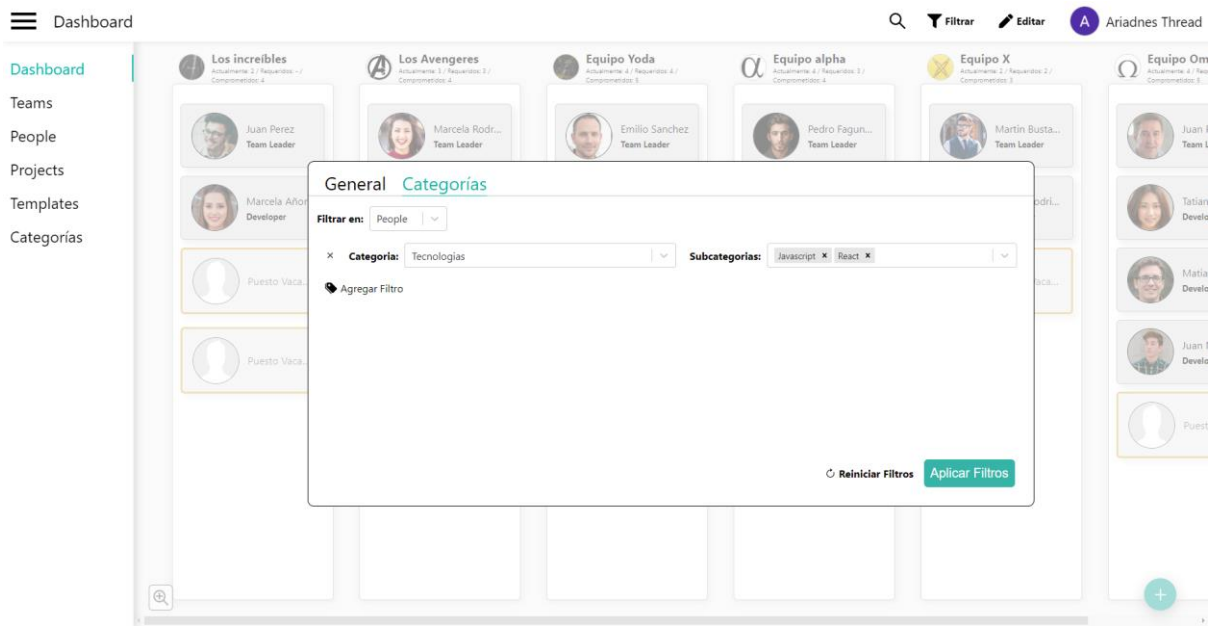


Figura 12.59: Filtros por categorías en la versión 2.0.0

Los filtros por categorías fueron agregados en esta versión, permitiendo filtrar cualquier entidad en base a sus categorías y subcategorías.

Buscador

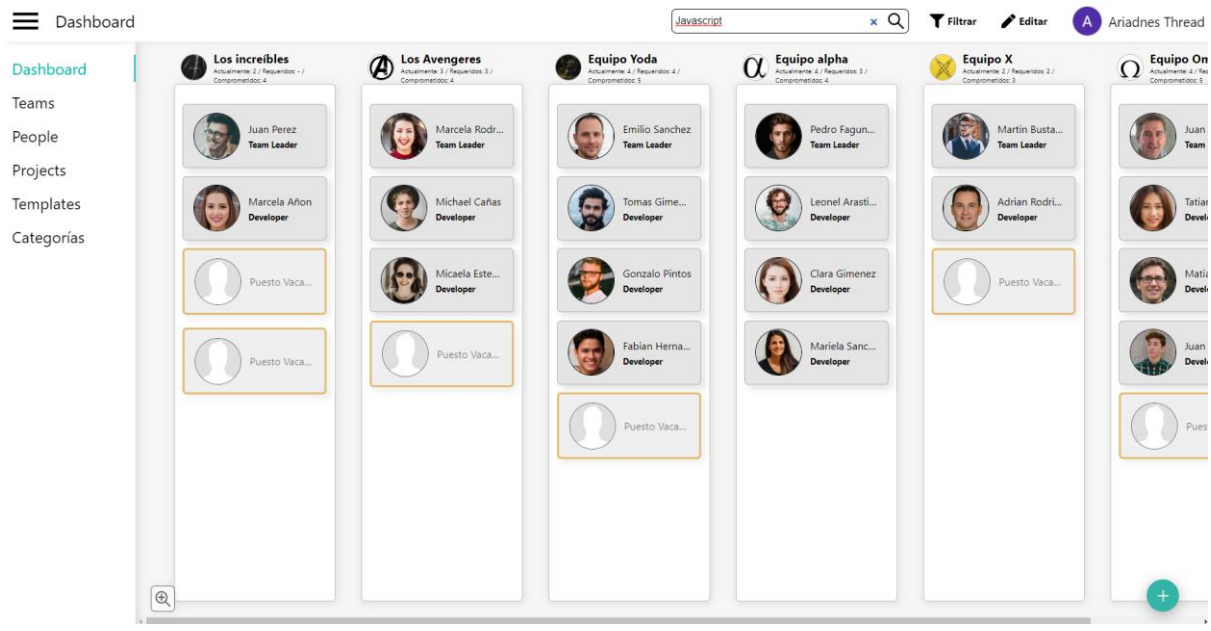


Figura 12.60: Buscador en uso en la versión 2.0.0

El buscador se puede usar desde cualquier lugar del sistema, se utiliza escribiendo una palabra y luego procede a buscar en las tres entidades principales los atributos que coincidan con esa palabra o parte de ella.

