Universidad ORT Uruguay

Facultad de Ingeniería

Tracker361: Plataforma de gestión y seguimiento de conversiones offline

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

Gonzalo Strauss - 213188

Itai Miller - 201244

Mauricio Pisabarro - 192095

Gonzalo Kurin - 192488

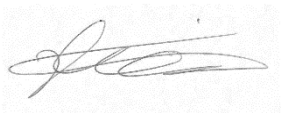
Tutor: Gastón Mousqués

2021

Declaración de autoría

Nosotros, Gonzalo Strauss, Itai Miller, Mauricio Pisabarro, Gonzalo Kurin, 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 de grado de la carrera de Ingeniería en Sistemas en la Universidad ORT Uruguay;
- 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 en conjunto 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.



Gonzalo Strauss
18 de marzo del 2021

Itai Miller
18 de marzo del 2021

Mauricio Pisabarro
18 de marzo del 2021

Gonzalo Kurin
18 de marzo del 2021

Agradecimientos

En primer lugar, queremos agradecerle a nuestro tutor Gastón Mousqués por el apoyo que nos ha brindado a lo largo de todo el proyecto, por su dedicación y compromiso con el equipo.

Asimismo, nos gustaría expresar nuestro agradecimiento a quienes trabajan en el laboratorio de *ORT Software Factory* por sus valiosos aportes durante las diferentes revisiones y reuniones.

Agradecemos también a todas las personas que nos brindaron su tiempo para ayudarnos y aportarnos *feedback* con el fin de obtener un mejor producto.

Este proyecto no hubiese sido posible sin la confianza de la empresa *Conecta361*, especialmente Marcelo Wilkorwsky, que nos planteó sus necesidades y confió en el equipo para poder satisfacerlas.

El proyecto de fin de carrera es el resultado de varios años de aprendizaje, en el que el apoyo incondicional de nuestras familias y amigos fue fundamental para llegar a cumplir el objetivo final de ser Ingenieros.

A todos ellos, muchas gracias.

Abstract

A pesar del constante y acelerado crecimiento del *ecommerce* en la última década, la mayoría de las ventas aún siguen produciéndose en locales físicos. De hecho, menos del 15% de estas ocurren *online*.

Mientras tanto, la publicidad digital continúa ganando terreno en el mundo del marketing con el auge de las redes sociales, generando que más del 80% de los compradores encuentren reseñas o anuncios del producto en internet previo a su compra en tienda. Este fenómeno, también conocido como *Research Online, Buy Offline (R.O.B.O* por siglas en inglés*),* implica que las empresas deban redoblar sus esfuerzos en rastrear las ventas producto de su inversión en publicidad para medir su verdadero impacto. Sin embargo, obtener y asociar estos datos no es tarea sencilla. De esta necesidad es que nace *"Tracker361".*

"Tracker361" es una plataforma de gestión y seguimiento de las conversiones producidas por ventas en tiendas físicas para la medición y optimización de las campañas publicitarias en internet. Tiene como objetivo simplificar y automatizar el proceso de atribución de dichas ventas al marketing *online*. Surge como un interés de negocio planteado por el cliente *Conecta361*, que se ha enfrentado con la necesidad de atribuir las ventas ocurridas en el mundo *offline* a las campañas publicitarias *online* de sus clientes. La misma proporciona una solución sencilla, moderna y altamente modificable a través de una aplicación web.

El proyecto se dividió en tres fases: investigación, desarrollo y documentación. La primera permitió entender las necesidades del cliente y profundizar sobre una temática desconocida para el equipo como lo era el marketing digital. En esta se realizaron pruebas de concepto que permitieron ampliar conocimientos y validar una futura solución. Durante la misma se utilizó la metodología *Kanban* debido a la incertidumbre de los requerimientos. La segunda fase consistió en el diseño, arquitectura y desarrollo del sistema, teniendo en cuenta la interoperabilidad y modificabilidad como los principales

atributos de calidad. Se utilizó como marco de gestión *Scrum* para el desarrollo ágil mediante iteraciones de una semana. A su vez, se implementaron las buenas prácticas de integración y despliegue continuo inspiradas en *DevOps*.

Como resultado del proyecto se obtuvo una solución capaz de automatizar los procesos manuales por parte del cliente, optimizando y complementando las soluciones ya existentes mediante la implementación de diferentes técnicas de conversiones *offline*.

El equipo incorporó conocimientos sobre el marketing digital. Además, aprendió sobre diferentes tecnologías, como *React*, infraestructuras *cloud* y un acercamiento a *DevOps*.

Hoy la empresa cuenta con el cliente Mundo Trabajo utilizando el servicio en producción. Se espera que en los próximos meses se ofrezca el servicio al resto de sus clientes. El equipo se lleva muchos aprendizajes fundamentales para la salida laboral como ingenieros en sistemas.

Palabras clave

Marketing Digital; Conversión; Conversion *Offline*; Campañas; Anuncios; *Retail*; Cuenta Publicitaria; Atribución; *Ads*; Ingeniería de *Software*; Arquitectura de *Software*; Metodologías Ágiles; Automatización de Procesos; *DevOps*.

Glosario

Administrador de anuncios: "El administrador de anuncios es una herramienta de Facebook que te permite crear y administrar tus anuncios de la plataforma. Puedes ver tus campañas y conjuntos de anuncios. Realizar cambios y consultar los resultados" [2].

Campaña publicitaria: "Una campaña publicitaria es una estrategia específicamente diseñada y ejecutada en diferentes medios para obtener objetivos de notoriedad, ventas y comunicación de una determinada marca, usando la publicidad" [3].

Conversión: "En marketing digital llamamos conversión a cada una de las acciones que realiza el cliente y que están alineadas con nuestros objetivos" [4].

Conversión *offline:* "La realización fuera de la web, por parte de un usuario o potencial cliente, de una determinada acción que nosotros tenemos definida" [5].

Cuenta publicitaria: "La cuenta publicitaria es aquella herramienta que nos permite administrar nuestros activos publicitarios" [6].

Atribución: Es el proceso por el cual un cliente llega a convertir [7].

Evento: Conjunto de datos de una venta. En esta se incluye datos del comprador, así como datos del producto.

Fuente de datos: Plataforma de donde se obtienen los eventos. Por ejemplo: *MercadoPago*, *Zoho*, *Mailchimp*.

Procesador: *API* de un administrador de anuncios. Por ejemplo: *Facebook Ads*, *Google Ads.*

Retorno de inversión: "El ROI (*Return on Investme*nt) es el valor económico generado como resultado de la realización de diferentes actividades de marketing. Con este dato podemos medir el rendimiento que hemos obtenido de una inversión" [8].

Web Scraping: "Técnica utilizada mediante programas de *software* para extraer información de sitios web" [9].

Tráfico*:* "Son todos aquellos usuarios que visitan una página *web*" [10].

Índice

Declaración de autoría ……..2

Agradecimientos ……..3

Abstract ……4

Palabras clave……6

Glosario ...……...7

Índice ...……9

1. Introducción 16

1.1 Descripción del cliente 19

1.2 Descripción del equipo 20

1.3 Objetivos.... 21

1.3.1 Objetivos académicos....21

1.3.2 Objetivos del producto 22

1.3.3 Objetivos del proyecto 22

1.4 Nombre, slogan y logo 23

1.5 Estructura del documento 25

2. Introducción al Marketing Digital 27

2.1 Marketing Tradicional vs Marketing Digital.... 27

2.2 Estrategias más utilizadas 29

2.3 Importancia del Marketing Digital 34

2.4 Administradores de anuncios.... 35

2.4.1 Facebook Ads....35

2.4.2 Google Ads 37
2.5 Resumen 39
3. Contexto, problema y solución 40
3.1 Contexto 40
3.2 El problema 43
3.2.1 Necesidades del cliente 44
3.3 Solución 48
3.3.1 Actores 56
3.4 Resumen 58
4. Marco metodológico 59
4.1 Características del proyecto 59
4.2 Roles 60
4.3 Relación con el cliente 61
4.4 Metodología y ciclo de vida 62
4.5 Fases del proyecto 63
4.5.1 Fase de investigación 64
4.5.2 Fase de desarrollo 65
4.5.3 Fase de documentación 66
4.6 Herramientas utilizadas 67
4.7 Resumen 67
5. Ingeniería de requerimientos 68
5.1 Proceso 68
5.1.1 Investigación 69
5.1.2 Técnicas de relevamiento de requerimientos 73

5.2 Listado de requerimientos 79
5.3 Requerimientos Funcionales 79
5.3.1 Aplicación Web 79
5.3.2 API 84
5.3.3 SDK 84
5.4 Requerimientos No Funcionales 84
5.4.1 Principales atributos de calidad 85
5.4.2 Otros atributos de calidad 87
5.5 Restricciones 87
6. Arquitectura y desarrollo 88
6.1 Requerimientos de arquitectura 89
6.2 Descripción de la arquitectura 92
6.2.1 Interoperabilidad 95
6.2.2 Modificabilidad 102
6.2.3 Disponibilidad 107
6.2.4 Seguridad 109
6.2.5 Testeabilidad 112
6.2.6 Otros atributos de calidad 113
6.3 Tecnologías 116
6.3.1 Backend 116
6.3.2 Frontend 117
6.3.3 Infraestructura 118
7. Gestión de proyecto 126
7.1 Fases e hitos del proyecto 126
7.2 Fase de investigación 126

7.2.1 Implementación de *Kanban* 127
7.2.2 Distribución de esfuerzo 129
7.3 Fase de desarrollo 130
7.3.1 Implementación de *Scrum* 130
7.3.2 Release plan 135
7.3.3 Métricas de gestión 137
7.4 Gestión de la comunicación 144
7.4.1 Comunicación entre el equipo 144
7.4.2 Comunicación con el cliente 144
7.4.3 Comunicación con el tutor y docentes de ORT 146
7.5 Gestión de riesgos 146
7.5.1 Metodología utilizada 146
7.5.2 Identificación de riesgos 148
7.5.3 Análisis cuantitativo 149
7.5.4 Control y seguimiento 151
7.6 Resumen 152
8. Automatización del proceso de ingeniería de software: un acercamiento a *DevOps* 154
8.1 DevOps 154
8.2 Integración y despliegue continuo 156
8.2.1 Investigación y construcción 157
8.2.2 Ejecución 162
8.3 Buenas prácticas utilizadas 168
8.4 Resumen 169
9. Gestión de la configuración 170

9.1 Elementos de configuración 170
9.1.1 Software 170
9.1.2 Librerías 171
9.1.3 Infraestructura 172
9.1.4 Documentación 174
9.2 Elección de herramientas 177
9.3 Metodología de trabajo 178
9.4 Resumen 181
10. Gestión de la Calidad 182
10.1 Objetivos de Calidad del Producto 182
10.2 Plan de Calidad 182
10.3 Aseguramiento de la Calidad 183
10.3.1 Aplicación de estándares 183
10.3.2 Capacitación en tecnologías 184
10.3.3 Revisiones 184
10.3.4 Retrospectivas 185
10.3.5 Verificaciones 185
10.3.6 Validaciones 185
10.3.7 Pruebas de Software 186
10.3.8 Automatización de pruebas 189
10.4 Gestión de incidentes 190
10.5 Métricas de calidad 191
10.5.1 Calidad de código 191
11. Conclusiones 194
11.1 Estado actual 194

11.2 Conclusiones del proyecto 195
11.2.1 Conclusiones de los objetivos académicos 195
11.2.2 Conclusiones de los objetivos del producto 196
11.2.3 Conclusiones de los objetivos del proyecto 197
11.3 Reflexiones y lecciones aprendidas 198
11.4 Próximos pasos 202
12. Referencias bibiliográficas 205
13. Anexos 223
13.1 Archivos de ventas de clientes de Conecta361 223
13.2 Formato y especificación de campos de *Facebook Ads* 225
13.3 Flujo de subida de archivo manual en *Facebook Ads* 226
13.4 *Wireframes, mockups* y prototipos iniciales 229
13.4.1 Wireframes 229
13.4.2 M*ockups* y Prototipos iniciales 230
13.5 Resumen de herramientas utilizadas 232
13.6 Justificación de herramientas utilizadas 235
13.7 Ingeniería Inversa 239
13.8 Evidencia de primeras investigaciones 242
13.9 Listado de principales temas de investigación 244
13.10 Evidencias de flujos de investigación de APIs y sistema 245
13.11 Product Backlog 246
13.12 API *Endpoints* 250
13.13 Análisis de hosting para el sistema 255
13.14 Evidencia de *feedback* recolectado en *Sprint Reviews* 257

13.15 Hitos principales del proyecto 259
13.16 Informes de revisiones formales de Universidad ORT Uruguay 261
13.16.1 Informe 1 de revisión ORTsf 261
13.16.2 Informe 2 de revisión ORTsf 262
13.16.3 Informe 3 de revisión ORTsf 263
13.17 Plan de riesgos 264
13.18 Seguimiento de principales riesgos del proyecto 271
13.19 Análisis y justificación de elección de herramienta de CI/CD 280
13.20 Elementos de configuración de template de desarrollo genérico 282
13.21 Evidencia de elementos de software 283
13.22 Evidencia de librerías 284
13.23 Evidencia de elementos de infraestructura 285
13.24 Evidencia de elementos de documentación 286
13.25 Gitflow 287
13.26 Evidencia de versionado 288
13.27 Sincronización de Github con Jira y CI 291
13.28 Plan de calidad 292
13.29 Estándares de codificación 295

1. Introducción

Conecta361 es una agencia de marketing digital uruguaya - en adelante el cliente - que se enfoca en hacer crecer negocios a través de medios digitales. Dentro de sus servicios más ofrecidos, se encuentran: desarrollo de *ecommerce* y sitios web, gestión de redes sociales, campañas de publicidad digital y consultorías [11].

Para las campañas digitales de sus clientes, *Conecta361* utiliza a *Facebook* y *Google Ads* como administradores de anuncios. A través de estos, *Conecta361* realiza estrategias de marketing y genera contenido publicitario de forma de hacer crecer las ventas y dar mayor visibilidad de marca a sus clientes. De manera de optimizar las campañas realizadas, los administradores de anuncios proveen estadísticas relacionadas a la rentabilidad de estas.

Dentro de las estadísticas que *Conecta361* quiere ofrecer a sus clientes, se encuentran la cantidad de ventas ocurridas en tiendas físicas debido a la influencia de un anuncio publicitario. A la detección de estas ventas se le llama atribución de conversiones *offline.*

Durante el 2019 el cliente encontró un mecanismo para poder asociar dichas ventas, mediante la utilización de los administradores de anuncios de *Facebook Business Manager* y *Google Ads*. Se encontró con la dificultad de que, si bien sus mecanismos funcionaban correctamente, estos no eran lo suficientemente escalables para el volumen de clientes a gestionar.

El trabajo era manual y requería de esfuerzo y tiempo para cumplir con cada una de las etapas del proceso necesario para registrar la atribución de las conversiones *offline.* En primer lugar, era necesario conseguir los datos de las ventas *offline* de cada cliente, y estos eran obtenidos a través de diferentes medios, como *e-mail* o W*hatsapp*, y en diversos formatos, como archivos de texto o imágenes. Luego, era necesario normalizar todas las transacciones en un único archivo de texto, conteniendo información de la venta e identificación del comprador. Por último, este era subido todos los días a través de

Facebook y *Google Ads*, quienes eran los encargados de identificar conexiones entre las transacciones y los anuncios.

Marcelo Wilkorwsky, CEO de *Conecta361*, expresó su deseo al equipo de desarrollar una nueva solución que sea escalable y que a su vez permita optimizar el rastreo de las ventas producidas en los canales *offline*.

El equipo, a su vez, buscaba un proyecto que permitiese aplicar los conocimientos adquiridos a lo largo de los cinco años de carrera con un desafío extra; tener un componente innovador, desafiante y a su vez desconocido que permita a los integrantes crecer e insertarse en otras facetas del mundo laboral.

Fue así como el equipo y el cliente comenzaron a transitar un camino continuo de investigación y desarrollo, surgiendo hoy "*Tracker361"*.

"Tracker361" es una plataforma de gestión y seguimiento de las conversiones producidas por ventas en tiendas físicas para la medición y optimización de las campañas publicitarias en internet. Su arquitectura favorece la interoperabilidad y modificabilidad, permitiendo comunicarse con los distintos procesadores, fácilmente agregando nuevas fuentes de datos y modificar o extender las existentes.

El sistema incluye un *backoffice web* tanto para los clientes como para administradores de *Conecta361*. También se cuenta con un *backend* que integra a todas las partes del sistema, conectando tanto a servicios propios y externos. Además, se ofrece una *Software Development Kit* (*SDK* por sus siglas en inglés*) y* una *Application Programming Interface* (API por sus siglas en inglés) que permite a los clientes integrarse fácilmente a la plataforma de manera agnóstica a los cambios que puedan ocurrir internamente.

El *backoffice* permite agregar nuevos administradores de anuncios y vincularlos a las cuentas publicitarias de *Facebook* y *Google Ads*. Además, se pueden habilitar o

deshabilitar fuentes de datos, como *MercadoPago*[1], para la obtención de las ventas y compradores en tiempo real. También se permite importar los datos de los compradores mediante los diferentes formatos de archivos de *CRMs* (*Zoho*[2] y *Mailchimp*[3] *por ejemplo*). Luego del procesamiento de los eventos, el sistema permite visualizar las estadísticas de las conversiones producidas de manera *offline* de los clientes y segmentaciones de estas según fuente de datos, fecha o proveedor, entre otros. Este subsistema puede ser accedido por administradores o clientes de *Conecta361*, con diferentes privilegios según el rol.