Ficha de creación de proyecto

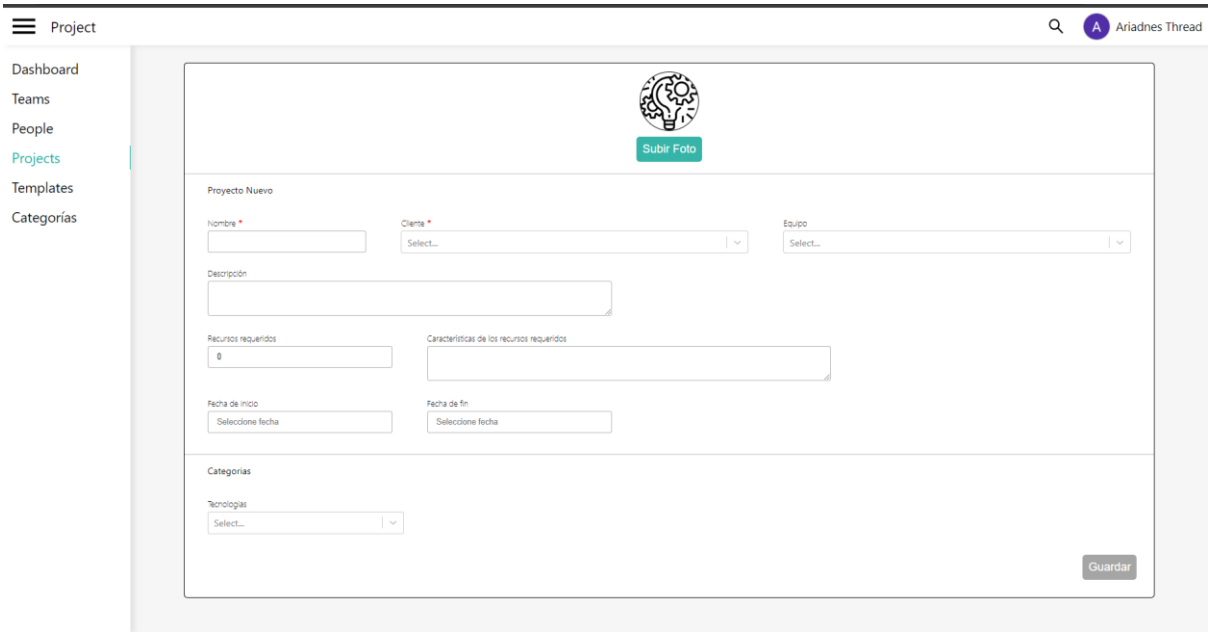


Figura 12.61: Ficha de creación de proyecto en la versión 2.0.0

La ficha de creación de proyecto presenta una nueva sección de categorías, reemplazando a la sección de tecnologías. Agregando, contiene una forma de asignar el equipo al crear el proyecto, y el campo de clientes presenta un gran cambio, pasando de ser un campo de texto, a ser una entidad creable y seleccionable por medio del selector.

Ficha de creación de empleado, parte 1

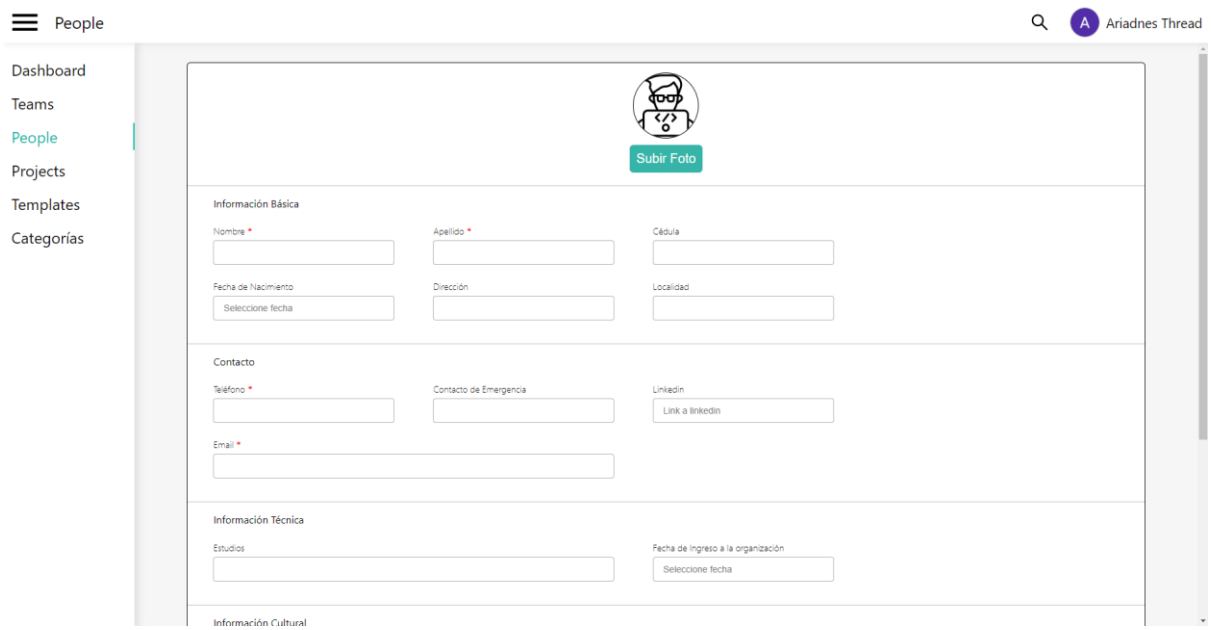


Figura 12.62: Primera parte de la ficha de empleado en la versión 2.0.0

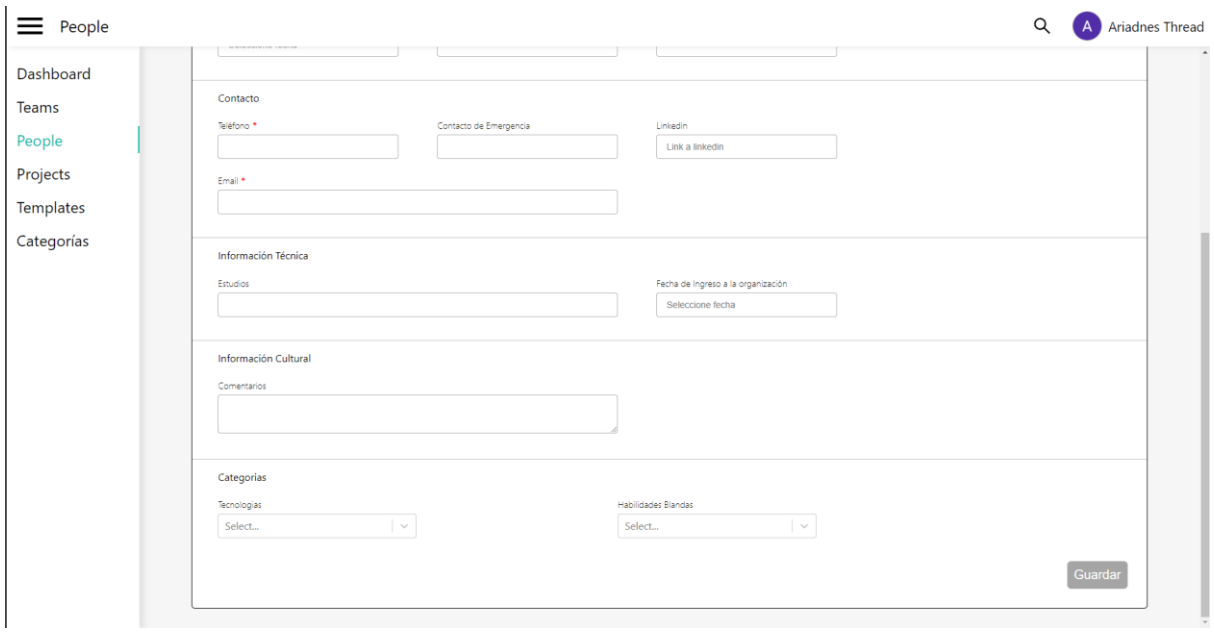


Figura 12.63: Segunda parte de la ficha de creación de empleado en la versión 2.0.0

La ficha de creación de empleado presenta grandes cambios respecto a la primera versión, particularmente el cambio respecto a los campos requeridos. La cédula ya no es más un campo requerido, se elimina el chequeo de validez de cédula y de teléfono, y el contacto de emergencia ahora permite texto libre. También se reemplaza el campo de tecnologías por un campo de categorías.