El *backend* contiene la lógica de negocio y almacenamiento de datos, integrando a los diferentes subsistemas externos e internos. Es el encargado de recopilar los datos de los compradores, procesar los eventos, enviarlos a los diferentes procesadores para luego detectar y asignar conversiones adecuadamente. Este interactúa con las diferentes API's de los administradores de anuncios de manera de esconder las complejidades particulares de cada una de ellas.

El desarrollo del *backend* fue realizado mediante *Java Spring*, mientras que el *frontend* se hizo con *React* y *Typescript.* A su vez, los sistemas utilizaron a los servicios *cloud* de *Amazon Web Services* como *Infrastructure As A Service* (IaaS por sus siglas en inglés).

Debido a la incertidumbre del dominio del problema y requerimientos cambiantes, el equipo decidió utilizar metodologías agiles para llevar a cabo la gestión del proyecto. Las

[1] https://www.mercadopago.com.uy/
[2] https://www.zoho.com/es-xl/
[3] https://mailchimp.com/

mismas fueron *Kanban* y *Scrum*, profundizando las mismas en capítulos posteriores. También se adoptaron prácticas que permitieron acercarse a la metodología de *DevOps* como lo son la integración y entrega continua, automatización y control de versiones.

Hoy en día, la plataforma ya se encuentra en producción y puede ser accedida a través del siguiente enlace: https://app.conecta361.tk/

1.1 Descripción del cliente

Conecta361 es una agencia de marketing digital uruguaya enfocada en el crecimiento de los negocios a través de medios digitales. Trabajan para pequeñas y medianas empresas con un único objetivo: hacer crecer las ventas con la mínima inversión y la máxima eficiencia [12]. El mismo está conformado por un equipo chico de siete personas, sin embargo, hoy en día cuentan con más de cien clientes, siendo la mayoría de estos uruguayos y otros de diferentes países de América Latina.

Marcelo Wilkorwsky - *product owner* del proyecto – es experto y consultor en marketing digital. Emprendedor desde hace más de siete años y fundador de *Oincs*, tiene experiencia en dar iniciativa a proyectos innovadores y desafiantes [13]. Se destaca por sus aptitudes en estrategias empresariales y *Social Media Marketing (SMM* por sus siglas en inglés*)* [14] [15].

Es importante destacar que el equipo se sintió muy cómodo trabajando con el cliente ya que en todo momento del proyecto se mostró muy predispuesto a ayudar y responder todas las dudas que iban surgiendo. Ayudó al equipo a entender la problemática existente y dio flexibilidad para explorar y presentar nuevas ideas.

1.2 Descripción del equipo

Conformado por cuatro estudiantes de la carrera de Ingeniería en Sistemas, los integrantes del equipo poseen habilidades complementarias que permiten potenciar al mismo.

- Itai Miller: Es desarrollador backend en *Directa24*. Anteriormente trabajó en una *start-up* llamada *"Spotlike"* que le permitió conocer a expertos del área del marketing digital. Se define como una persona responsable, detallista y estructurada.
- Gonzalo Strauss: Es desarrollador *backend* en *Directa24*. Tiene la ventaja de haber comenzado a trabajar en los comienzos de la carrera, brindándole otras herramientas ajenas a lo académico. Se reconoce como una persona independiente, creativa y vanguardista.
- Gonzalo Kurin: Trabaja como desarrollador *full stack* en *Attica Labs*. En este y en su anterior trabajo – también como desarrollador – siempre ha tenido la responsabilidad de estar en contacto directo con clientes. Es una persona racional, mediadora y que le gusta trabajar en equipo.
- Mauricio Pisabarro: Es profesor en *ORT* en la catedra de teoría de la computación. Recientemente ha tenido ofertas laborales en *Google, Facebook* y *Microsoft*. Es una persona estudiosa, perseverante y que trabaja en base a resultados.
- Gastón Mousques: jefe del departamento de Ingeniería de *Software* y docente de varios cursos del cual los demás integrantes fueron parte. Tiene una amplia trayectoria en el ámbito académico tutorando proyectos de excelencia. El equipo lo define como una persona altamente predispuesta, exigente y apasionado por la enseñanza.

A lo largo de la carrera, los integrantes trabajaron juntos en diferentes oportunidades, distinguiendo las fortalezas y debilidades de cada uno.

1.3 Objetivos

Definir los objetivos ayudó a dirigir y alinear los esfuerzos hacia lo que se quería conseguir. Ayudó a evaluar si los resultados obtenidos al finalizar el proyecto cumplían con los objetivos iniciales. Los mismos fueron divididos en tres categorías:

1.3.1 Objetivos académicos

Aplicar los conocimientos adquiridos durante la carrera

Durante la carrera el equipo se nutrió de distintos conceptos, herramientas y buenas prácticas que fueron aplicadas en diversas instancias con la finalidad de que el proyecto culmine de manera exitosa.

Aprender nuevas tecnologías y herramientas

En una industria tan cambiante como la tecnológica, es clave mantenerse actualizado con las nuevas tendencias que emergen del mercado. Por esta misma razón el equipo fijó como meta el aprendizaje y uso de algunas de estas que sean desconocidas para los integrantes.

Producto de gran complejidad

El equipo se propuso elegir un proyecto que sea desafiante a nivel tecnológico y requiera de satisfacer necesidades reales. La complejidad debe ser afrontada buscando y aplicando las metodologías más optimas respecto a la naturaleza del proyecto.

1.3.2 Objetivos del producto

Acortar la brecha entre las ventas *offline* y el marketing digital

Poder obtener estadísticas y medir las conversiones *offline* a partir de publicidad en campañas digitales, mediante la aplicación de diferentes mecanismos de detección de conversiones *offline.*

Agregar valor a los clientes mediante la optimización de su presupuesto publicitario

Que el cliente pueda obtener más datos a la hora de medir la rentabilidad de sus campañas y analizar su efectividad. Esto es, calcular el verdadero retorno de inversión de las campañas digitales para dar datos más exactos del rendimiento de las mismas a los clientes de *Conecta361*.

Automatizar procesos que permitan la detección de ventas físicas producidas a través de campañas digitales

Para poder agilizar los procesos de detección de conversiones *offline*, el equipo se propuso utilizar diferentes herramientas y API's que permitan automatizar procesos humanos y obtener de manera rápida y eficaz dichas métricas.

1.3.3 Objetivos del proyecto

Autogestión del equipo

Los integrantes del equipo no tenían experiencia previa en gestionar y liderar un proyecto por sí mismos. Aprender a trabajar y organizarse en forma autónoma fue motivo para poder aumentar el rendimiento y maximizar su potencial.

Cumplir con el alcance acordado

El equipo se planteó cumplir con todos los requerimientos funcionales marcados como requeridos en el proyecto e intentar incluir la mayor cantidad de estos con prioridad baja. A su vez, cumplir con la totalidad de los requerimientos no funcionales.

Lograr un producto que sea puesto en producción antes de la entrega final

Uno de los objetivos más desafiantes que se planteó el equipo fue la construcción de un sistema que sea puesto en producción antes de la entrega final.

1.4 Nombre, slogan y logo

El equipo tuvo como desafío crear un nombre y logo del producto como una estrategia de marca de *Conecta361*. Este último solicitó que el nombre sea atractivo, corto y fácil de pronunciar. A su vez, el logo debía ser diseñado de modo que se respeten los colores actuales y que sea sencillo relacionar el producto con la misma.

Ilustración 1 - Logo de Conecta361

Con estas indicaciones, el equipo utilizó la estrategia de *brainstorming* para pensar palabras o frases que pudiesen cobrar sentido con el nombre, logo y *slogan* actual de la empresa. Entre las candidatas, se destacan "*The conversion machine*", "*Data Tracker*", "*Tracker361*", "*TrackerBridge*" y "*OfflineTracker*". Por unanimidad el nombre elegido fue "*Tracker361*". Incluir en el nombre el número 361 permitió darle una directa relación al producto con la empresa. El nombre *tracker*, a su vez, fue utilizado dada su simpleza y significado con el objetivo del producto.

Posterior a la elección del nombre, el equipo se reunió con los integrantes del departamento de marketing de *Conecta361* con la finalidad de validar el nombre y buscar

un slogan de acuerdo con la elección. De allí surgió el siguiente *slogan*: “Conectando tu negocio en más que una vuelta”, siendo este un juego de palabras con el *slogan* anterior.

Por último, se definió un icono que acompañe al nombre y *slogan* de modo de captar la atención del público y reforzar aún más la identidad de marca de la empresa. La intersección entre los aros grafica el objetivo del producto: reducir la brecha entre el mundo online y offline. A su vez, el mismo es consistente con el icono de la empresa.

Ilustración 3 - Icono de Conecta361

Ilustración 2 - Icono de Tracker361

Finalmente, los logos principales y alternativos del producto son respectivamente:

Ilustración 4 - Logo de Tracker361

Ilustración 5 - Logo alternativo de Tracker361

1.5 Estructura del documento

A continuación, se resume el contenido de cada capítulo del presente documento.

Introducción

Este capítulo introduce un panorama general del proyecto; su origen, equipo, cliente y objetivos planteados.

Introducción al marketing digital

En este capítulo se introducen conceptos básicos del marketing digital destacando el impacto e importancia que tiene en la actualidad.

Contexto, problema y solución

En este capítulo se presenta el contexto actual, problema planteado por el cliente y se describe la solución de esta junto con los principales actores del sistema.

Marco metodológico

En este capítulo se definen las características del proyecto, los roles de cada integrante, el ciclo de vida utilizado y las distintas fases del proyecto. También se detallan las metodologías y herramientas de apoyo.

Ingeniería de requerimientos

En este capítulo se detalla el proceso utilizado por el equipo para investigar, analizar y relevar los requerimientos funcionales y no funcionales del sistema.

Arquitectura y desarrollo

En este capítulo se especifican los principales desafíos de la arquitectura y los atributos de calidad. Se describe el diseño de la arquitectura, las principales decisiones y las tecnologías utilizadas.

Gestión de proyecto

En este capítulo se detallan las distintas fases e hitos del proyecto y las metodologías utilizadas durante el mismo junto con sus adaptaciones. Se describe como fue la gestión del proyecto y las métricas recolectadas.

Automatización de los procesos de ingeniería de software: un acercamiento a DevOps

En este capítulo se detallan las buenas prácticas utilizadas para automatizar el proceso de la ingeniería de software y su importancia dado los requerimientos del proyecto. Se hace especial énfasis en la implementación de la integración y entrega continua.

Gestión de la configuración

En este capítulo se detallan los elementos de configuración, elección de herramientas, organización de los repositorios, metodología de trabajo y gestión de versionado.

Gestión de calidad

En este capítulo se describen las diferentes actividades para asegurar la calidad del proyecto. Se detallan los objetivos de producto y proceso a nivel de calidad. Se destaca la utilización de estándares y buenas prácticas y las métricas obtenidas.

Conclusiones y lecciones aprendidas

En este capítulo se presentan las conclusiones obtenidas del proyecto y se reflexiona sobre las lecciones aprendidas durante el transcurso del mismo. También se detalla los próximos pasos a seguir.

2. Introducción al Marketing Digital

Este capítulo busca introducir al lector en algunos conceptos básicos del marketing digital que permitirán reforzar los conocimientos generales del área, siendo estos relevantes para entender el dominio del problema a resolver. Se explica su objetivo e importancia en la actualidad, las ventajas frente al marketing tradicional y las plataformas y tendencias más utilizadas para hacer y gestionar publicidad digital.

El marketing es el conjunto de actividades que tienen como objetivo satisfacer al consumidor mediante un producto o servicio, con un beneficio empresarial de por medio [16]. Estas actividades podrían ser incrementar las ventas de la empresa, así como también potenciar la marca. Tiene como objetivo el conocer y comprender tan bien al consumidor que el producto se ajuste perfectamente a sus necesidades [17].

2.1 Marketing Tradicional vs Marketing Digital

Existen dos tipos de marketing; tradicional y digital. El primero refiere al método de publicidad que las empresas utilizaron desde el comienzo del marketing (mediados del siglo XX) para comercializar su producto [18]. Incluye anuncios impresos en periódicos o revistas, así como también en la radio y televisión [19]. El segundo refiere a la promoción de productos o marcas a través de una o más formas de canales digitales. En este se incluye al internet y sus derivados: redes sociales, emails, videos o aplicaciones móviles [20].

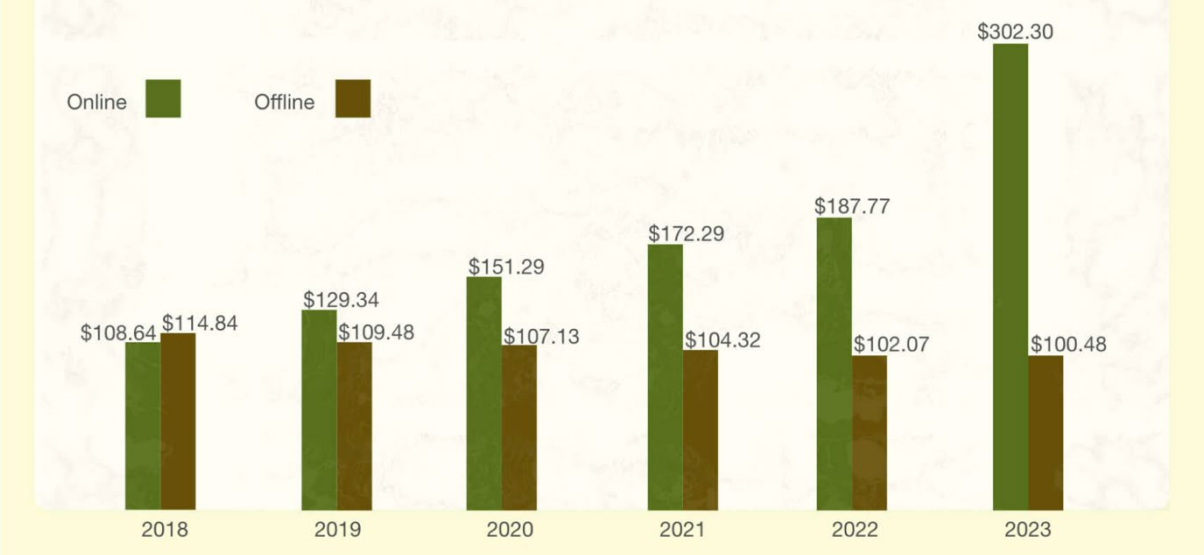
A continuación, se detalla una tabla comparativa para conocer cuáles son las principales diferencias entre ambas:

	Marketing tradicional	Marketing digital
Disponibilidad horaria	Limitada	Continua (24/7)
Poder de expansión	Expansión geográfica limitada	Expansión mundial
Presencia en el mercado	Mercado local	Mercado global
Necesidad de capital por campaña.	Mayor capital por campaña	Menor capital por campaña
Dificultad para medir los resultados de las campañas	Alta	Baja
Tipo de comunicación con el cliente	Comunicación unidireccional y lineal	Comunicación bidireccional e interactiva
Feedback recibido del cliente	Poco *feedback*	Más *feedback*
Dificultad de modificar una campaña en función de resultados	Alta	Baja

Tabla 1 - Marketing Tradicional vs Marketing Online [21]

Tal como se puede visualizar en la tabla anterior, el marketing digital aventaja al tradicional en aspectos como el costo, alcance y herramientas de métricas. Es por esta razón que cada vez más comercios incluyen dentro de sus estrategias de marketing alguna forma de publicidad *online*. De hecho, desde el año 2019 la inversión en marketing digital sobrepasó a la del tradicional, y se estima que la brecha se agigante de manera exponencial.

Este comportamiento se puede visualizar en la siguiente gráfica, en donde se observa el gasto en billones de ambos canales:



Gráfica 1 - Gasto de inversión, Publicidad Tradicional vs Digital [139]

2.2 Estrategias más utilizadas

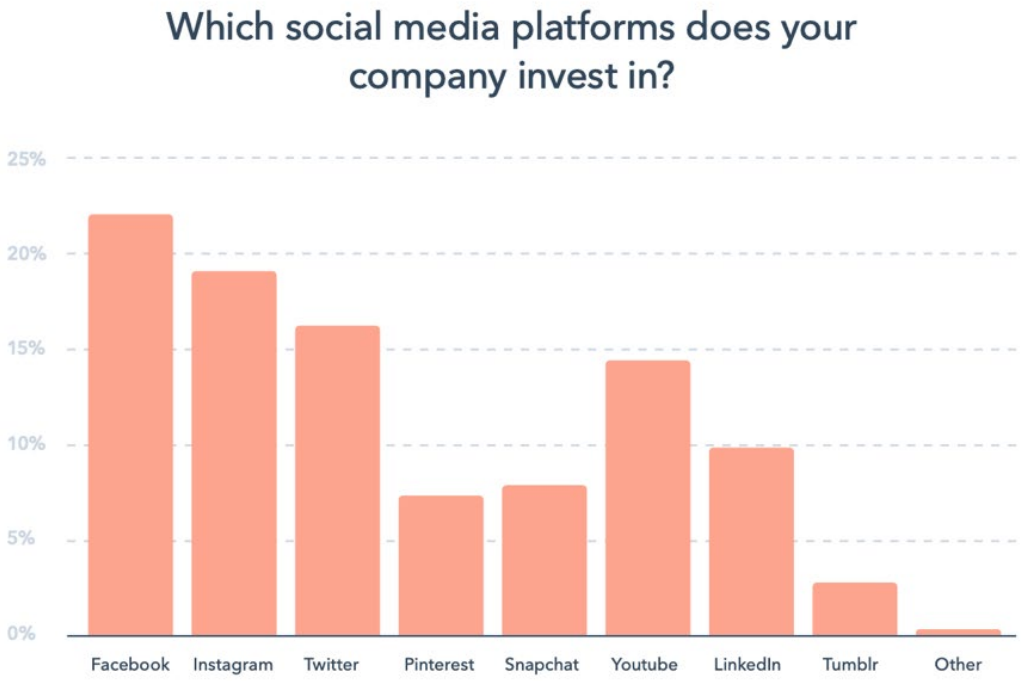
En la actualidad, existen más de diez estrategias de marketing digital [22]. Una de las más utilizadas hoy en día es la denominada marketing de redes sociales [23]. Con la irrupción de las mismas, las empresas se vieron forzadas a adaptar sus canales de comunicación para obtener la manera más eficaz y rápida de llegar a los usuarios. Esta estrategia representa al conjunto de acciones y objetivos que lleva a cabo una empresa, para promover sus productos y/o servicios y construir un vínculo con su público objetivo con la ayuda de las redes sociales [14]. La misma permite una comunicación interactiva entre la marca y las personas. Cada red social presenta sus propias características y las marcas deben entender cuál utilizar dependiendo de sus objetivos.

Según una encuesta realizada por Crystal King publicada en febrero de 2020, el *Social Media Markerting (*SMM por sus siglas en inglés*)* es utilizada como estrategia de marketing digital por el 74% de las compañías en el mundo.

Algunos de los puntos que sostienen su popularidad son [24]:

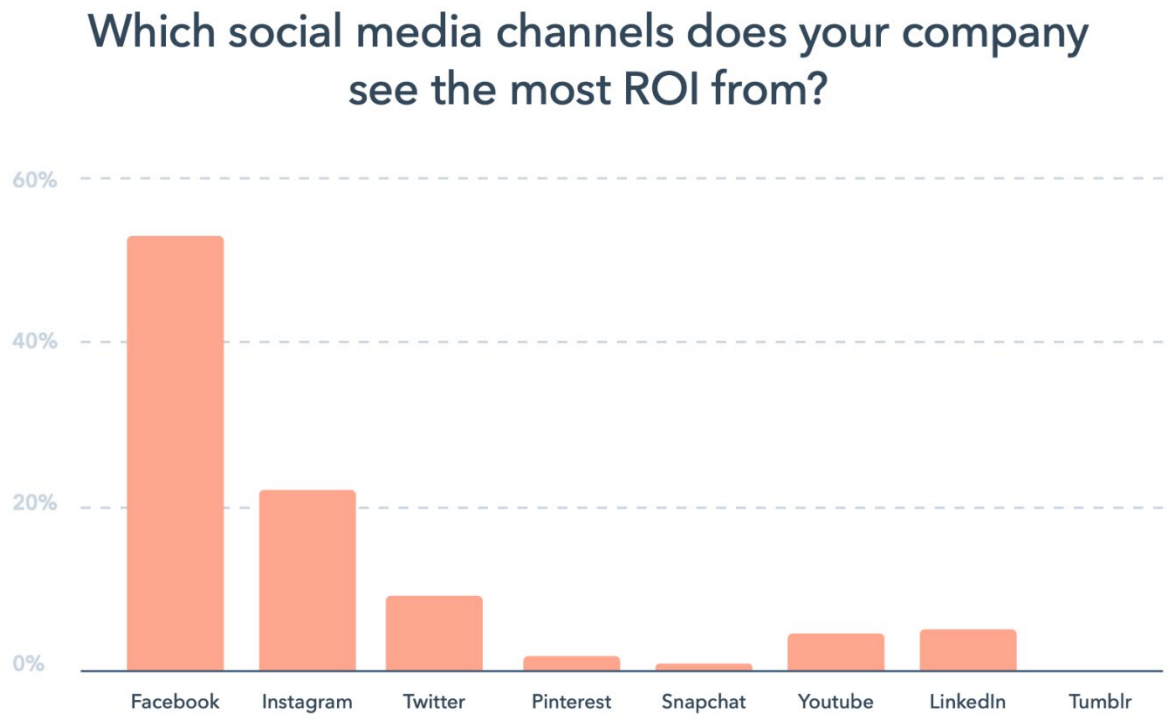
- Es más medible que otro tipo de marketing
- Es *real-time,* instantáneo.
- Es fácil personalizarlo.
- Si puede ser viral, será viral.
- Soporta todos los esfuerzos de *marketing* posibles.

Según un estudio realizado por *Hubspot* entre noviembre y diciembre de 2019 [25], las principales redes sociales en las cuales las compañías deciden invertir son las siguientes:



Gráfica 2 - Fuente: HubSpot Research, Global Survey, Nov - Dic 2019

De esta se desprende que *Facebook* e *Instagram* son las dos redes sociales preferidas para invertir en publicidad. Además, ambas se encuentran en el podio de las redes sociales con mayor retorno de inversión en el año 2020, tal como se puede visualizar en la siguiente gráfica [25]:



Gráfica 3 - Fuente: HubSpot Research, Global Survey, Nov - Dic 2019

Del *Social Media Marketing* es que se desprende la metodología denominada Atraer, Convertir y Transformar (ACT) [26]. Esta consta de los tres componentes mencionados en su nombre que serán detallados:

Atraer: Llevar visitas al sitio web. Para esto, es clave tener un buen posicionamiento orgánico en los buscadores (SEO por sus siglas en inglés), realizar publicidad en internet (SEM por sus siglas en inglés) y tener presencia en medios sociales como pueden ser Facebook e Instagram [27] [28].

Convertir: Convertir a un extraño que visita el sitio web en un consumidor o cliente. En esta segunda etapa se busca cumplir con el objetivo propuesto. En marketing digital se habla de conversión cuando se consigue que otra persona realice una acción que la empresa desea. Una conversión no tiene por qué ser una venta. También podría ser subscribirse al blog o al canal de YouTube, dar "me gusta" en Facebook, dejar un comentario en Instagram, entre otros.

En la siguiente imagen se pueden visualizar dos flujos con el propósito de entender los diferentes tipos de conversiones. El primero corresponde a una conversión online mientras que el segundo a uno offline.

Se describirán cada uno de los flujos mediante escenarios de ejemplo con la finalidad de entender que es una conversión.

Ilustración 6 - Flujos de conversión *online* y *offline*

Escenario de conversión online

1. Juan está navegando en *Facebook*.

2. Juan ve un anuncio de una campera *Adidas* en *Facebook*.

3. A Juan le gusta la campera y decide hacer click en el anuncio para realizar su compra.

4. Juan es redirigido al *ecommerce* de *Adidas* para que efectúe la compra.

5. Juan compra la campera.

Como resultado, se logró que a través de un anuncio publicitario en internet Juan logre comprar una campera a través del sitio web de *Adidas*, generando una conversión *online*.

Escenario de conversión offline

1. Juan está navegando en *Facebook*.

2. Juan ve un anuncio de una campera *Adidas* en *Facebook.*

3. A Juan le gusta la campera y decide hacer clic en el anuncio.

4. Juan se dirige a la tienda física de *Adidas* para efectuar la compra.

5. Juan compra la campera.

Como resultado, se logró que a través de un anuncio publicitario en internet Juan logre comprar una campera a través del local físico de Adidas, generando una conversión *offline*.

Transformar: Hacer que los consumidores pasen a ser clientes. Esta se basa en estimularlos a que visiten la página y efectúen la compra, Utilizando los éxitos del pasado y presente de la empresa para atraer más éxito. Entre las estrategias más utilizadas para transformar se encuentran realizar testimonios de clientes satisfechos con sus compras o mostrar la trayectoria y éxito de la empresa.

2.3 Importancia del Marketing Digital

A continuación, se mencionan los principales puntos para destacar del impacto y posición que ocupa el marketing digital en la actualidad [29].

Visualización de la marca

Estar presentes en internet es fundamental ya que hoy en día son cada vez más las personas que buscan el producto/servicio en internet antes de realizar la compra.

Captar clientes

Con el marketing digital, es posible atraer y captar clientes potenciales.

Medición de resultados

Medir los resultados de las campañas es mucho más sencillo hacerlo de forma digital que de forma online ya que hoy en día existen muchas herramientas que ayudan a la misma. Una frecuentemente utilizada en el mercado es *Google Analytics.*

Alcance

Al utilizar internet y redes sociales como canal, permite tener un gran alcance a sus seguidores y potenciales clientes, así como también posicionar mejor la marca.

Bajo costo de inversión

el costo de inversión en marketing digital es menor que en marketing tradicional. Esto hace que pequeñas y medianas empresas tengan la oportunidad de invertir en campañas.

Crear una comunidad

El marketing digital permite crear una comunidad en redes sociales haciendo que las personas interactúen con la marca.

2.4 Administradores de anuncios

En lo que a marketing se refiere, hoy en día existen muchas plataformas que ayudan a publicitarse digitalmente. A continuación, se realizará una breve descripción de las más importantes y relevantes para el dominio del proyecto.

2.4.1 Facebook Ads

Facebook cuenta con un administrador de anuncios, cuyo objetivo es brindar el punto de partida a la publicidad realizada en *Facebook, Instagram* y cualquier otra red social de esta. Con esta herramienta se pueden crean anuncios, administrar cuando y donde se van a poner en circulación y realizar un seguimiento del rendimiento de las campañas en un solo lugar.

El administrador de *Facebook,* también conocido como *Facebook Business Manager,* constituye una herramienta muy eficaz para administrar todos los aspectos de la publicidad, y está pensado para anunciantes con cualquier grado de experiencia.

Su funcionamiento se basa en los siguientes pasos. Primero se debe crear la campaña publicitaria con un objetivo específico para luego crear los anuncios a promocionar. Para cada anuncio, se ingresa la inversión a realizar y la duración del mismo. Luego, se publica la campaña y con el correr de los días se puede ir optimizando la misma a partir de los resultados que se van obteniendo [30].

En la siguiente ilustración se visualiza una pantalla de ejemplo del panel administrativo de *Facebook Business Manager,* en donde se puede observar la creación de una campaña y el objetivo de la misma, así como la visualización de uno de sus anuncios en un teléfono celular:

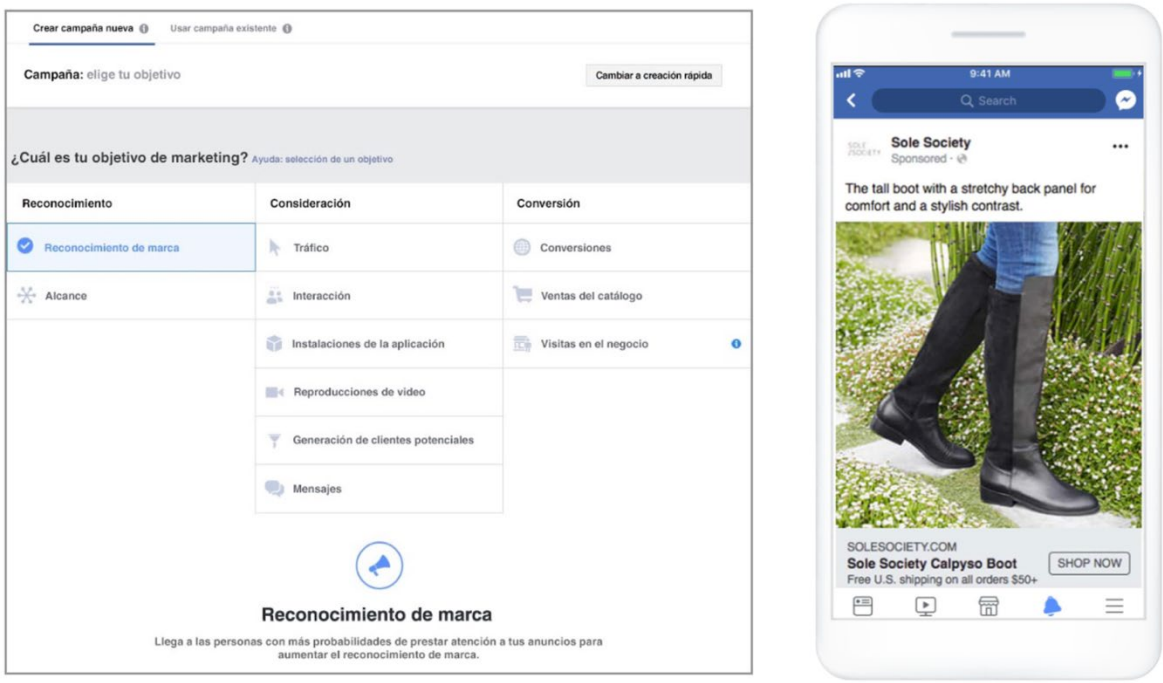


Ilustración 7 - Facebook Ads

Utilizar el administrador de anuncios de la red social con más usuario en el mundo trae los siguientes beneficios [31] [32]:

- Posibilidad de segmentar y ajustar los anuncios publicitarios de manera detallada.
- Medición de todas las métricas relevantes en un solo panel consolidado.
- Análisis de los resultados y llevar a cabo modificaciones con fundamento del historial de anuncios y campañas realizadas. Gracias a estos informes se da la posibilidad de optimizar cada vez más la campaña e incrementar su rendimiento.
- Es muy económica ya que solo se paga por clicks obtenidos.
- Viralización absoluta de los anuncios.

2.4.2 Google Ads

Google Ads es la red publicitaria de *Google*. La misma, está dividida en varias redes y se detallan a continuación [33]:

Red de búsqueda

Son los anuncios que aparecen al realizar una búsqueda en *Google*. Se los reconoce porque tienen asignada la etiqueta "anuncio". Este tipo de red es una de las más utilizadas por las empresas ya que arroja muy buenos resultados cuando el consumidor busca el producto con la intención de compra. A continuación, se muestra un ejemplo de este tipo de publicidad [33].

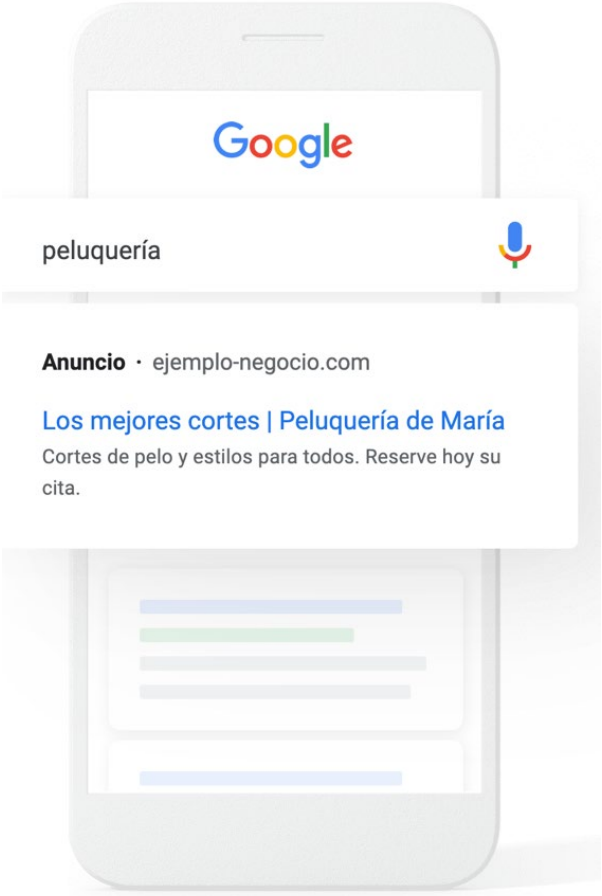


Ilustración 8 - Red de búsqueda en Google

Red de *display*

Son los anuncios que aparecen en portales asociados a *Google* o en propiedades de *Google* como son *Youtube* o *Gmail*. Se utiliza este tipo de publicidad cuando el anunciante tiene el objetivo de llegar a más personas [33]. En la siguiente imagen se puede visualizar un anuncio *display* que hizo *BBVA* en *Youtube*.