Ficha de creación de equipo

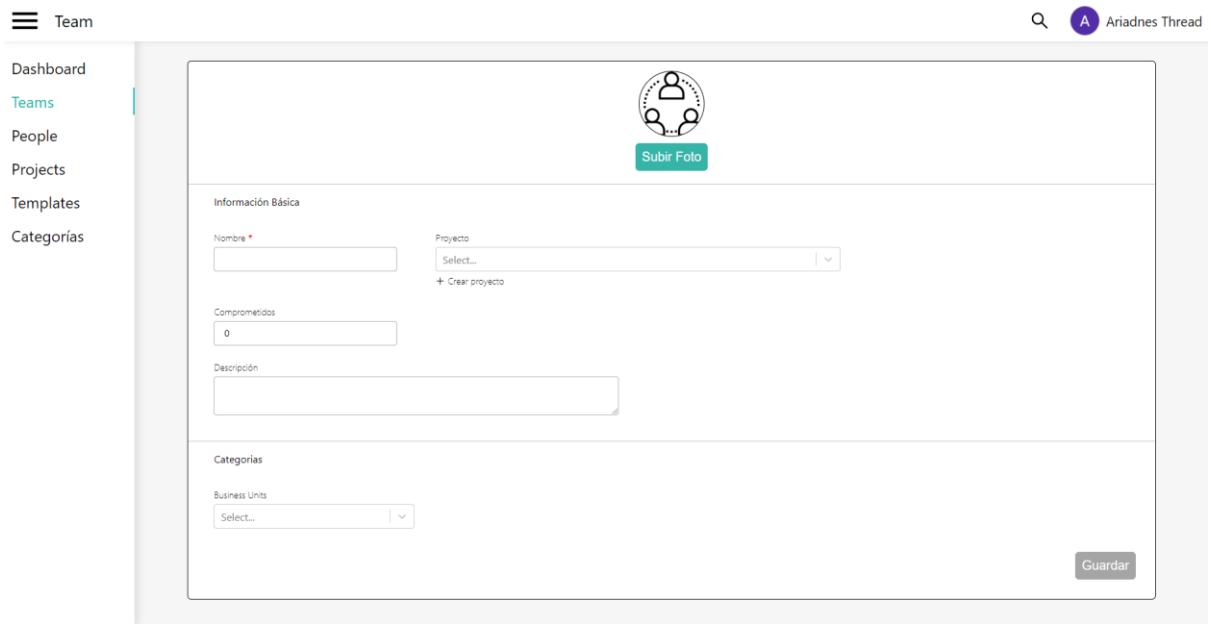


Figura 12.64: Ficha de creación de equipo en la versión 2.0.0

La ficha de creación de equipo contiene un nuevo campo en la forma de “comprometidos”, representa al número de recursos humanos que la empresa Qualabs compromete al equipo, pudiendo ser un número superior al número de los requeridos por el cliente. Agregando, el selector de proyecto también permite la creación rápida de proyectos.

Ficha de visualización de equipo, *tab* principal

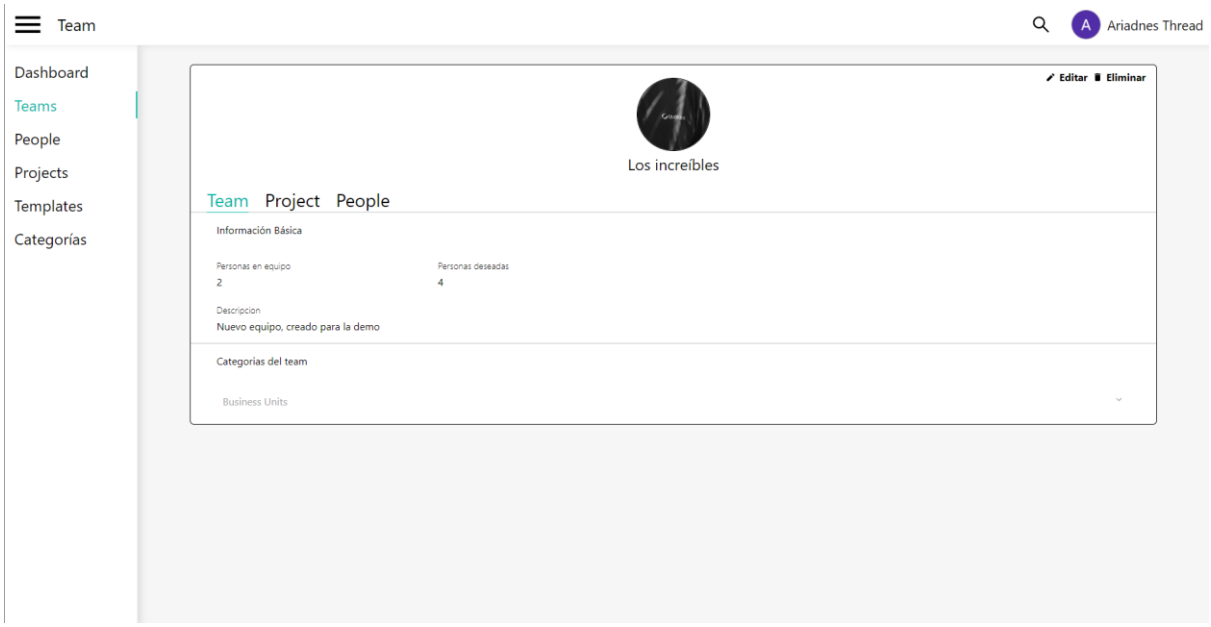


Figura 12.65: Ficha de visualización de equipo en la *tab* de equipo en la versión 2.0.0

En la *tab* principal se decidió sacar los datos referentes al proyecto y mostrarlos en su propio *tab*. Agregando, se muestra el campo de comprometidos. El modelo de *tabs* fue aplicado también para la ficha de visualización de empleados -y que puedan ver el equipo al que pertenecen- y para la ficha de visualización de proyectos -mostrando el equipo que lo está realizando-. Siguiendo, se muestran las categorías en la parte inferior de la *tab*. Por último, se agregaron descripciones a las dos acciones posibles para el equipo, editarlo y eliminarlo.

Ficha de visualización de equipo, *tab* de proyecto

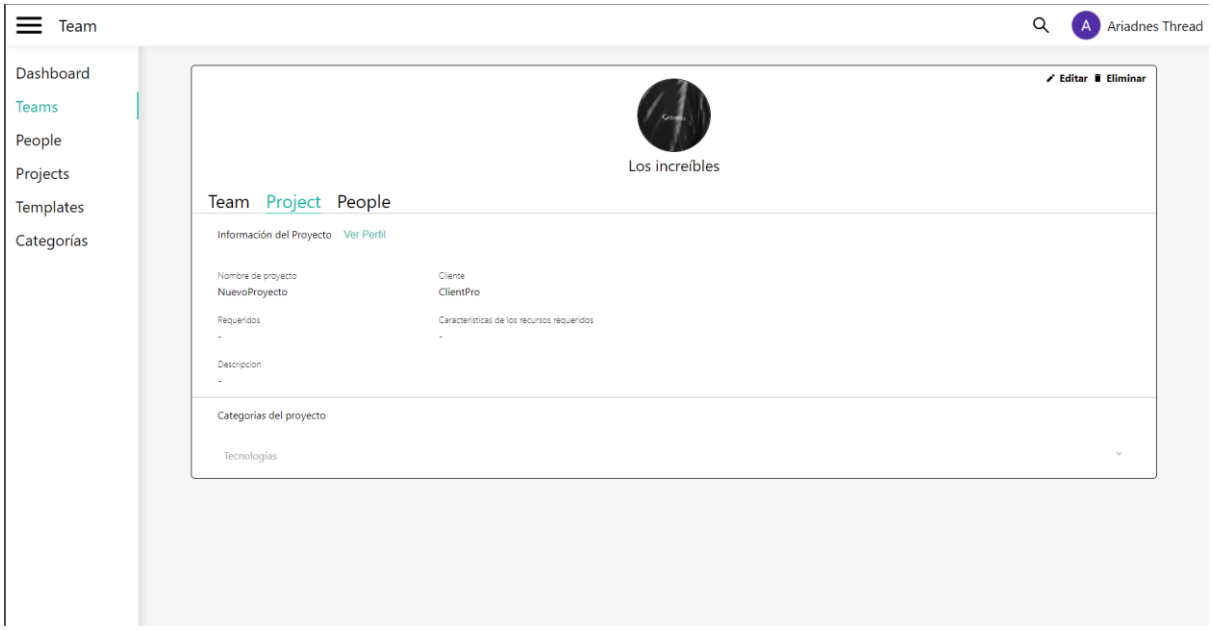


Figura 12.66: *Tab* de proyecto en la ficha de visualización de equipo en la versión 2.0.0