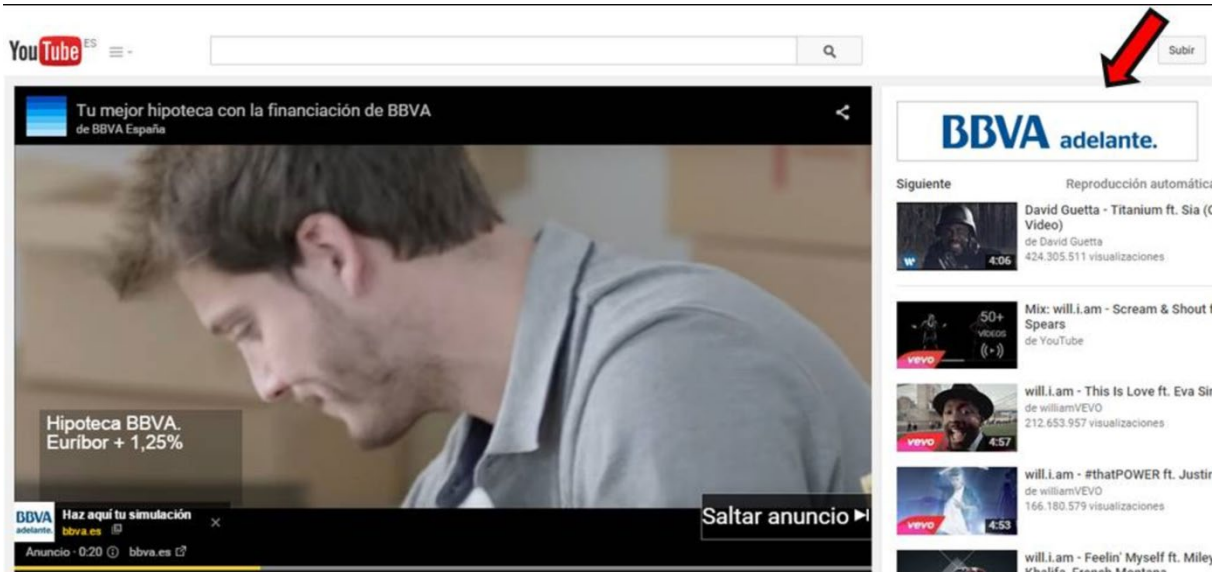


Ilustración 9 - Red de *display* en Google

Su funcionamiento es bastante complejo. Algunas de las variables que se configuran para optimizar la visibilidad de los anuncios son: geografía para llegar a las personas que están en el área deseada, sexo, edad, presupuesto deseado, duración, entre otros.

Algunos de los beneficios de utilizar *Google Ads* como plataforma de anuncios son:

- Segmentación de anuncios, permitiendo dirigir los anuncios a personas con intereses específicos y mostrar aquellos relevantes para estos.
- Control de costos, dando libertad para elegir cuanto invertir por día, mes o anuncio. Solo se paga cuando alguien realiza un click en el anuncio.
- Medición del éxito, realizando un seguimiento a los anuncios visualizando en cuales de estos hicieron click y en cuáles no.

- Proporciona herramientas para administrar las campañas y monitorear fácilmente los resultados.

2.5 Resumen

El crecimiento de la inversión en publicidad digital ha ido creciendo de manera acelerada en la última década, en gran parte gracias a la irrupción de las redes sociales. Entre las estrategias más populares para realizar marketing, se encuentra el *Social Media Marketing,* que es medible, instantáneo y personalizado. De esta se desprende la metodología "Atraer, Convertir, Transformar", en donde los administradores de anuncios de *Facebook* y *Google Ads* permiten crear y administrar campañas digitales detectando conversiones atribuidas a la influencia de estas.

Todos estos conceptos introductorios al marketing fueron fundamentales para facilitar el entendimiento del proyecto. Si bien el campo del marketing abarca una extensa variedad de temáticas y áreas, reconocer sus principales aristas permitió un mayor desempeño y desarrollo del proyecto, mitigando posibles riesgos para que el mismo no fracase.

3. Contexto, problema y solución

Una persona se encuentra con una publicidad de un auto para comprar en *Facebook*. Luego de buscar al auto en *Google*, visitar el sitio web y darle "Me Gusta" a la publicación en *Instagram*, decide dirigirse a la automotora dispuesto a comprar el mismo. Se acerca al mostrador, paga con su tarjeta de crédito a través de *MercadoPago* y se retira.

Al finalizar el mes, la automotora decide evaluar que tan rentables están siendo sus campañas publicitarias. A pesar de crear contenido digital, visualizar buen volumen de tráfico en su sitio y recibir consultas y llamadas telefónicas a diario, los administradores de anuncios de *Facebook* y *Google* muestran un 0% de ventas debido a las campañas publicitarias. Con el paso del tiempo, se da cuenta del problema y se enfrenta con el siguiente desafío:

¿Cómo lograr rastrear cuales de las ventas concretadas en su local se debieron a su inversión en publicidad digital?

Por consiguiente, este capítulo se centra en presentar los conceptos necesarios del *retail* y el impacto del marketing digital en la actualidad, contextualizando al lector sobre los desafíos que el dueño de la automotora presenta. Se profundizará sobre el problema en cuestión y posteriormente se detallará su solución.

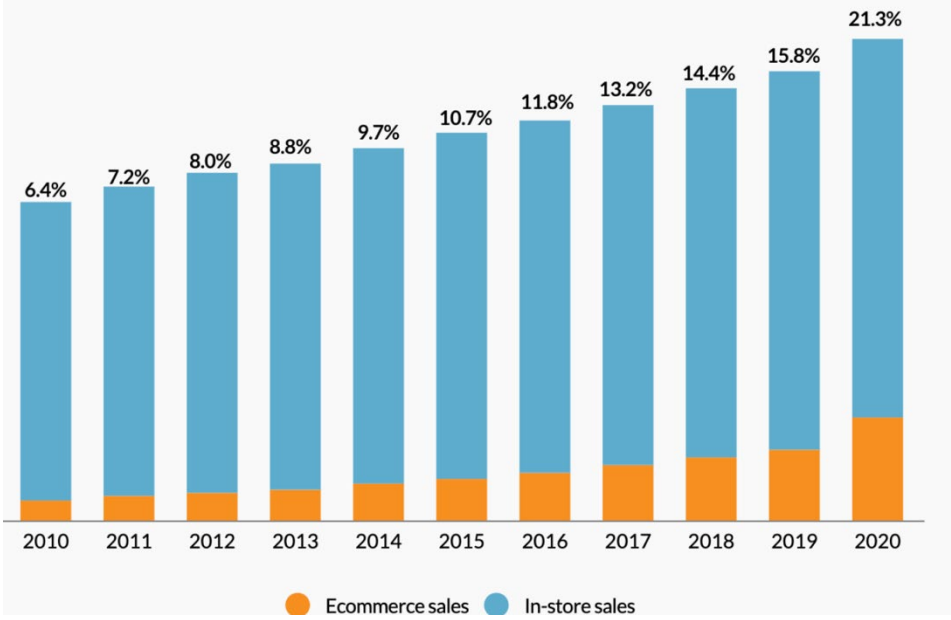
3.1 Contexto

El *retail* es un tipo de comercio que se caracteriza por vender al por menor [34] . Este sector engloba a las empresas que tienen como objetivo llegar a una gran cantidad de consumidores por medio de un stock masivo de productos o servicios. Entre los ejemplos más comunes se encuentran los centros comerciales, supermercados, farmacias, librerías, restaurantes, tiendas de conveniencia y derivados [35]. El *retail* no solo se asocia a un local físico, sino que estos también ocurren desde otros canales como el internet – denominado *eretailer* o *ecommerce* – o mediante un sistema hibrido el cual

combina tanto tiendas físicas como tiendas *online* [36]. Por ejemplo, un local de electrodomésticos que vende sus artículos en su tienda física y a través de *Mercado Libre* es considerado un *retailer* hibrido.

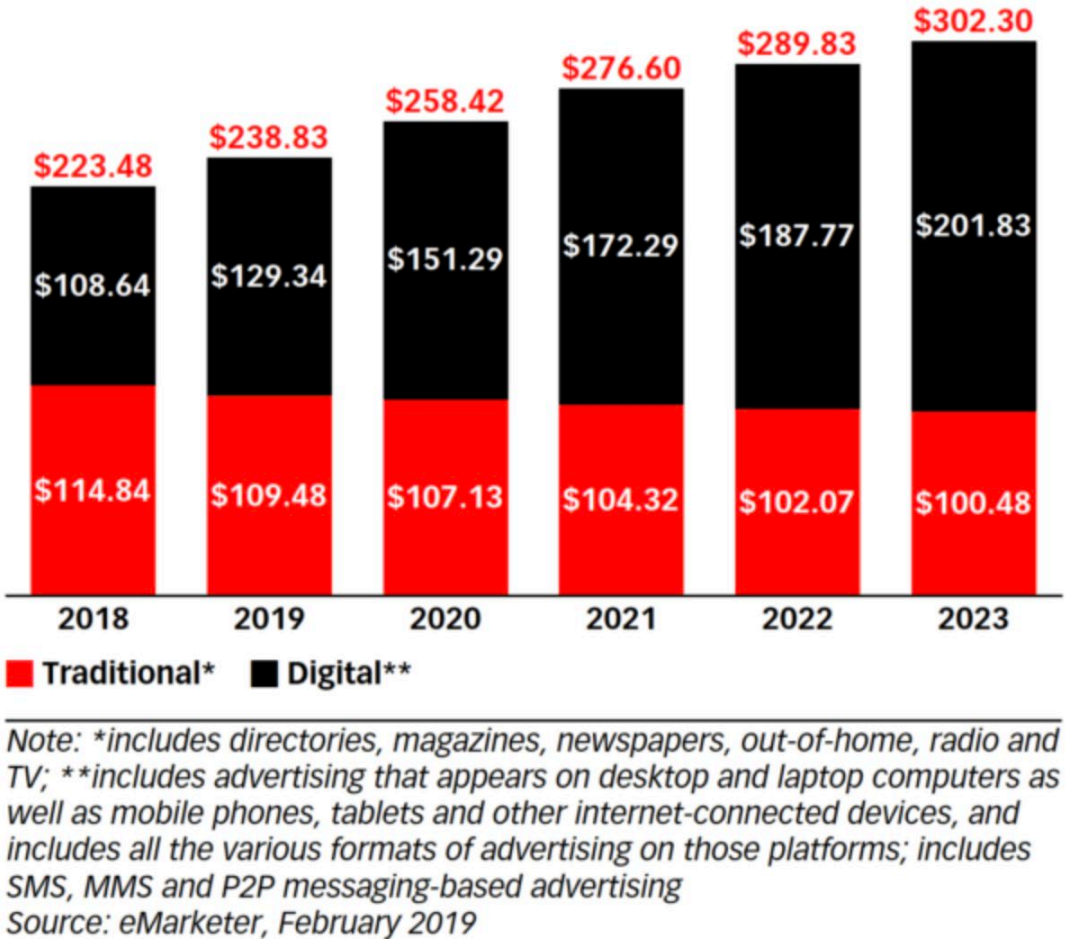
En el año 1991 surge el *World Wide Web* (WWW por sus siglas en inglés) y con la irrupción de las tecnologías del Internet empresas como *Amazon*, *Ebay* o *MercadoLibre* no tardaron en consolidar el negocio *del ecommerce* como un complemento e incluso sustituto viable y rentable al *retail* físico o tradicional a comienzos del siglo XXI [37]. Hoy en día la transformación digital y la globalización permiten que millones de comercios ofrezcan sus productos y servicios a través de medios digitales. Sin embargo, la amplia mayoría de los *retailers* aún mantienen sus locales físicos como principal canal de venta [38]. Beneficios exclusivos de los locales físicos como ver, tocar, probar y obtener de manera instantánea la mercancía, son algunas de las explicaciones de la vigencia de dicho canal [39].

Tomando en cuenta el crecimiento de las ventas *retail* por canal en los últimos ocho años, se puede notar que, si bien el comercio electrónico está creciendo de manera acelerada y constante, casi el 80% de las mismas se siguen produciendo en locales físicos [40].



Gráfica 4 - Ecommerce vs In-store sales [40]

En contraparte y considerando la última década, se puede observar como la publicidad *online* fue creciendo año tras año mientras que la publicidad *offline* fue perdiendo volumen de inversión. De hecho, en el año 2018 se dio un punto de inflexión cuando el fenómeno digital se aventajo frente al tradicional en inversión en billones, tal como se visualiza en siguiente gráfico.



Gráfica 5 - Inversión publicitaria Tradicional vs Digital [139]

A pesar del acelerado y constante crecimiento del ecommerce, las ventas físicas no están muertas, ni mucho menos. Se estima que en ocho de cada diez compradores encuentran reseñas o anuncios del producto en internet previo a realizar compras y el 90% de ellas se concretan de manera *offline*. Solamente un 10% de los compradores finalizan su compra en internet luego de ver un producto. De hecho, The ROBO Economy (Research

Online, Buy Offline), reveló en 2020 que se gastaron casi cinco dólares en la tienda por cada dólar gastado en línea [41] [42] [43].

A modo de ejemplo, se visualiza un escenario común del fenómeno R.O.B.O:

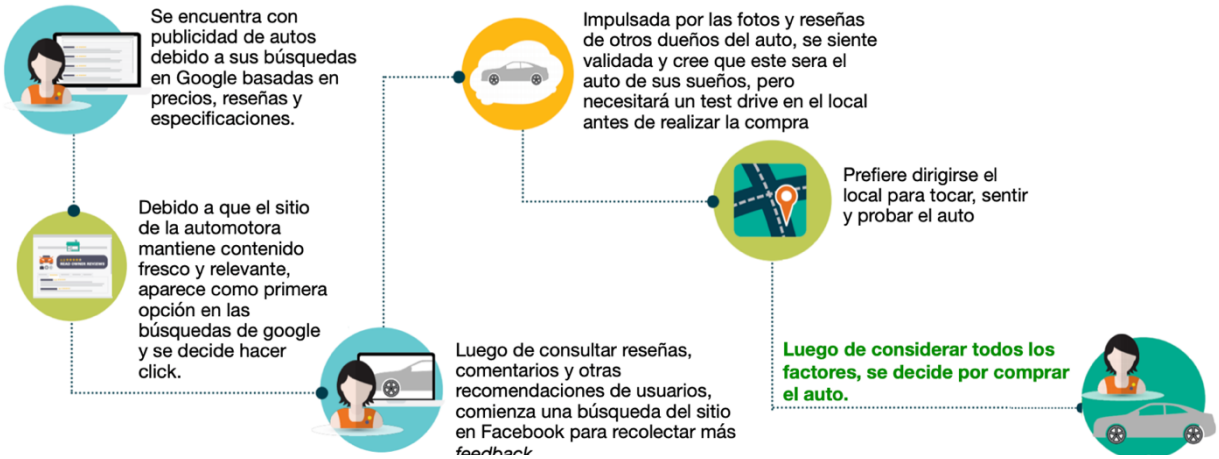


Ilustración 10 - Escenario de *"The Robo Economy"*

3.2 El problema

"Sé que la mitad de mis dólares invertidos en publicidad están siendo malgastados… Solo que no se cual mitad" – John Wannamaker [44].

Este hombre de negocios y pionero de la publicidad de comienzos de 1900 se enfrentó con el desafío de atribuir los esfuerzos del marketing y las ventas generadas. Hoy, más de un siglo después, las agencias de marketing atraviesan el mismo problema.

Las herramientas de publicidad en la actualidad, como *Facebook* y *Google*, permiten de manera fácil y rápida detectar las ventas online mediante sofisticados sistemas de atribución. Sin embargo, para aquellos negocios con gran porcentaje de sus ventas ocurridas en tiendas físicas, la forma de rastrear cuales de ellas se deben a sus inversiones en campañas digitales se dificulta.

Al igual que la automotora, sigue habiendo muchos negocios cuyo grueso de ventas tiene lugar en establecimientos físicos o por vía telefónica. Estos son negocios que:

- Disponen de puntos de venta físicos para la comercialización de sus productos (automotora).
- Recurren activamente al uso de *call-centers.*
- Combinan las ventas *online* y *offline.*

Las consecuencias de permitir tal fenómeno pueden ser determinantes en establecer el presupuesto. Estadísticas en base a únicamente conversiones *online* pueden hacerle creer a una empresa que su inversión en publicidad no es suficiente, o viceversa. Si un negocio no es capaz de atribuir fácilmente sus campañas a todas sus conversiones, ya sean *online* u *offline,* entonces estará haciendo *marketing* a ciegas [45].

La automotora que obtuvo un 0% de conversiones, pensará que sus campañas no están rindiendo de la manera esperable. Sin embargo, tomando en cuenta las conversiones *offline*, se tendría un panorama más completo y real del comportamiento de sus usuarios.

3.2.1 Necesidades del cliente

Muchos de los clientes de *Conecta361* se enfrentaban con el mismo problema que el de la automotora. En otras palabras, sus ventas producidas en locales físicos nunca eran rastreadas a un canal *online*, generando que las estadísticas de sus campañas no sean precisas.

Conecta361, como agencia de marketing digital, quería ayudar a sus clientes a optimizar sus campañas digitales.

Como primer acercamiento a una solución, el cliente se encontró con que las herramientas publicitarias de *Facebook* y *Google* permitían, desde el 2016, monitorizar las transacciones que tienen lugar fuera de Internet y relacionarlas con anuncios que el comprador haya visto previamente.

El sistema cruza los datos correspondientes a la transacción con los informes de las campañas de anuncios online. Mediante la utilización de una serie de datos de identificación, se logra establecer coincidencias, de forma de saber que clientes *offline* se han visto influenciados por las campañas *online*.

Ilustración 11 - Detección de conversión *offline*

Si bien esta solución resolvía de manera funcional el problema de la automotora, le causaba otros tantos a *Conecta361*.

El primer obstáculo enfrentado fue poder obtener las ventas *offline* de cada uno de sus clientes de forma frecuente. Por un lado, muchos de sus clientes no sabían cómo extraer sus ventas, y en caso positivo, no todos guardaban los datos de la misma forma. Algunos los tenían en bases de datos, otros utilizaban sistemas de facturación electrónica, planillas *Excel*, *CRMs cloud* y puntos de venta (POS por sus siglas en inglés). Incluso, algunos no tenían ninguno de estos, o tenían todos (véase Anexo 13.1).

Si todo iba bien, como difícilmente ocurría, se presentaba el próximo obstáculo. *Facebook*, por ejemplo, solicitaba incluir una serie de campos que son listados a continuación y pueden verse en más detalle en el Anexo 13.2:

- Descriptores de la transacción: Fecha de creación, valor, moneda, número de identificación de la orden, entre otros.
- Identificadores del comprador: Nombre y apellidos, número de teléfono, email, fecha de nacimiento, cuidad y hasta otros 17 datos distintos.

Al conseguir estos, y con la dificultad que esto conllevaba, *Conecta361* no sabía cómo leerlos ni cómo interpretarlos. Los nombres de los campos variaban o no eran nemotécnicos, algunos archivos estaban corruptos o no eran compatibles con su sistema operativo, y muchos otros archivos eran excesivamente pesados para abrirlos.

Pero si por alguna razón y luego de completados todos los pasos mencionados, estos eran exitosos, *Conecta361* se enfrentaba al último y gran obstáculo: la subida manual del archivo. Este paso consistía en la normalización de los datos de las ventas en un único formato admitido por la plataforma. De cada uno de los archivos obtenidos de los clientes, se debía realizar otro con el formato adecuado. Esta subida contaba con varios pasos y validaciones requeridas que muchas veces eran omitidas y retrasaban a toda la operación. En muchos casos los eventos a procesar no contaban con un identificador único, una fecha de creación u otro tipo de campos requeridos por los administradores de anuncios. El flujo completo se detalla en el Anexo 13.3.

Hay que recordar que el cliente estaba conformado por un equipo pequeño, y que el tiempo de sus recursos era valioso. El esfuerzo humano y manual que requería limitaba el tiempo para realizar las otras actividades indispensables de la agencia. Además, todo este proceso, agotador y friccionado, debía de hacerse todos los días, y aun así, tampoco se aseguraba el éxito de la gestión.

De comprender este proceso es que se desprenden las principales necesidades de *Conecta361:*

- Poder obtener, entender e interpretar los datos de las ventas de sus clientes, sin importar su formato o modo de almacenamiento.
- Poder traducir los campos de las ventas de sus clientes a los requeridos por los administradores de anuncios.
- Poder realizar el proceso de una manera más ágil y menos engorrosa que la demandada por las subidas de archivo manual.
- Poder reducir las complejidades y requerimientos específicos de cada administrador de anuncios.

De todas estas necesidades se encuentra un común denominador: la automatización de los procesos de detección de conversiones *offline.*

Las complejidades de estos procesos son conocidos por *Facebook y Google*, y es en base a este motivo que ofrecen una alternativa a la manual. Estas ofrecen una API que sustituye la subida de los archivos a las respectivas plataformas. Si bien esta es una buena solución en comparación con el proceso rudimentario realizado por *Conecta361*, requieren de conocimiento técnico del dominio. Además, automatizar los procesos mediante estas API's conllevan desafíos. Por ejemplo, la API de *Facebook* es un servicio inmenso, no solo para lo relacionado al *marketing*, sino que para todos los productos ofrecidos por esta. Entender las funcionalidades provistas lleva tiempo, por los conceptos y por sus dificultades técnicas. Estos son muy cuidadosos en proteger su seguridad, y requieren de muchas validaciones y especificaciones relacionadas a esta.

Debido a que la API de conversiones *offline* es relativamente moderna, esta es actualizada de forma frecuente, muchas veces eliminando las versiones anteriores. Además, es necesario aclarar que en esta se envían datos personales de usuarios reales, por lo que es común que se agreguen validaciones extra a las ya solicitadas anteriormente.

Comprendiendo el problema de la automotora y las necesidades planteadas por el cliente, es que el equipo se dispuso a implementar una solución que satisfaga a todas las partes.

3.3 Solución

En primer lugar, *Conecta361* debe poder gestionar a todos sus clientes en único panel consolidado. Este maneja las cuentas publicitarias de los mismos mediante los administradores de anuncios de *Facebook* y *Google*. Cada cliente es único, tiene sus propias necesidades, su propio público y su estrategia de realizar publicidad. Por tal razón, las configuraciones realizadas a cada uno de estos varían y deben ser tratados de forma independiente. El sistema les permite a los administradores de *Conecta361* visualizar y configurar a los clientes como se muestra en la siguiente ilustración:

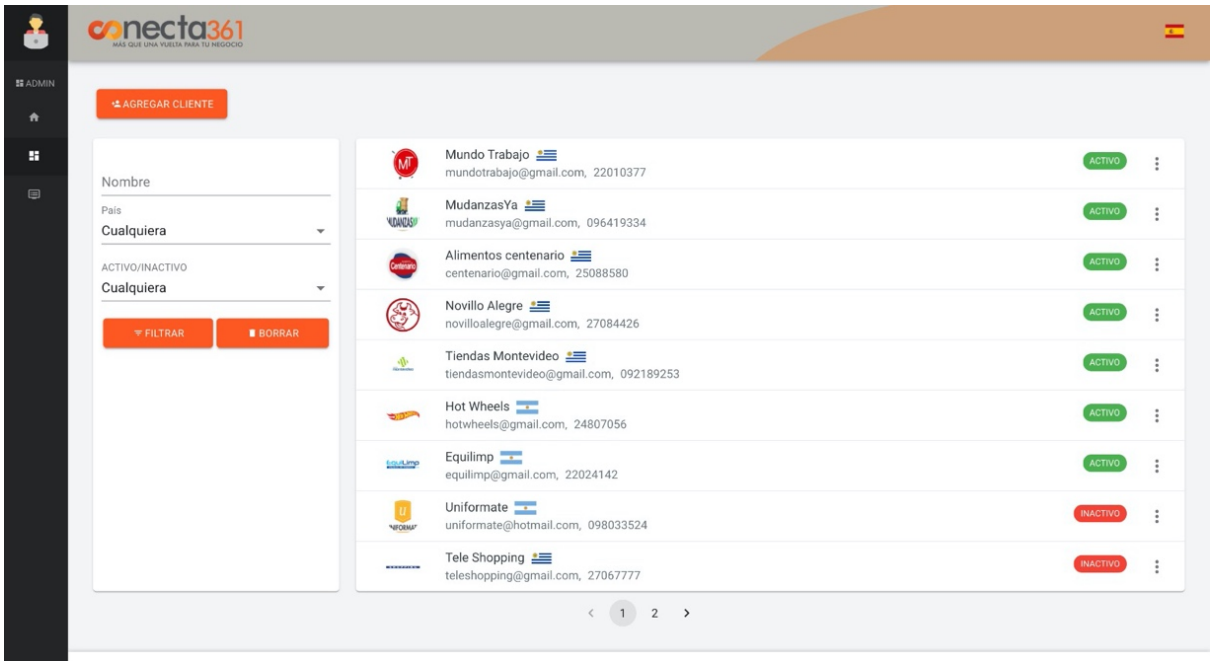


Ilustración 12 - Vista de clientes de *"Tracker361"*

La solución propuesta consta de un sistema que permita a *Conecta361*, como agencia de marketing digital, poder brindarles a sus clientes los mejores y más reales datos sobre sus campañas digitales.

Esto se logra bajo un principio fundamental: todas las ventas deben ser rastreadas hasta su origen publicitario, si es que este existe.

Por esta razón, el primer paso de la solución es encontrar las diferentes formas posibles de rastrear las ventas realizadas en locales físicos, como es el caso de la automotora, pero causadas a través de algún canal online, como en *Facebook*. Una vez encontradas, el sistema debe procesarlos para obtener si los mismos pueden considerarse como una conversión.

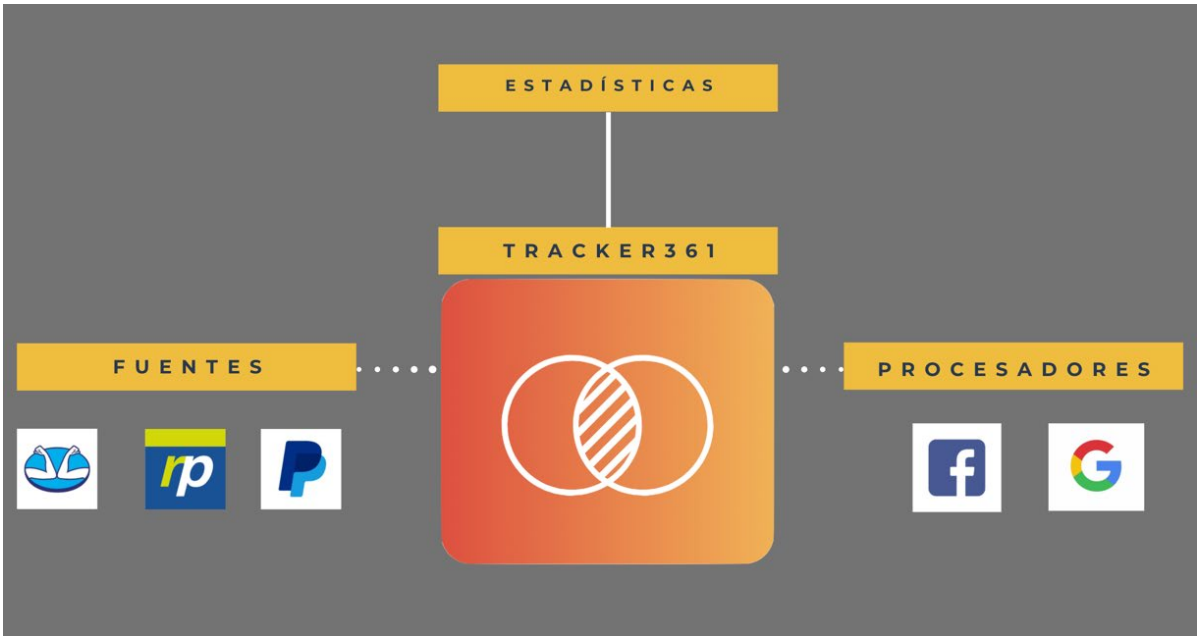


Ilustración 13 - Diagrama de alto nivel de *"Tracker361"*

Por consiguiente, el sistema es capaz de conectarse con cualquier tipo de fuente de datos externa, vía API o en caso de no tenerla, por *Web Scrapping*. Las fuentes de datos son aquellas que de una u otra manera dan información al respecto de las ventas de un cliente, sea el nombre del comprador, teléfono o descripción del producto. De esta forma, el sistema centraliza en un único punto de entrada la información de los eventos de ventas de cualquier cliente, a cualquier hora y desde cualquier lugar.

Cada cliente puede optar por configurar sus propias fuentes de datos. La automotora, por ejemplo, acepta como métodos de pago a *MercadoPago*. De esta manera, si este

habilita tal opción como fuente de dato, cada pago realizado en su local con este medio será posteriormente obtenido por el sistema *"Tracker361"*. A su vez, si además guarda la información de sus compradores en un *CRM cloud*, como *Zoho*, este mismo también podrá ser configurado para obtener y/o importar los datos de estos.

A continuación, se visualizan las fuentes de datos configuradas para un cliente. En el mismo se puede observar como *MercadoPago* y *RedPagos* están activos para procesar los eventos de sus ventas. Tanto *Zoho* como *Paypal* están desactivados, pero si el cliente desea activarlos a posteriori, el sistema automáticamente comenzara a procesar los eventos provenientes por dichas fuentes.

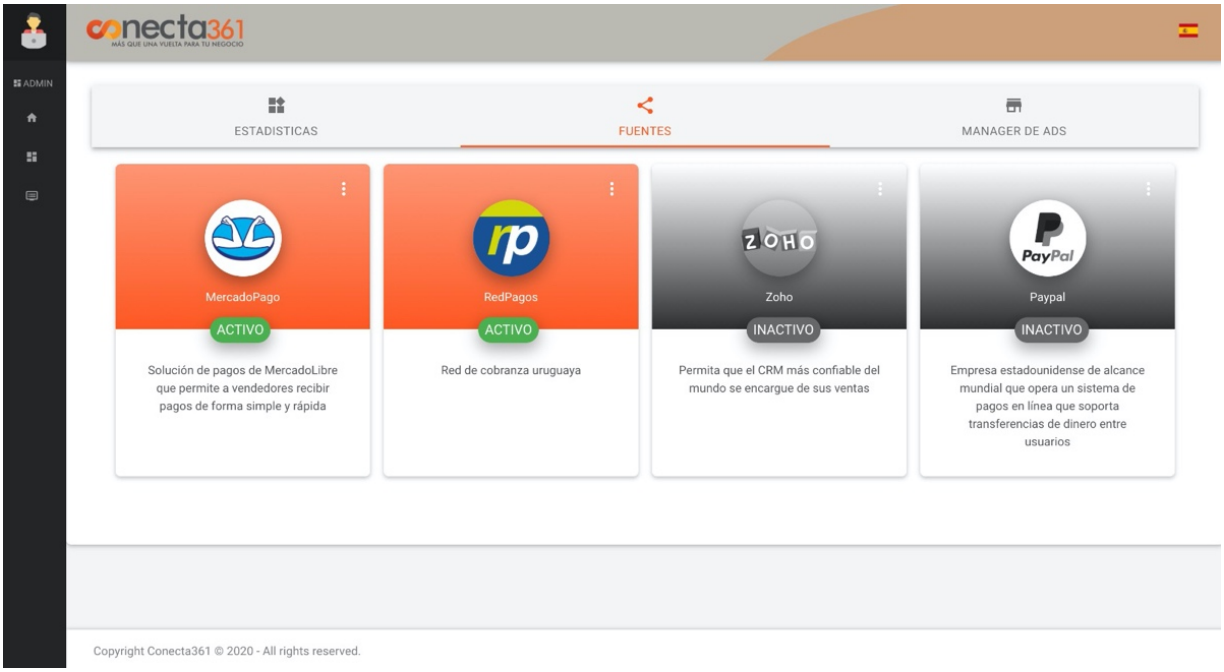


Ilustración 14 - Vista de fuentes de datos de *"Tracker361"*

Los eventos provenientes de las fuentes de datos deben ser luego procesados por el sistema y enviados a *Facebook* y *Google* para evaluar si una venta convirtió o no de forma *offline*. Para ello cada cliente deberá tener configurada su cuenta publicitaria. La configuración de los procesadores se puede ver a continuación a modo de ejemplo:

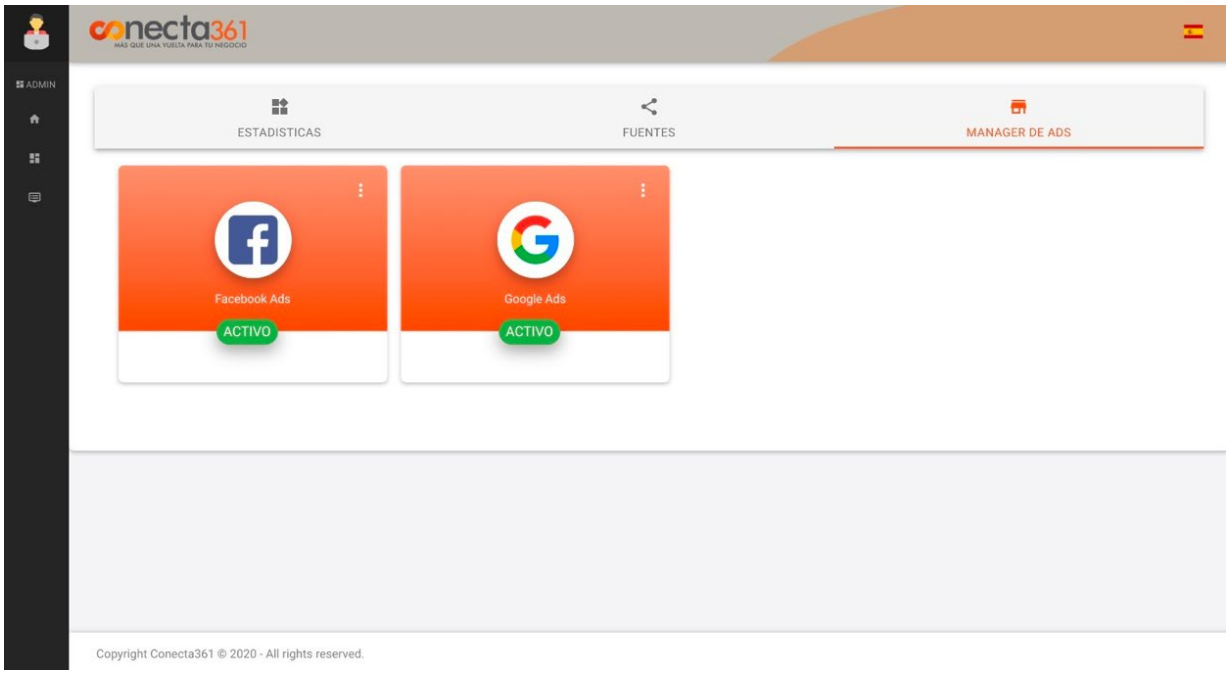


Ilustración 15 - Vista de procesadores en *"Tracker361"*

Una vez configuradas las fuentes de datos y los procesadores, el sistema está preparado para comenzar con el rastreo de ventas y atribución de conversiones. Todos los eventos serán mostrados en el panel con su detalle. Este incluye la fuente de origen, monto y moneda de la venta y perfil del comprador. A su vez, cada evento está asociado a un estado interno del sistema, que puede ser pendiente o procesado. El primero significa que la venta aún no ha sido analizada por los procesadores para su posible conversión, mientras que el segundo sí.