Siguiendo con el modelo general mencionado anteriormente, se muestra el proyecto en otra *tab* con sus datos, y las categorías en la sección inferior de la pantalla.

Ficha de visualización de equipo, *tab* de personas

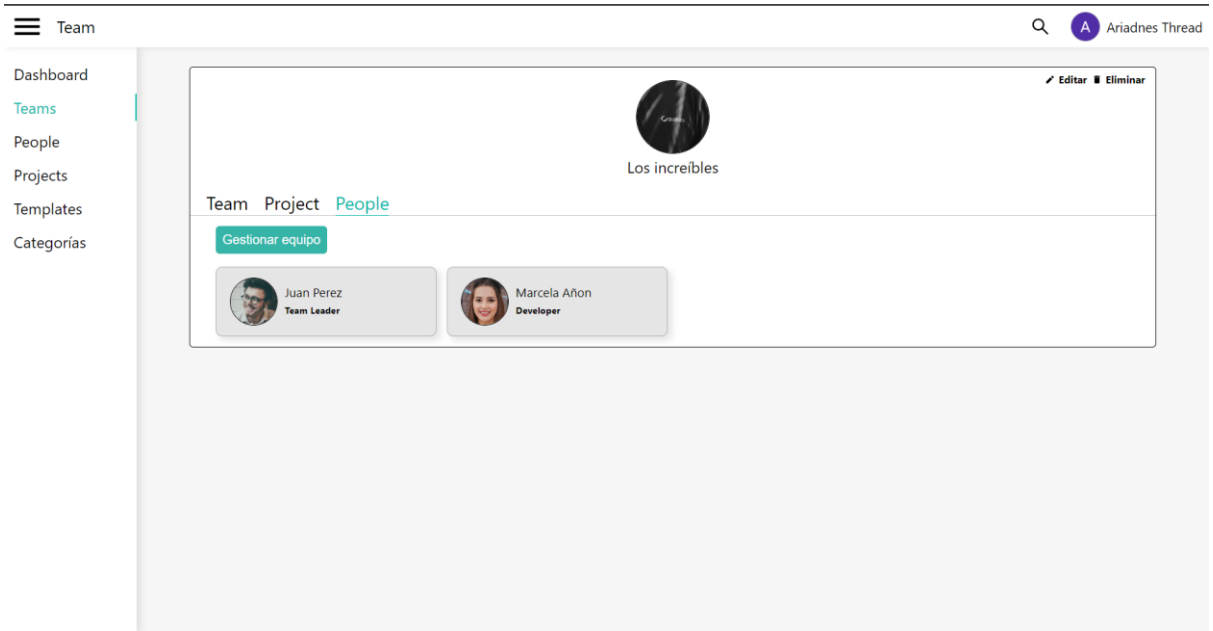


Figura 12.67: *Tab* de personas en visualización de equipo en la versión 2.0.0

Finalmente, se agregó una *tab* para ver y gestionar las personas del equipo, con el fin de facilitar el proceso de armado de los mismos. Esto surgió en base a *feedback* del cliente.

Ficha de visualización de equipo, gestión de personas

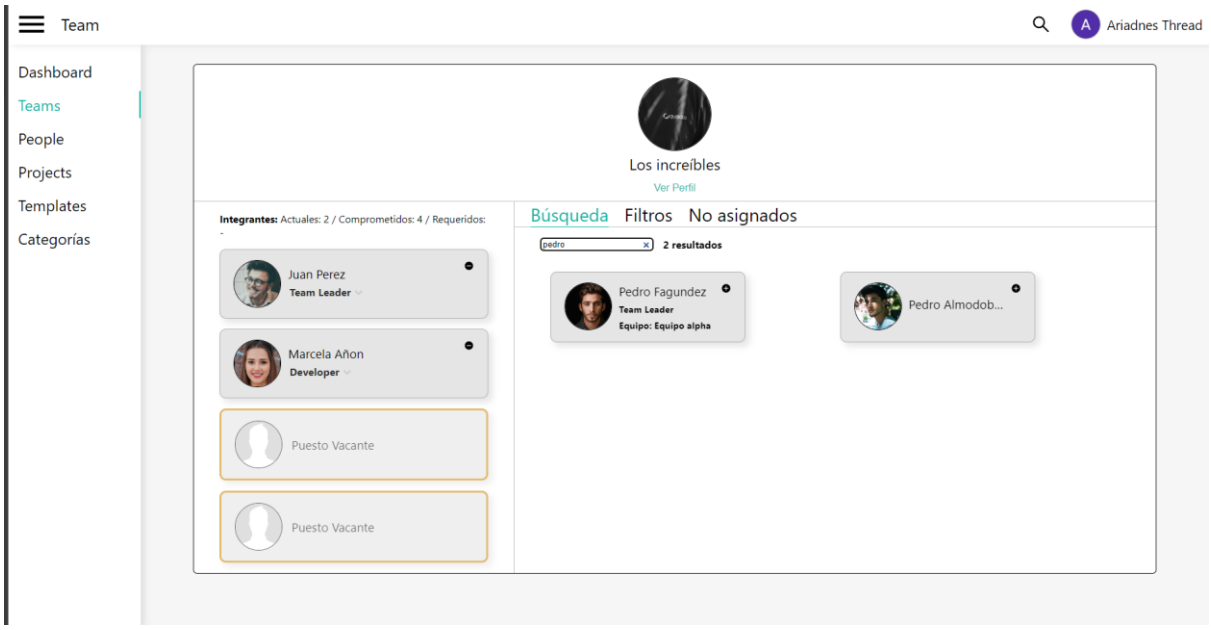


Figura 12.68: Gestión de personas en el equipo en la versión 2.0.0

El proceso de gestión de personas presenta dos partes, el equipo actual, y las formas de buscar futuros miembros del equipo. Para esto, se agregaron tres componentes del *Dashboard*, siendo estos la búsqueda, los filtros y los empleados no asignados.

12.11.3 Principales funcionalidades en la versión 3.0.0

En esta sección se verá el estado de las principales funcionalidades en la versión 3.0.0, tomando en cuenta funcionalidades antiguas de la versión 1.0.0, funcionalidades agregadas o que evolucionar en la versión 2.0.0 y también en la versión 3.0.0.

Filtros generales

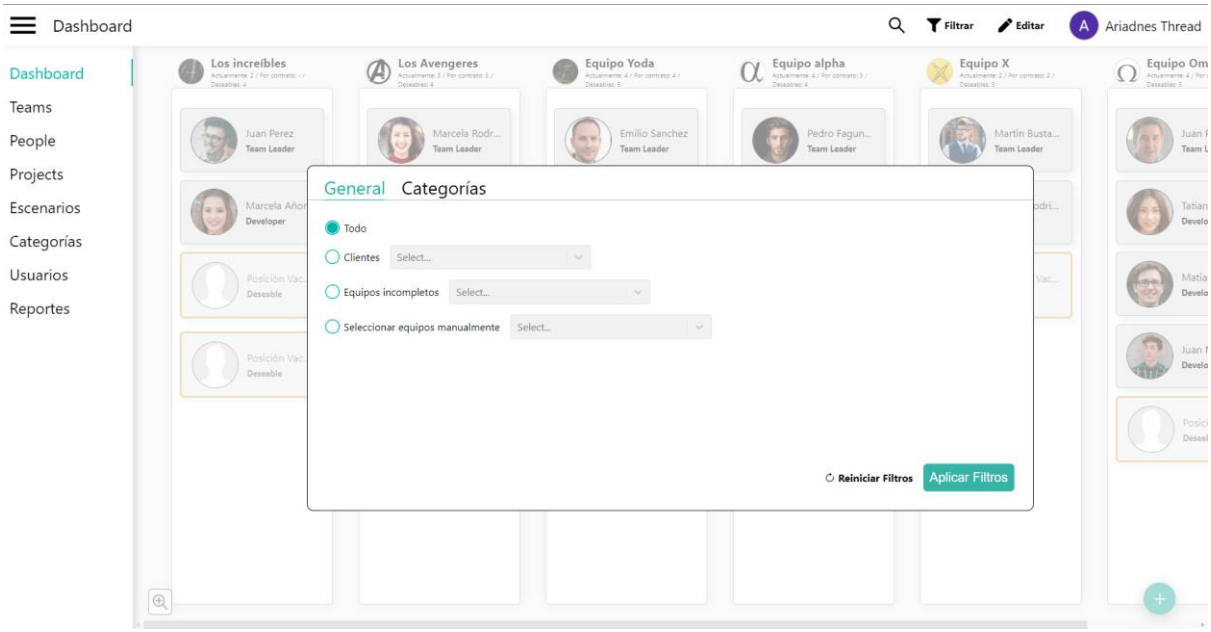


Figura 12.69: Filtros generales en el *Dashboard* en la versión 3.0.0

Se agrega un nuevo filtro en base a pedido del cliente, pudiendo seleccionar manualmente los equipos deseados. Esta funcionalidad surge de problemas de escala, dado que la empresa está creciendo rápidamente, y quizás sea necesario poner el foco en un número limitado de equipos para la operación del sistema.

Modal de equipo, *tab* de categorías y links.

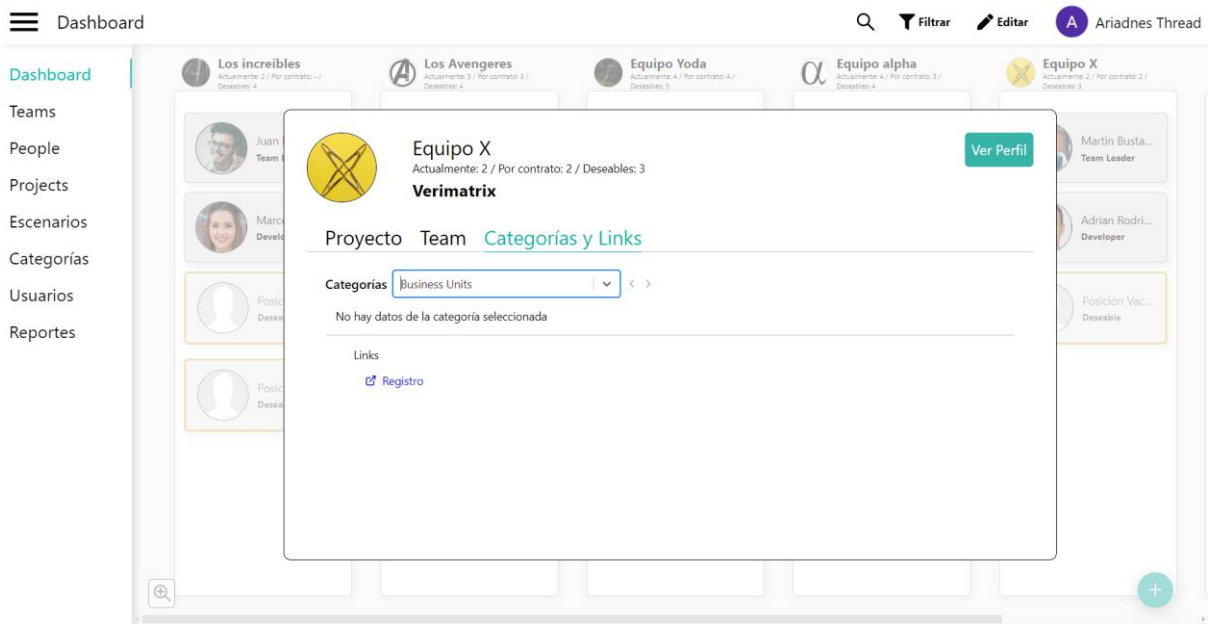


Figura 12.70: *Tab* de categorías y links en el modal de equipo en la versión 3.0.0

Se agregan los links a la antigua *tab* de categorías. Los links se utilizan para representar aspectos de interés en las entidades del sistema, como registros, notas, entre otros.

Modal de equipo, *tab* de proyecto.