Se muestra la pantalla de eventos de un cliente de ejemplo:

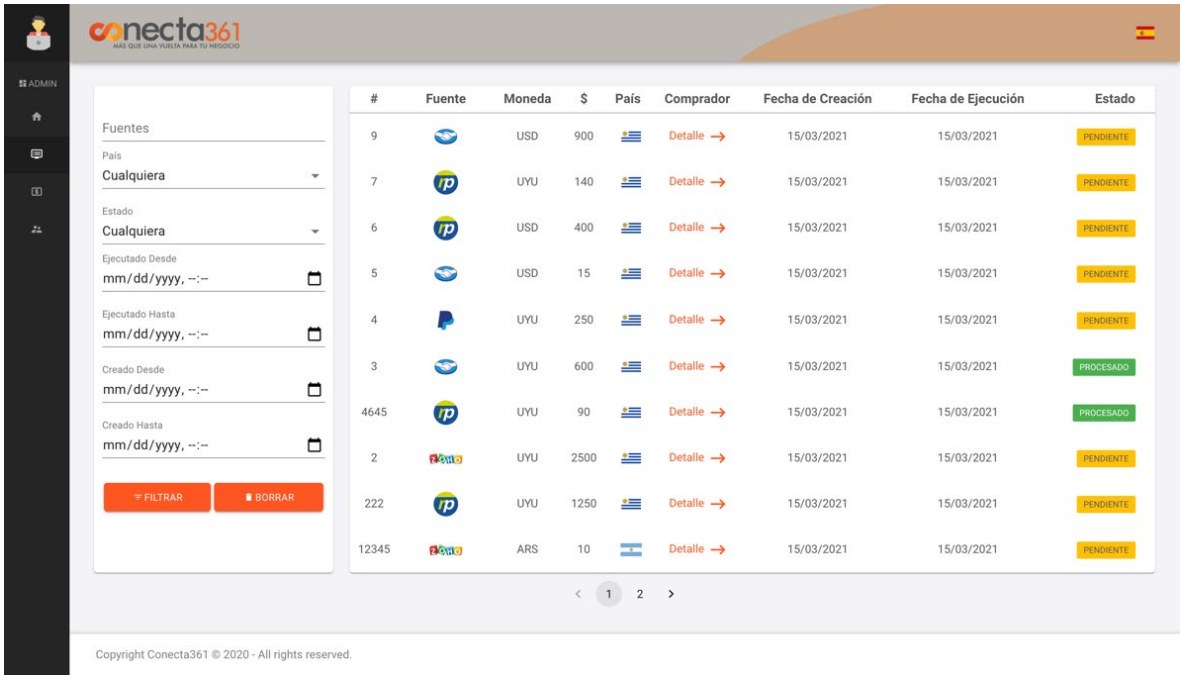


Ilustración 16 - Vista de eventos en *"Tracker361"*

Relacionado al manejo de eventos, se presentó un desafío que la solución resuelve de manera satisfactoria.

Construcción de perfil del comprador

Mediante el historial de compras de una persona, el sistema construye un perfil de comprador que permite mejorar los datos a enviar a los procesadores para una siguiente instancia. A modo de ejemplo, se presenta el siguiente escenario:

Una persona realiza una compra de unas zapatillas el 10 de marzo. El sistema identifica el nombre y cedula de identidad de la persona, armando un perfil con sus datos.

Al cabo de un mes, la persona realiza la compra de un teléfono. Mediante el método de pago utilizado, el sistema es capaz de identificar el email y la cédula de identidad del comprador.

Antes de enviar los datos del producto y del comprador, el sistema analiza si puede identificar al mismo dentro de su base de datos con los datos de la compra actual. Al identificar que el documento ya estaba registrado en la base de datos, el sistema actualiza el perfil del comprador, optimizando sus datos para luego enviar información más completa al procesador.

Tal como se puede ver en la siguiente ilustración, los datos finalmente son recolectados y enviados en base al historial de compra.

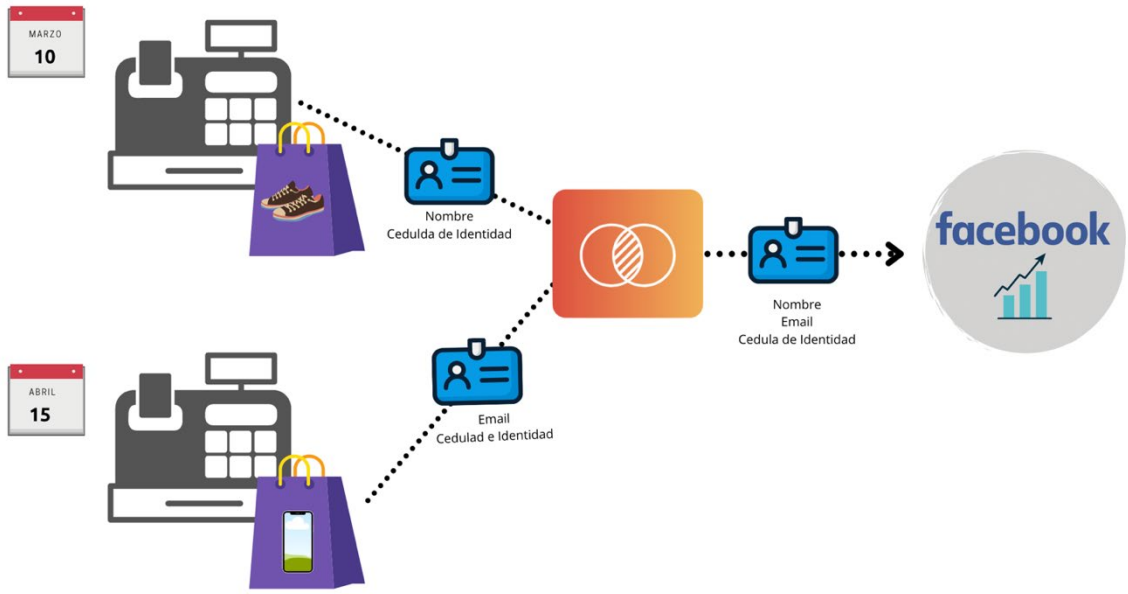


Ilustración 17 - Construcción de perfil de comprador

Si bien este proceso puede parecer sencillo, aporta un gran valor al procesamiento de atribución de conversiones. Los algoritmos de *Facebook* y *Google Ads* se basan en la mayor cantidad de datos posibles de la venta para atribuir conversiones, por lo que reconstruir los perfiles de los usuarios puede mejorar la optimización en la identificación de las conversiones.

Como consecuencia del procesamiento de las conversiones, se le presenta a *Conecta361* un panel consolidado en el cual puede visualizar todas las estadísticas de las campañas publicitarias de sus clientes. En él, se puede observar quiénes son sus clientes que obtuvieron mayores ganancias en el mes, los que más convirtieron y un resumen comparativo de los eventos convertidos y enviados. A su vez, se muestra que porcentaje de eventos fueron atribuidos por cada canal. Cabe destacar que hay muchas estadísticas que no se muestran en el panel pero que pueden ser explotadas a demanda del cliente. En la siguiente ilustración se observa el mismo:

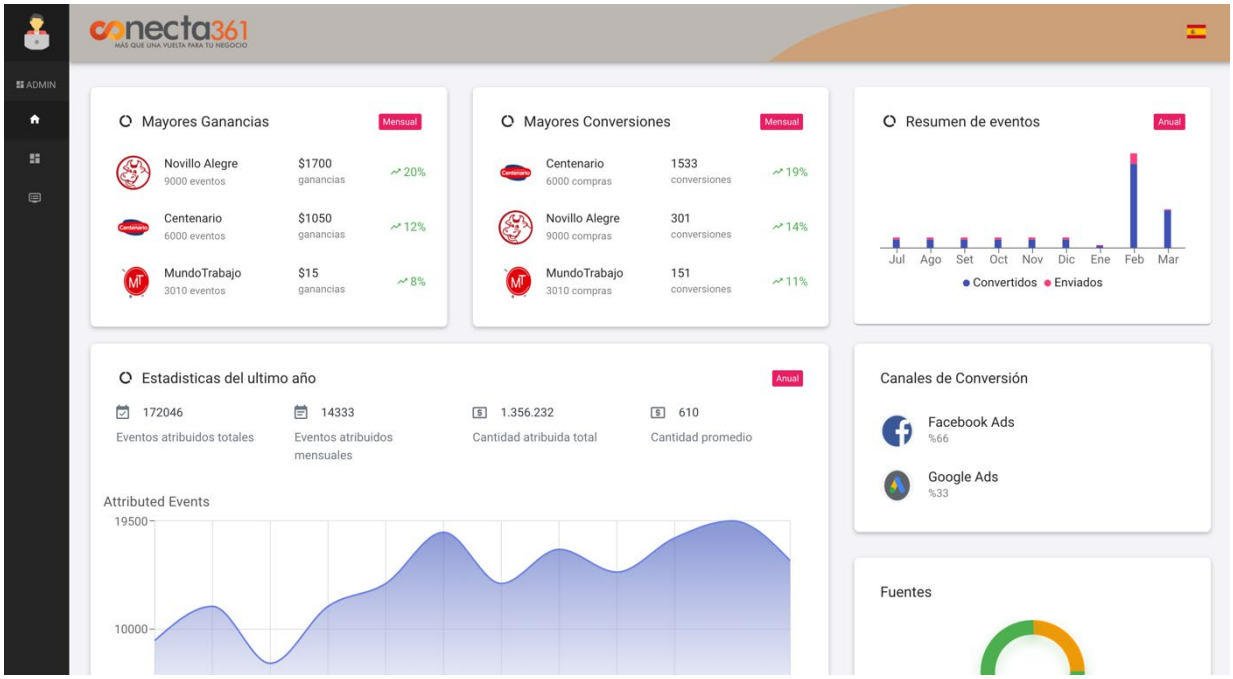


Ilustración 18 - Vista de estadísticas de *"Tracker361"*

Como resultado de interpretar bien las estadísticas, Conecta361 optimiza las campañas de sus clientes de forma de mejorar su rentabilidad.

Por otro lado, la solución ofrece una integración directa a la plataforma sin necesidad de utilizar la plataforma web. El sistema expone una API que permite realizar peticiones para la consulta de estadísticas de conversiones e importación de eventos y compradores.

De esta manera, si un cliente desea desarrollar su propia solución que se acomode más a sus necesidades, sin dejar de utilizar las funcionalidades provistas por el sistema, lo podrá hacer con facilidad.

Además, la solución destaca la implementación de una primera versión de SDK, construida bajo el lenguaje Java. Esta permite realizar las mismas funcionalidades que las expuestas por la API, pero ofrece otras ventajas en la facilidad de desarrollo. Además, la SDK fue construida de manera que, si el versionado de la API cambia, esta siempre se integre con la última versión de forma agnóstica al desarrollador.

A continuación, se puede visualizar la especificación de la *SDK* realizada para los clientes de *Conecta361:*

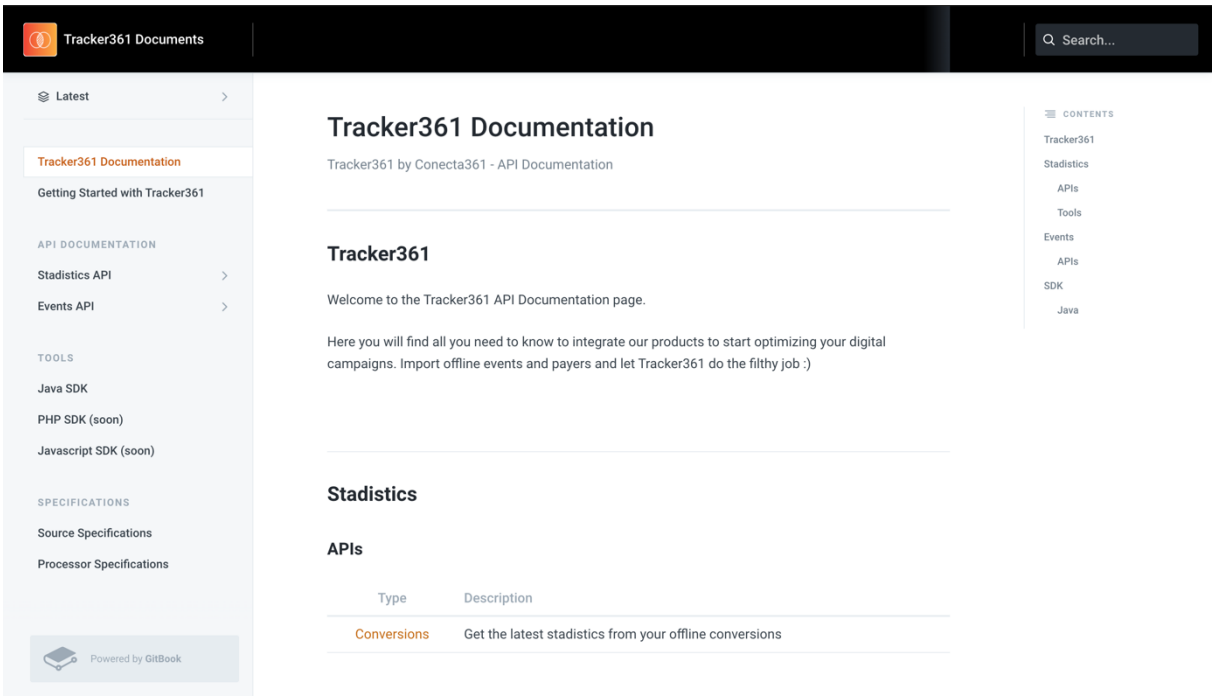


Ilustración 19 - Especificación de la SDK

3.3.1 Actores

Se identificaron a los siguientes actores principales en el contexto de la solución. Fueron clasificados en dos categorías: directos e indirectos. Los actores directos son aquellos que tienen alguna interacción directa dentro de plataforma. Los indirectos, si bien son parte fundamental para el funcionamiento de la solución, no tienen ningún tipo de acceso al sistema. A continuación, se pude ver un diagrama para entender sus relaciones.

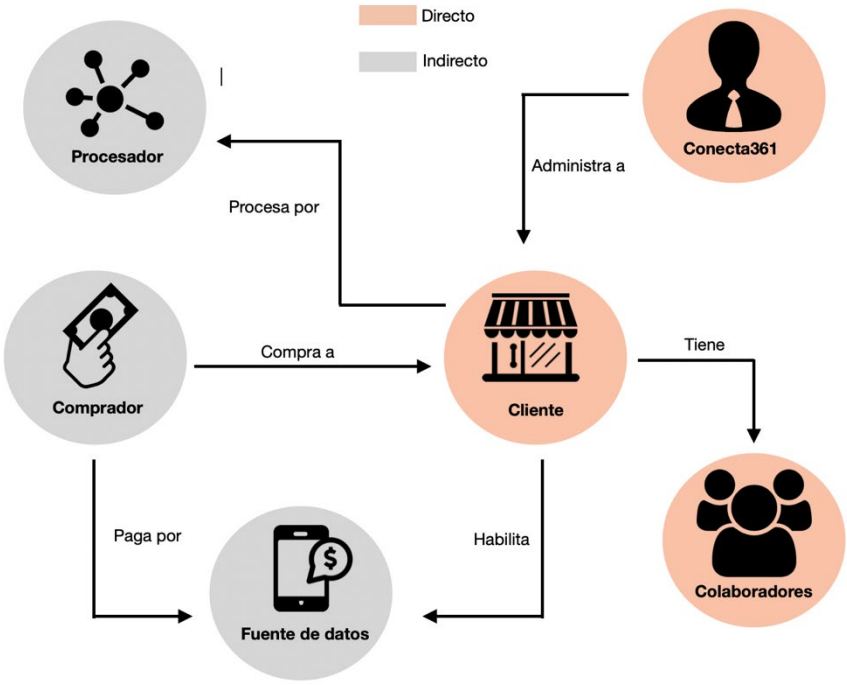


Ilustración 20 - Diagrama de actores

Conecta361 – Super Administrador (Super Admin)

Son los administradores del sistema. Su rol es de super *admin*. Estos se encargan de gestionar a todos sus clientes y las configuraciones asociadas a estos.

Cliente – Administrador (Admin)

El cliente representa a una empresa perteneciente de la agencia *Conecta361*. Es la responsable de habilitar y configurar las fuentes de datos que desea procesar. También

debe configurar a los administradores de anuncios disponibles como *Facebook* o *Google Ads*. Su rol es de *admin.*

Colaboradores – Usuario (User)

Los colaboradores son usuarios de un cliente. Estos tienen menos privilegios que los administradores. Sin embargo, tienen acceso de lectura para todas las mismas funciones que el cliente. Estos deben ser invitados a través del administrador un cliente.