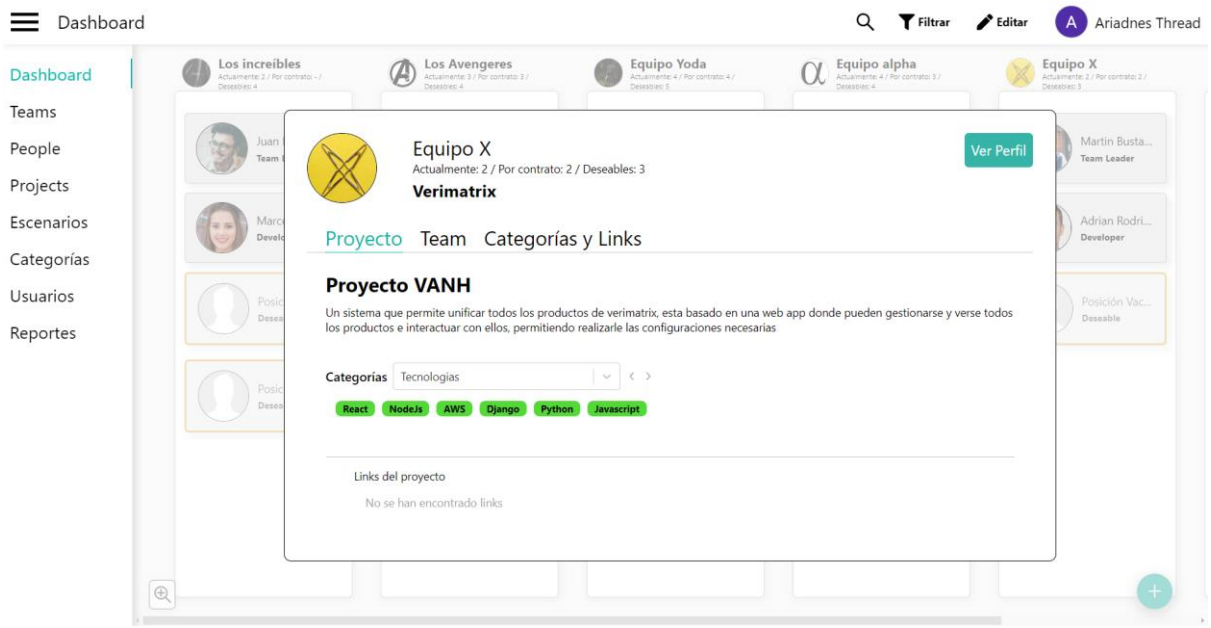


Figura 12.71: Tab de proyecto en la versión 3.0.0

A la *tab* de proyecto también se le agregan los *links* para su rápida visualización.

Visualización de equipo, *tab* principal

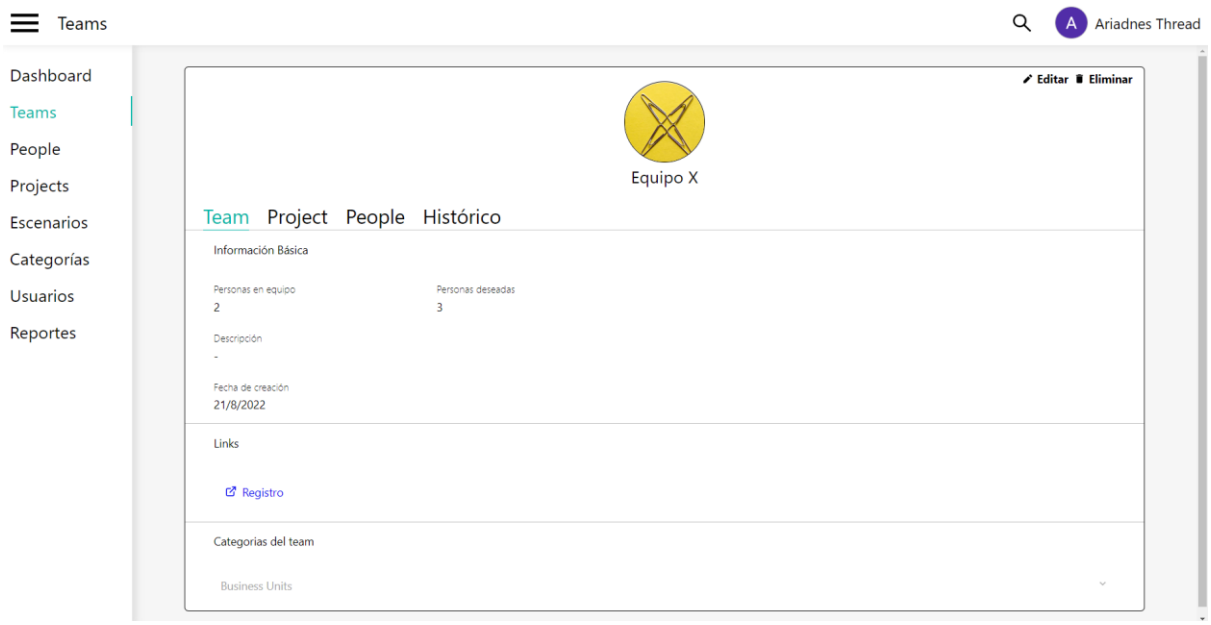


Figura 12.72: Visualización de equipo en la versión 3.0.0

La visualización de equipo presenta el agregado de los *links* (lo cual aplica para el resto de las entidades del sistema), además de una nueva *tab* llamada “Histórico”. Esta funcionalidad muestra el “proceso” que tuvo el equipo o persona en la empresa, como cuando entraron a un equipo, o cuando alguien salió de un equipo. Además, la *tab* de proyecto presenta sus links para visualizar.

Visualización de equipo, tab de histórico

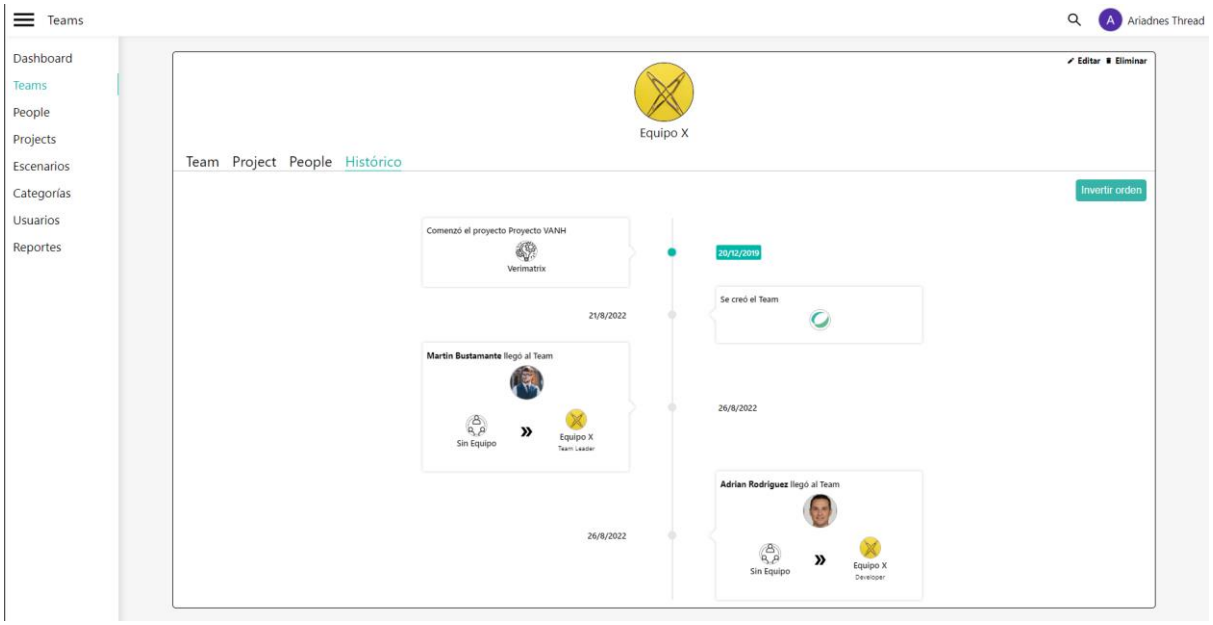


Figura 12.73: Tab de histórico en visualización de equipo en la versión 3.0.0

Se muestra el histórico del equipo en esta pestaña, mostrando la fecha de creación del equipo y los movimientos que sucedieron, ya sea entrada y salida de personas al equipo o cambios de roles. Está ordenado en base al tiempo, y se puede invertir el orden de los hitos de ser necesario.