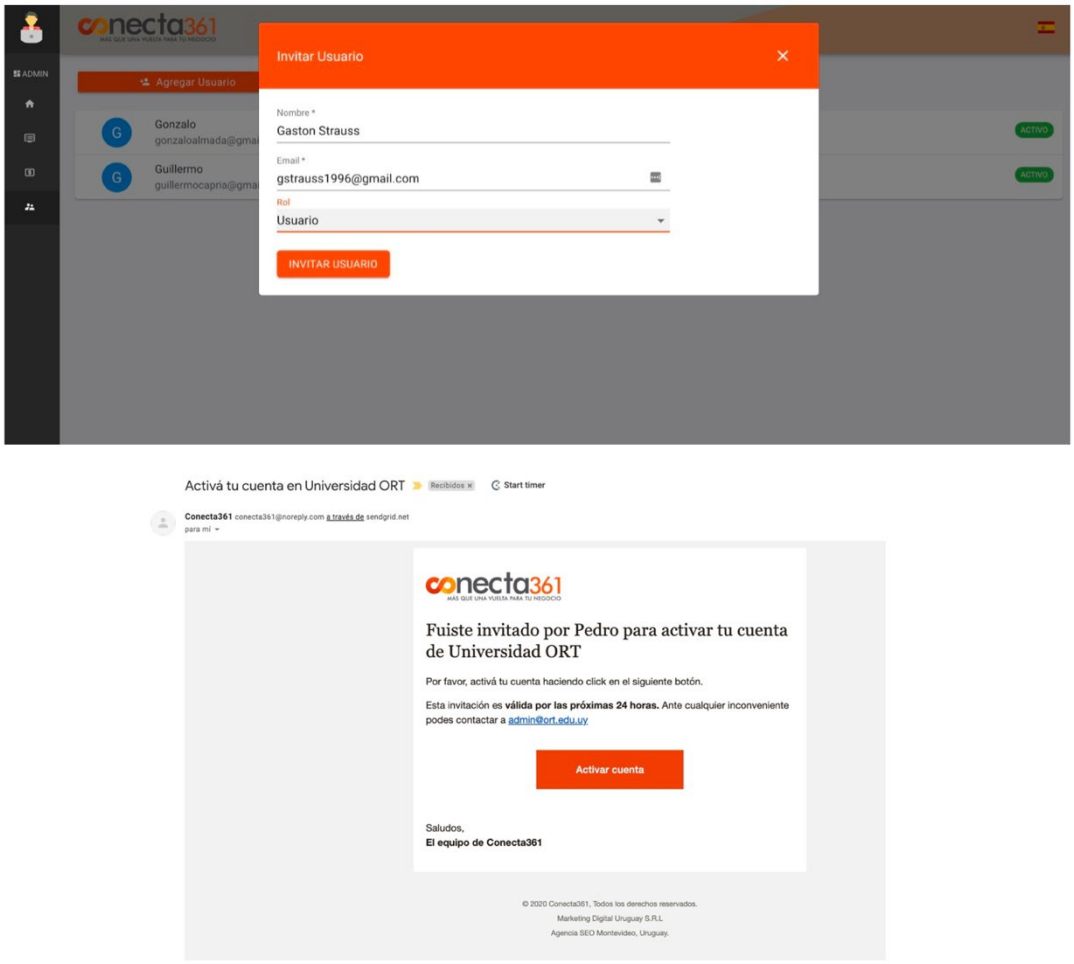


Ilustración 21 - Invitación de cliente a usuario

Comprador

Es un actor indirecto del sistema, y no tiene conocimiento de su participación en los procesos de este. Es la persona que realiza una compra en el local de un cliente. Estos son recolectados mediante la obtención de los eventos.

Fuente de datos (plataforma de pagos / CRMs)

Las fuentes de datos son todos los servicios externos que permiten la extracción de datos de las ventas de un cliente. En este se incluyen a plataformas de pago y *CRMs.*

Procesador

Son los administradores de anuncios. Se encargan de procesar a los eventos para luego evaluar si alguno de ellos convirtió.

Todos los actores, tanto directos como indirectos, fueron contemplados para la realización de los requerimientos funcionales y no funcionales del sistema. Estos se pueden ver en la sección Listado de requerimientos.

3.4 Resumen

El problema de la automotora es común a los clientes de *Conecta361*. Por ello, se vio la necesidad de asociar las campañas publicitarias de sus clientes a las ventas ocurridas *offline,* de forma de proporcionar mejores estadísticas. Las soluciones manuales que ofrecen los administradores de anuncios no son escalables y sus API's son muy complejas tanto conceptual como técnicamente.

Es por esta razón que se implementó una solución que sea capaz de resolver la detección de conversiones *offline*, mediante la automatización de los procesos de obtención de datos y su procesamiento. Los detalles de los requerimientos son encontrados en el capítulo de Ingeniería de requerimientos.

4. Marco metodológico

En este capítulo se detallarán las principales características del proyecto. Estas impactaron en la decisión de las metodologías a utilizar y en las herramientas y tecnologías que el equipo utilizó con el objetivo de tener una estrategia de trabajo definida. Cabe destacar que todos las metodologías y fases mencionadas en este capítulo se profundizarán a lo largo del documento.

4.1 Características del proyecto

A continuación, se explican las principales características del proyecto:

Cliente experto en marketing digital, pero no especializado en desarrollo de software: Marcelo Wilkorwsky, CEO de *Conecta361*, es experto y consultor de marketing digital por lo que tiene mucho conocimiento sobre el dominio del problema [46]. Dentro de su agencia de marketing no existe un departamento ni desarrolladores de software. Esto implicó que el equipo tenga que desarrollar el *software* y su infraestructura desde cero, con la libertad de elegir las herramientas y tecnologías más adecuadas.

El equipo no conocía sobre el área del marketing digital: El marketing digital es cada vez más utilizado como forma de realizar publicidad y medir las campañas. Sin embargo, esta área era desconocida por parte de los integrantes del equipo debido a que no tuvieron un acercamiento académico ni tampoco experiencia profesional en esta. El gran grado de incertidumbre presente en el dominio del problema, requirió que la etapa de investigación se basara en aprender los conceptos básicos del área y de la implementación de pruebas de concepto como forma de mitigar el riesgo.

Producto de gran complejidad: Sumado a la complejidad natural del dominio y del problema, el equipó optó por elegir un proyecto a nivel tecnológico que sea desafiante y requiera de satisfacer necesidades reales. Este requirió de automatizar y mejorar los procesos manuales realizados por el cliente. A su vez, la recopilación de grandes

volúmenes de datos de diferentes fuentes, la interoperabilidad requerida para obtener estadísticas publicitarias y la modificabilidad necesaria ante la posible implementación de nuevas herramientas en el futuro fueron algunos de los desafíos que el equipo tuvo que afrontar.

Producto exportable a otros países: El producto resuelve problemas reales y no es excluyente a nivel local; es altamente exportable a cualquier ciudad y país del mundo. El software se basa en el modelo *Software As A Service* (SaaS por sus siglas en inglés) y está preparado para adaptarse a cualquier idioma, siendo hoy en día español e inglés los principales.

Todas estas características hacen que haya sido necesario manejar una estrategia que permita al equipo adaptarse al cambio rápidamente [47]. Por estos motivos fue que la aplicación de metodologías agiles fueron ideales, ya que los valores de su manifiesto eran compatibles con lo que el equipo quería lograr [48]. A su vez, estas impactaron de forma directa con la implementación de las distintas fases del proyecto (véase sección 4.5).

En base a estas características es que se tomaron diferentes decisiones que serán explicadas en las siguientes secciones.

4.2 Roles

De manera de mantener una gestión de trabajo organizada, cada miembro del equipo ocupó un rol. Esta asignación de roles fue basada en la experiencia e interés de los integrantes. El definir roles fue clave para que aumente la productividad del equipo y por ende del proyecto. Se detallan a continuación:

- Gonzalo Strauss: responsable de la arquitectura y desarrollador *backend.*
- Gonzalo Kurin: responsable de la ingeniería de requerimientos y desarrollador *frontend.*
- Itai Miller: responsable de la calidad y desarrollador *backend.*

- Mauricio Pisabarro: responsable de la gestión del proyecto, arquitectura y desarrollado del *frontend.*

Cabe destacar que los cuatro integrantes del equipo fueron responsables del desarrollo del sistema.

4.3 Relación con el cliente

El equipo buscó mantener una estrecha relación con el cliente de forma de poder alinear las expectativas del producto. Poder aprovechar su experiencia y conocimiento en el área del marketing fue clave para lograr fortalecer los conceptos fundamentales de la temática por parte del equipo, cuyo conocimiento inicial era casi nulo.

Durante la fase inicial el cliente estuvo dispuesto a pactar reuniones a demanda con el objetivo de validar el problema y relevar la especificación de los requerimientos para la solución. Debido a la pandemia causada por el COVID-19 el equipo mantuvo una comunicación remota, realizando video llamadas y utilizando documentos compartidos para establecer las funcionalidades a implementar. Al comenzar el desarrollo del producto, el equipo se contactó con el cliente periódicamente mediante los eventos de la metodología de trabajo ágil que es detallada más adelante. De esta forma, fue posible validar los requerimientos de forma frecuente.

Tanto el cliente como los integrantes del equipo propusieron ideas y soluciones a lo largo del proyecto. Fue el propio cliente el que explícitamente solicitó al equipo la proactividad por parte del mismo para sugerir ideas que aporten valor a la solución del proyecto.

Cabe destacar que la relación con el cliente fue excelente durante todo el proyecto y a lo largo del mismo se mostró muy predispuesto a responder y aclarar las dudas que fueron surgiendo.

4.4 Metodología y ciclo de vida

Debido a las características del proyecto ya mencionadas en el punto 4.1 el equipo decidió utilizar el ciclo de vida incremental e iterativo [33]. La incertidumbre en la definición de requerimientos, uso de las tecnologías y necesidades cambiantes fueron mitigadas al adoptar las metodologías agiles de *Kanban* y *Scrum*, compatibles con en el ciclo de vida mencionado. Otra de las razones por la cual se optó por este ciclo de vida es por el alto compromiso de los integrantes del equipo para con el proyecto.

En el siguiente diagrama, se puede observar cómo se implementó dicho ciclo de vida. Se detallan las diferentes fases que atravesó el equipo durante el proyecto.

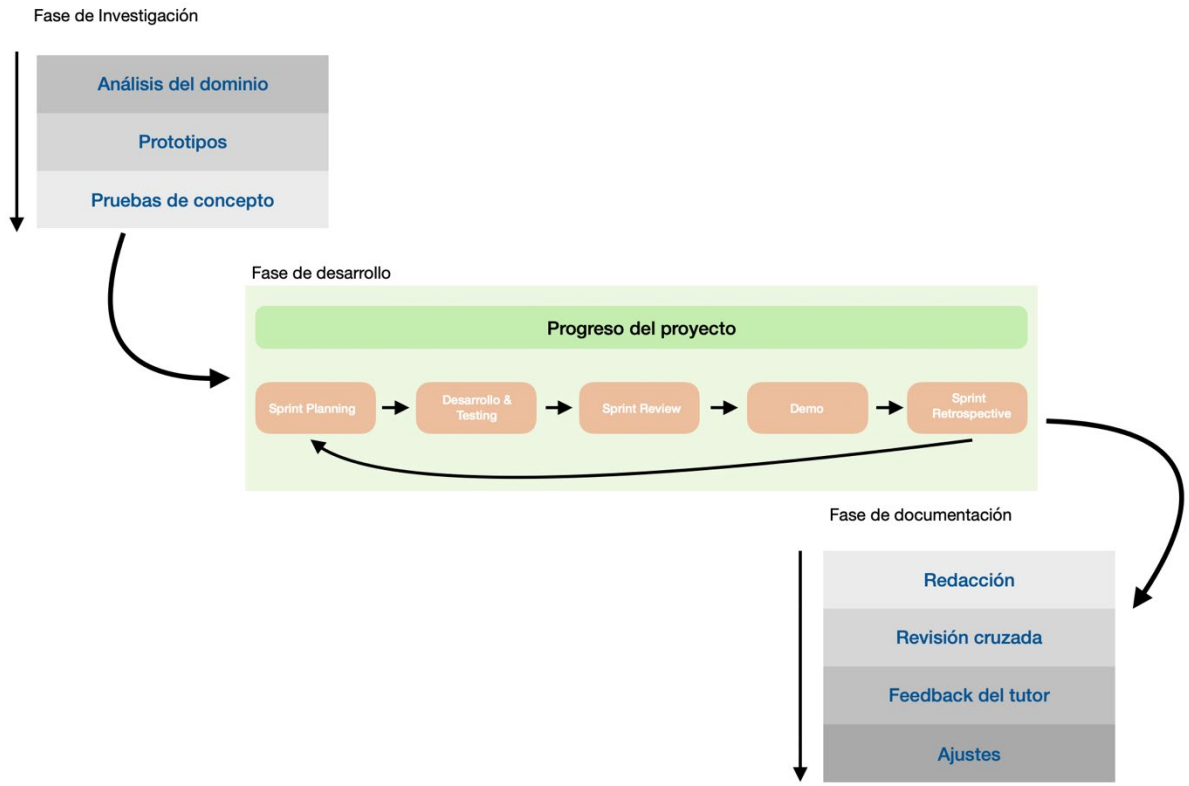


Ilustración 22 - Ciclo de vida

Al utilizar un progreso incremental se lanzaron versiones de *software* en ciclos cortos, facilitando a las fases de investigación y desarrollo. Algunos de los beneficios de haber trabajado bajo este ciclo de vida son:

1. Iteraciones cortas que permiten al equipo adaptarse y reaccionar a las necesidades cambiantes del cliente.
2. Permite mostrar de forma continua los avances al cliente y recibir *feedback*. Al construir un producto complejo, es imperativo validar de forma frecuente con el cliente si el desarrollo realizado hasta el momento cumple con sus expectativas y necesidades.
3. La aceleración de la entrega de *software* ofrece valor al cliente de forma constante.

4.5 Fases del proyecto

A continuación, se detallan las diferentes fases del proyecto teniendo en cuenta las características mencionadas en el punto 4.1.

En estas fases el equipo buscó optimizar sus procesos mediante el uso de la automatización para maximizar su eficiencia. Se utilizó la combinación de metodologías ágiles mediante *Kanban* y *Scrum* junto con algunas de las buenas prácticas del ciclo de vida de *DevOps* como lo son la entrega y despliegue continuo, detalladas en la sección Integración y despliegue continuo .

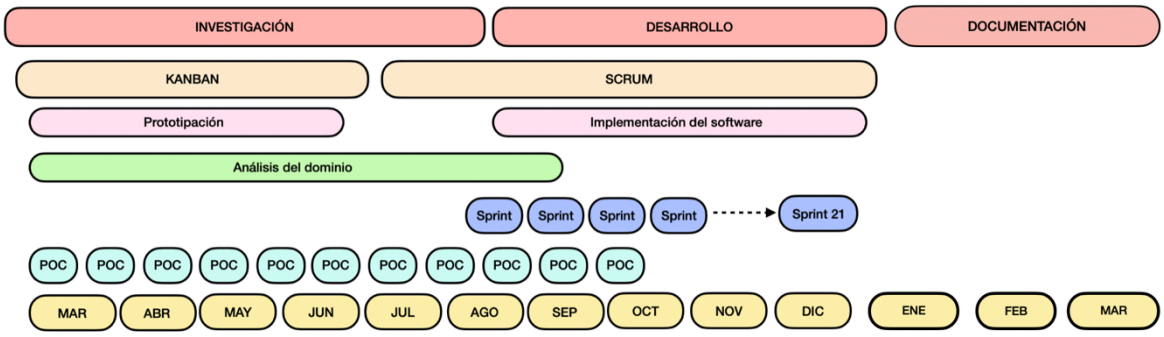


Ilustración 23 - Fases del proyecto

4.5.1 Fase de investigación

La primera fase del proyecto fue la de investigación. Esta tuvo una duración de cinco meses comenzando la misma en la primera semana de marzo y culminando la fase el 31 de Julio.

Durante este periodo, el equipo investigó acerca del marketing digital para introducirse en ese mundo desconocido. Se llevaron a cabo reuniones cada dos semanas con el cliente para validar lo investigado y consultarle las dudas que fueron surgiendo durante la misma. El propósito de haber realizado reuniones frecuentes fue para mantenerse alineados en todo momento.

Durante el aprendizaje de los conceptos básicos del marketing digital, se le presentó al cliente *wireframes*, *mockups* y prototipos (véase Anexo 13.4) con el objetivo de proponer ideas y poder definir de manera conjunta los requerimientos del sistema. Se realizaron pruebas de concepto de las diferentes plataformas *online* y API's de marketing tales como *Facebook* y *Google Ads*. El objetivo de tales pruebas fue mitigar los riesgos y entender el funcionamiento de estas antes de entrar en la fase de desarrollo.

El equipo considero útil dedicarle tiempo a investigar sobre el manejo de la infraestructura del sistema e integración continua de modo de hacer integraciones automáticas y detectar fallas en etapas tempranas.

Para esta primera fase, el equipo decidió utilizar la metodología ágil *Kanban* para obtener un flujo continuo de investigación y poder adaptarse al cambio de una manera fácil y rápida para la etapa de desarrollo [49]. En la sección Fase de investigación se profundizará sobre el mismo.