Visualización de persona, *tab* principal

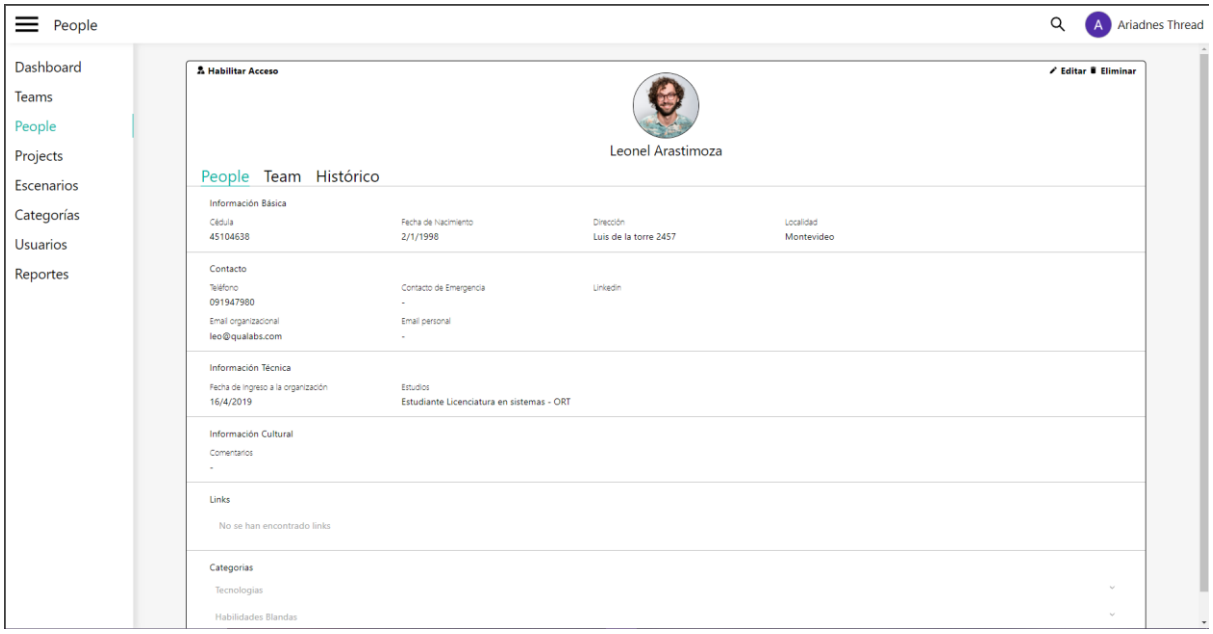


Figura 12.74: Visualización de empleado en la versión 3.0.0

Como en el caso de la visualización de equipo, se agregan los links, y una pestaña para el histórico de empleado. Además, presenta una nueva opción en la parte superior izquierda llamada “Habilitar Acceso” (o “Deshabilitar acceso” si se encuentra habilitado), que representa habilitar el acceso al sistema de la persona, es decir, entrar a su cuenta y hacer uso de ciertas funcionalidades del sistema. Para el caso de un empleado, este puede ver y editar partes de su perfil.

Se separó el *mail* original en dos partes, un *mail* personal, y un *mail* organizacional. Se usa este último para crear y dar acceso al sistema.

Visualización de persona, *tab* de histórico

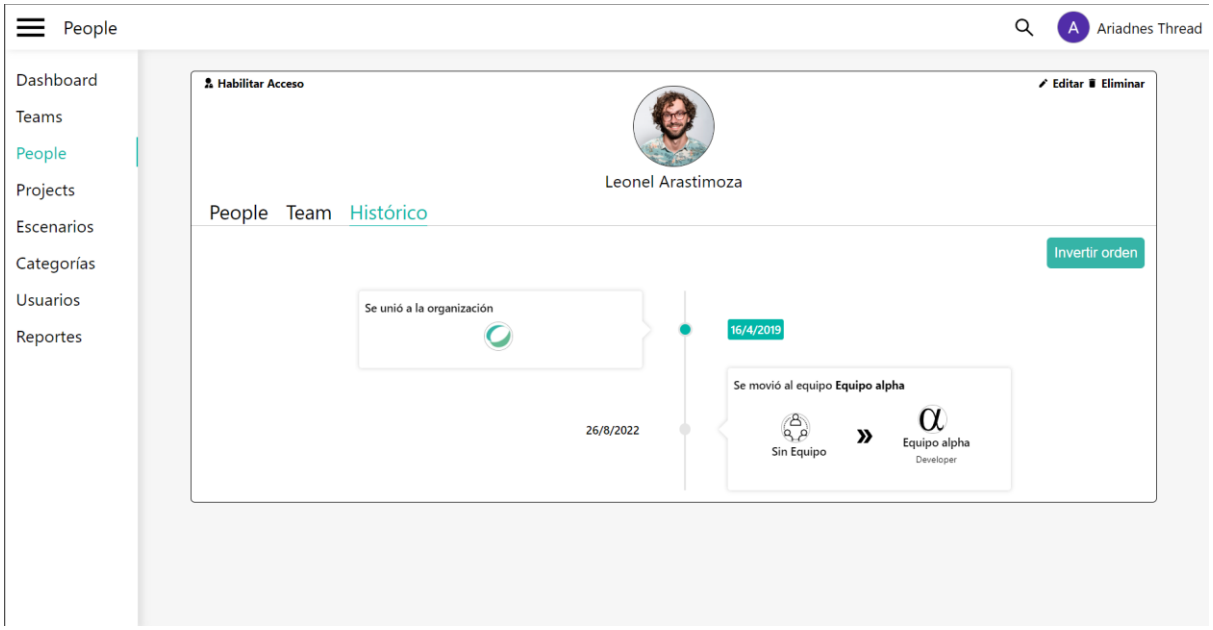


Figura 12.75: Histórico de empleado en la versión 3.0.0

Se muestra el histórico del empleado en esta pestaña, mostrando cuando se unió a la organización, y el pase de equipo a equipo que tuvo a lo largo del tiempo, ordenado en base a tiempo. Se puede invertir el orden de los hitos que sucedieron de ser requerido.

Creación de persona

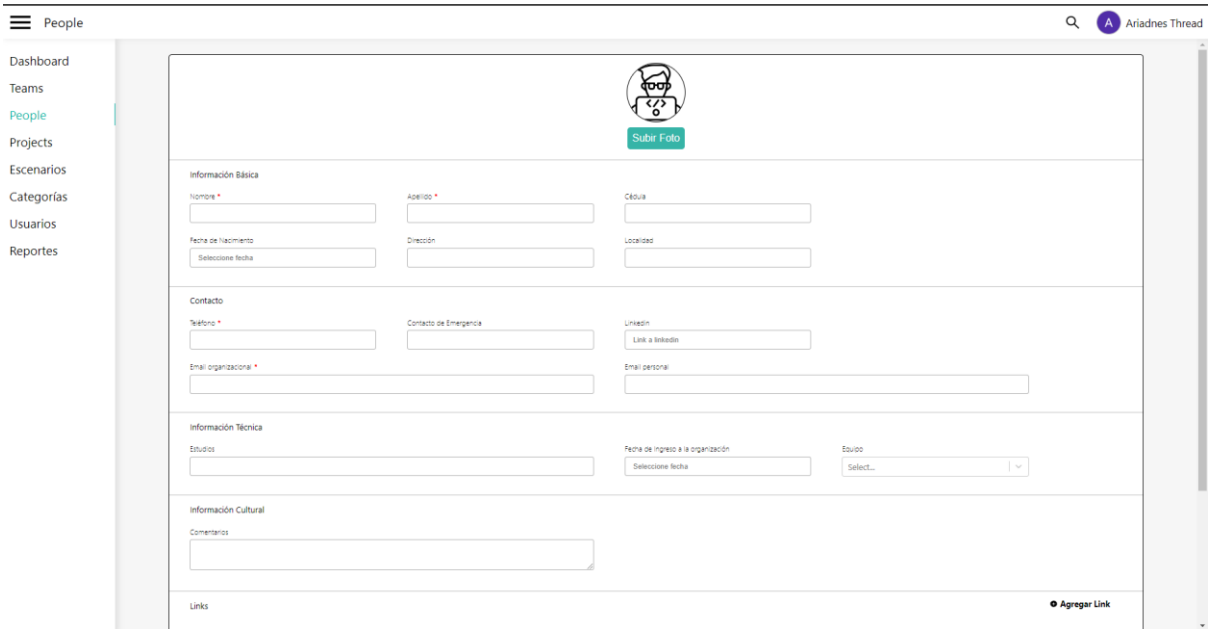


Figura 12.76: Creación de persona en la versión 3.0.0

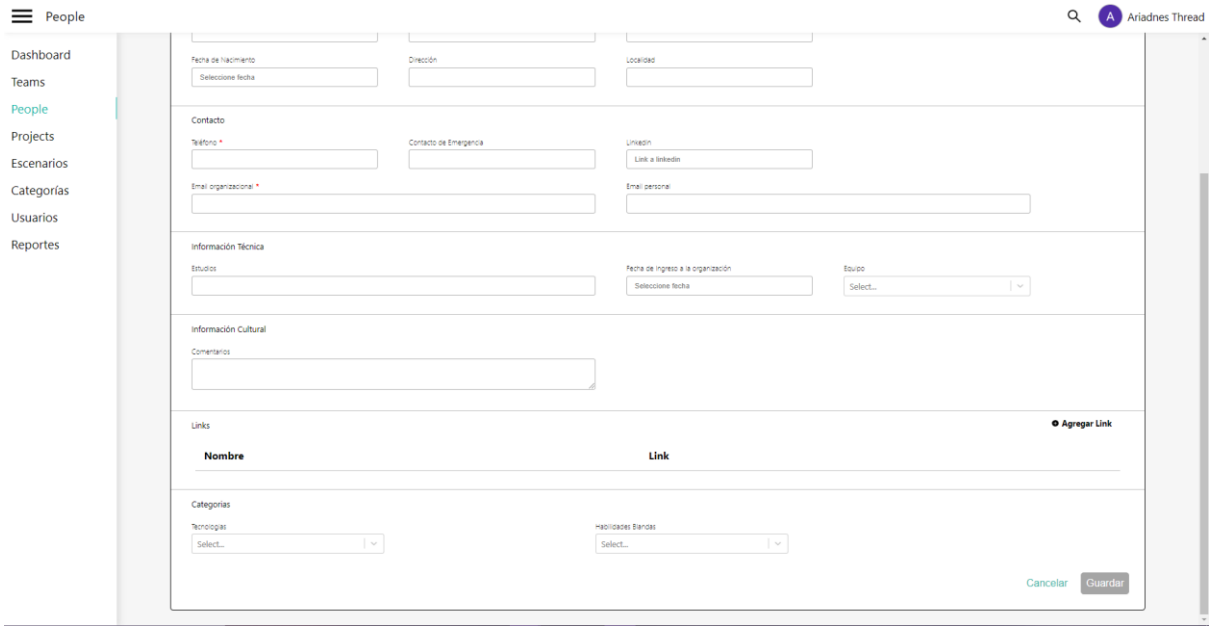
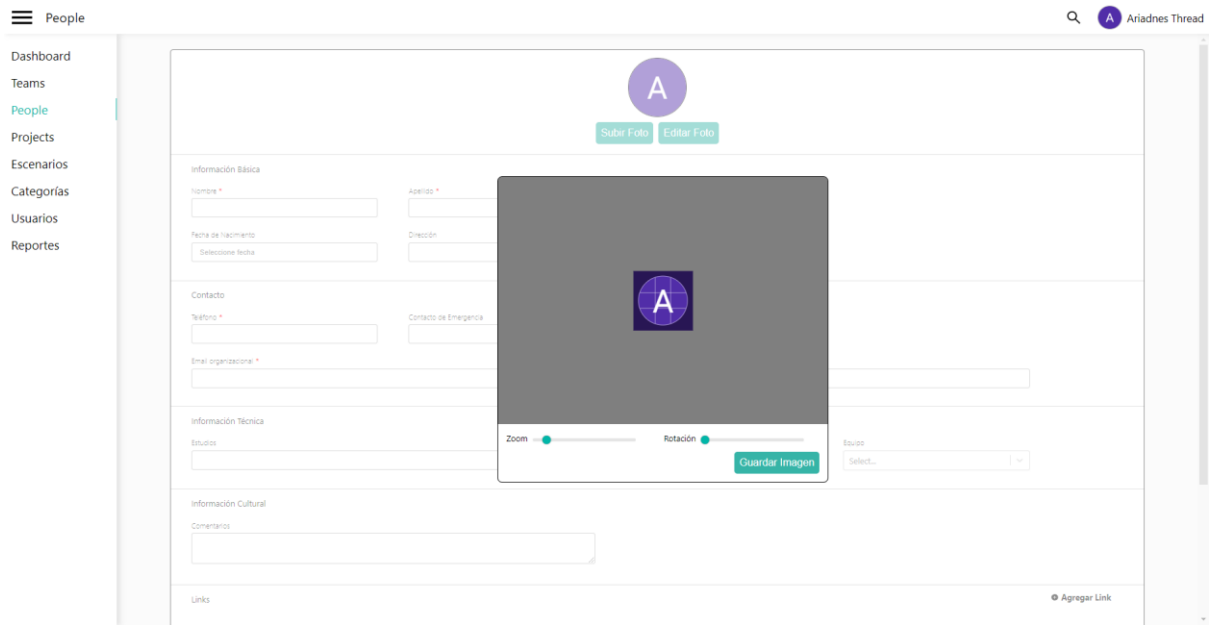


Figura 12.77: Creación de personas en la versión 3.0.0

Los formularios de creación de equipo, persona y proyecto fueron actualizados con una sección específica para el agregado de links, pero por temas de visualización solo se mostrará la creación de personas. Agregando y como ya se mencionó anteriormente, ahora existen dos campos de mail, uno requerido llamado mail organizacional (mail que se usará para entrar al sistema) y un mail opcional llamado mail personal.

Cambio en la subida de imágenes



Figuras 12.78: Cambio en la subida de imágenes en la versión 3.0.0

Anteriormente, la subida de imágenes era simple y no tenía ninguna funcionalidad como recortar la imagen o rotarla, sino que se subía como estaba en el *file system* utilizado. Se decidió modificarlo y permitir recortar, aumentar o disminuir o rotar la imagen para darle un gran componente de personalización a las imágenes que son utilizados en todas las partes del sistema.