4.5.2 Fase de desarrollo

La segunda fase del proyecto consistió en desarrollar la solución del sistema. Fue muy importante tener una buena comunicación frecuente entre los equipos de *backend y frontend* de forma de ir alineados con el *sprint* y realizar pruebas de integración para probar el correcto funcionamiento de las diferentes partes.

Scrum

Para la gestión del proyecto, se optó por un enfoque ágil como lo es la metodología *Scrum* [50] debido a las características mencionadas en el punto 4.1. Algunas ventajas que presenta son el poder definir roles dentro del equipo, establecer iteraciones de tiempos fijos (*sprints*), tener flexibilidad en agregar valor a los usuarios pudiendo interactuar frecuentemente de manera de priorizar y validar de forma constante. También poder recolectar métricas como lo son la velocidad del equipo y esfuerzo de las tareas. Cabe destacar que todos los miembros del equipo ya contaban con experiencia trabajando con esta metodología por lo que la decisión fue unánime. Se utilizó Scrum adaptado ya que el evento de la *daily* meeting no fue realizada de manera estricta debido a las diferentes actividades y disponibilidad de cada miembro del equipo.

Prácticas de desarrollo

Las prácticas de desarrollo estuvieron muy ligadas a la metodología utilizada y a la naturaleza cambiante del proyecto. Es por esta razón que estas fueron aplicadas en base a los valores del manifiesto ágil [51].

En primer lugar, el desarrollo del sistema surgió de entornos inestables, con el cambio como condición inherente y una evolución rápida y continua, por lo que se le dio prioridad a la capacidad de respuesta que a planes de seguimiento preestablecidos. Por esta razón se realizaron *sprints* muy pequeños de una semana, de forma validar los requerimientos con el cliente rápidamente. También se realizaron pruebas de concepto en la etapa de desarrollo y técnicas de relevamiento que ayudaron a agregar conocimiento

eficientemente. Se realizó una arquitectura evolutiva, de forma de agregar o modificar sus componentes a medida que las necesidades iban modificándose y anticipar el cambio de la manera más productiva posible.

Se valoró que el *software* funcione más que una documentación extensiva, por lo que se aplicaron estándares de *clean code* y pruebas automatizadas, además de aplicar prácticas de apoyo a DevOps, como la integración y entrega continua. Se siguieron estándares de revisión de código y *pull-requests* de forma de asegurar que el código en producción siempre esté funcionando, utilizando además diferentes ambientes y entornos para pruebas.

Es importante destacar que en todo momento el equipo se vio ante la necesidad de proponer iniciativas de cambio, realizando técnicas de relevamiento como el *brainstorming.*

Por último, pero no menos importante, cabe destacar que la prioridad siempre estuvo en satisfacer al cliente a través de entregas de software de valor. Por este motivo el equipo llevó a cabo la automatización de todo el proceso de ingeniería de software, desde el incluyendo las pruebas y despliegue a producción, asegurando en todo momento la calidad del software entregado.

4.5.3 Fase de documentación

La fase de documentación fue incluida dentro de la aplicación del ciclo de vida iterativo e incremental. Esta tercera y última fase del proyecto consistió en la iteración de incrementos del documento. La metodología de trabajo propuesta se basó en la redacción de secciones del documento de forma individual, para luego ser revisada por otro integrante y finalmente por el tutor. Al finalizar cada iteración se añadió una fase de ajuste del documento con sugerencias del tutor y luego ejecutadas por los integrantes del equipo. De esta forma se aseguró que cada iteración agregue más valor que la anterior y sus ajustes no dañen lo realizado hasta el momento.

4.6 Herramientas utilizadas

Al inicio del proyecto el equipo investigó las principales herramientas a utilizar a lo largo de todo el proyecto, divididas en diferentes categorías como comunicación, gestión, documentación, entre otras. El objetivo de aplicar estas herramientas a lo largo de todo el proceso del proyecto fue poder maximizar la eficiencia y productividad del equipo. En los capítulos que hagan referencia a alguna de las herramientas utilizadas, se explicará con mayor detalle su utilidad y uso práctico. El resumen de estas se puede encontrar en el Anexo 13.5 y Anexo 13.6.

4.7 Resumen

Las decisiones sobre la elección de las metodologías, procesos, herramientas y tecnologías fueron basadas en las características particulares del proyecto, como la incertidumbre del dominio, requerimientos cambiantes y complejidad técnica. Habiéndose realizado una fase de investigación mediante *Kanban* y otra de desarrollo con *Scrum*, el equipo trabajo bajo los lineamientos y principios del manifiesto ágil, que permitieron en última instancia entregar software de valor al cliente de forma frecuente.

Como lección aprendida, el equipo valoró poder aplicar metodologías acordes a las necesidades del cliente, aprendiendo a anticiparse ante los cambios y automatizando los procesos necesarios.

5. Ingeniería de requerimientos

En el presente capítulo se explicarán las diferentes actividades realizadas en cada fase del proceso de la ingeniería de requerimientos. También se listarán los requerimientos funcionales y no funcionales del sistema. Debido a que el cliente no estableció una serie de requerimientos formales, si no que le dio libertad al equipo a la hora de definir los mismos, este se realizó de una manera ágil y mediante una constante comunicación con el mismo. Para realizar el proceso, el equipo tuvo que entender sus necesidades para luego interiorizarse con el dominio del problema, y de forma conjunta especificar los requerimientos a implementar.

5.1 Proceso

El proceso se puede dividir en dos etapas. Una primera etapa de investigación, la cual se basó en el estudio del dominio de manera de ser proactivos ante la propuesta de ideas hacia el cliente. Por un lado, el equipo se informó del área a través de diversos artículos, libros, tutoriales y páginas web que fomentaron los debates internos y aprendizajes colectivos. Además, se realizaron pruebas de concepto y se aplicó la técnica de ingeniería inversa sobre otros sistemas de funcionalidades similares con el objetivo de tener una visión más amplia del negocio, así como el surgimiento de nuevas ideas que agreguen valor a la plataforma [52]. En el Anexo 13.7 se puede encontrar un análisis más profundo de la aplicación de esta técnica. A su vez, hay que destacar que los principales conceptos obtenidos durante esta fase se pueden encontrar en el capítulo Introducción al Marketing Digital y los resultados de aplicar estos se encuentran en Contexto, problema y solución.

Finalizada la etapa de investigación, el equipo comenzó con la aplicación de diferentes técnicas para relevar requerimientos. La propuesta de ideas partió en base a las necesidades del cliente mencionadas en la sección 3.2.1. Debido al desconocimiento del dominio y a que se trata de un problema abierto y dinámico, se formalizaron las *user stories* a medida que se iban abordando estas [53]. El cliente siempre estuvo afín a

mantener una constante comunicación con el equipo, de forma de validar los cambios en estas *user stories.* El resultado de aplicar lluvia de ideas, prototipos, *mockups* y escenarios de uso permitieron definir los requerimientos funcionales y no funcionales del sistema. Dichas técnicas fueron aplicadas durante toda la fase de desarrollo.

5.1.1 Investigación

Para entender las necesidades del cliente el equipo tuvo que, en primera instancia, investigar sobre el dominio. Más allá de las dificultades propias de este (conceptuales y técnicas), las necesidades del cliente se centraban en un problema ya de por si moderno, que demandaban al equipo interiorizarse en la temática. En el siguiente Anexo 13.8 se pueden encontrar evidencias de la comprensión del problema y del dominio mediante ideas en lápiz y papel.

En esta primera instancia se aprendieron conceptos muy relevantes del marketing digital, como lo son las conversiones *offline*, campañas publicitarias, retorno de inversión, administradores de anuncio y muchos otros más. Permitieron no solo entender sobre el dominio, sino que también generar debates internos y constructivos que resultaron en propuestas de ideas de parte del equipo, fomentando la proactividad. Cabe destacar que el aprendizaje de la temática fue constante durante todo el proyecto, ya sea leyendo, debatiendo, haciendo o enseñando, tal como lo indica la pirámide de *Gassler* [54].

El cliente tenía identificado el problema a solucionar, además de contar con algunas vagas ideas como solución*. "Ustedes sabrán mejor, son ingenieros y técnicos",* decía al terminar cada una de las reuniones. Su forma de actuar daba libertad al equipo a sugerir posibles soluciones y funcionalidades para agregar valor al sistema.

En primer lugar, se definió una lista inicial de temas a estudiar y preguntas abiertas a responder, que fueron planificados junto con el cliente, y otros a voluntad de los integrantes. Algunas de las principales fueron:

- ¿Qué es el marketing digital? ¿En qué ventaja al tradicional?

- Contexto actual: ¿Qué porcentaje de ventas ocurren offline? ¿Por qué?
- ¿Cómo funcionan y para qué sirven los administradores de anuncios?
- ¿Qué son las conversiones offline y que mecanismos existen en la actualidad? ¿Cómo funcionan?

A medida que se iban profundizando los conceptos y se respondían las preguntas iniciales, otras nuevas iban surgiendo. El equipo debatía sobre estas y volvía nuevamente a repetir el ciclo de investigación. De modo de llevar una investigación organizada, cada integrante elegía un tema o pregunta a estudiar según su nivel de interés. En caso de coincidencias, se llegaba a un acuerdo sobre quien la tomaba. El listado de las principales temáticas investigadas se puede observar en el Anexo 13.9.

Se utilizó la metodología *Kanban* para esta etapa, utilizando el concepto de *spike* en esta para que el equipo estudie y se interiorice en el dominio [55]. Ayudó a estimar las mismas y tener una mejor organización de las tareas a seguir. Cada uno de estos contaba con su propio criterio de aceptación y el resultado a esperar, junto con un conjunto de tareas a realizar.

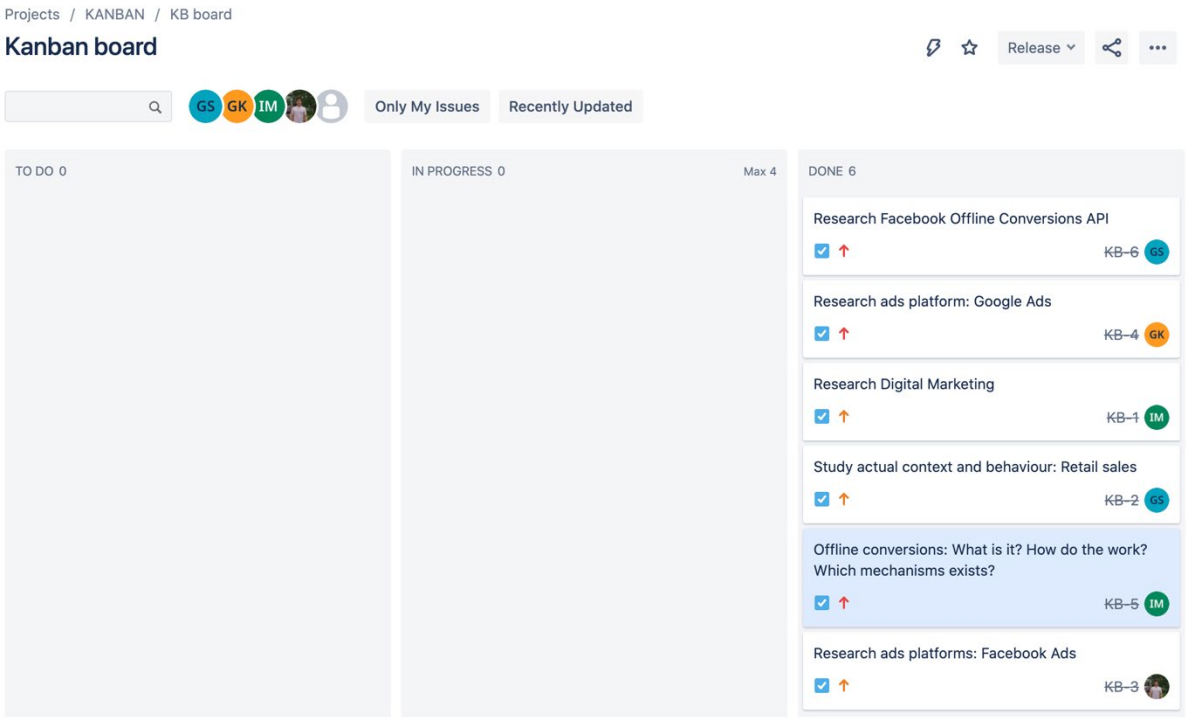


Ilustración 24 - Kanban board

En la ilustración anterior, se puede visualizar un ejemplo de algunas tareas del tablero de *Kanban* de la etapa de investigación.

El proceso de investigación se basó en gran parte en la realización de pruebas de concepto que permitieron evaluar la viabilidad del uso de algunas posibles funcionalidades del sistema. Estas pruebas se realizaron en su mayoría sobre APIs de fuentes de datos y administradores de anuncios. A modo de ejemplo, las pruebas de concepto sobre la API de *MercadoPago* fueron fundamentales para corroborar que esta era una fuente de datos usable por el sistema. A su vez, las realizadas con *Facebook* y *Google Ads* permitieron, además de entender cómo funcionan, aprender sobre diversos flujos que fueron fundamentales en el desarrollo y arquitectura del sistema. Sin estas, posiblemente muchas de las decisiones tomadas con relación al desarrollo hubieran resultado en una serie de cambios que pudiesen haber impactado directamente sobre la viabilidad del proyecto. En el siguiente Anexo 13.10 se muestra evidencia de estas.

Vale aclarar que, si bien estas comenzaron en la etapa de investigación, se prolongaron durante la fase de desarrollo.

A continuación, se observa una ilustración de una colección de *Postman* albergando diferentes pruebas de concepto de las APIs.

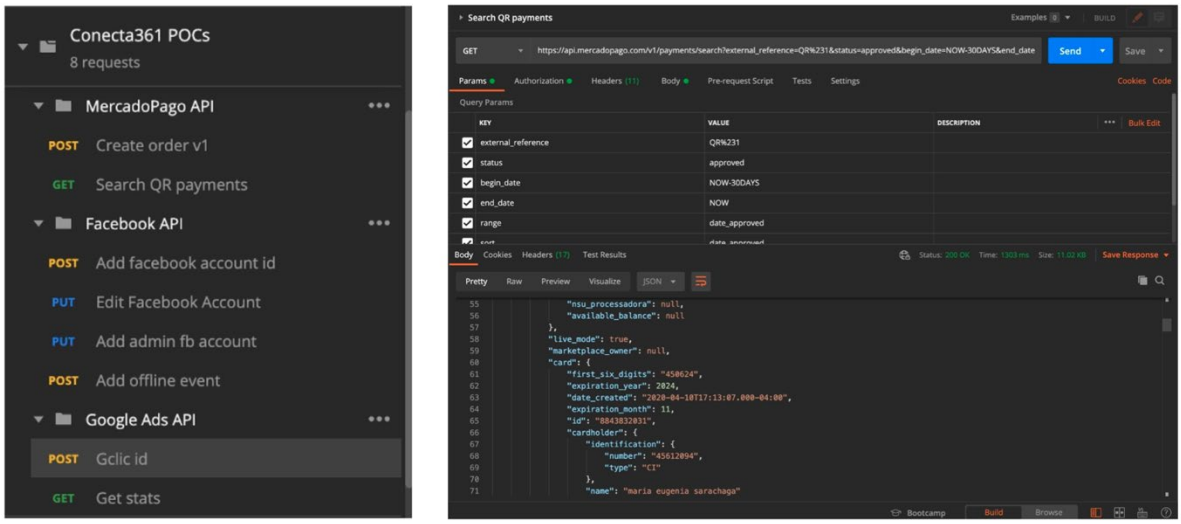


Ilustración 25 - Pruebas de concepto en *Postman*

Por otro lado, el equipo investigó sobre plataformas similares como *LeadsBri*dge, *InConcert* y *Segment*, con el propósito de buscar inspiración en las mismas y poder luego aportar nuevas soluciones al cliente. De prácticamente todos estos sitios se logró recopilar algún tipo de información, ya sea de posibles funcionalidades a realizar, o de ideas de diseño para ciertas secciones del sistema, aprovechando y maximizando la efectividad y eficiencia del tiempo de investigación.

5.1.2 Técnicas de relevamiento de requerimientos

Durante y luego de la etapa de investigación, y una vez que el equipo tuvo más claro el dominio del problema, se comenzaron a aplicar diferentes técnicas de relevamiento de requerimientos. Es necesario destacar que la etapa de investigación fue fundamental para poder aplicar las técnicas que se mencionarán a continuación, ya que la interacción con las diferentes APIs realizadas en las pruebas de concepto, así como el conocimiento adquirido durante el proceso, incrementaron la proactividad del equipo en sugerir ideas y nuevas soluciones para colaborar con el cliente.

En una primera instancia, el equipo optó por definir a grandes rasgos lo que serían las épicas del sistema, es decir las más grandes funcionalidades de este, para luego ir iterando sobre las mismas y definir las distintas historias de usuario de cada una [56]. De esta manera, el equipo logró pasar de tener grandes ideas de funcionalidades o procesos para el sistema, a una cantidad mucho más granular de *user stories*, las cuales más adelante pudieron ser divididas en tareas para desarrollar. En la siguiente imagen se ilustra dicha aplicación:

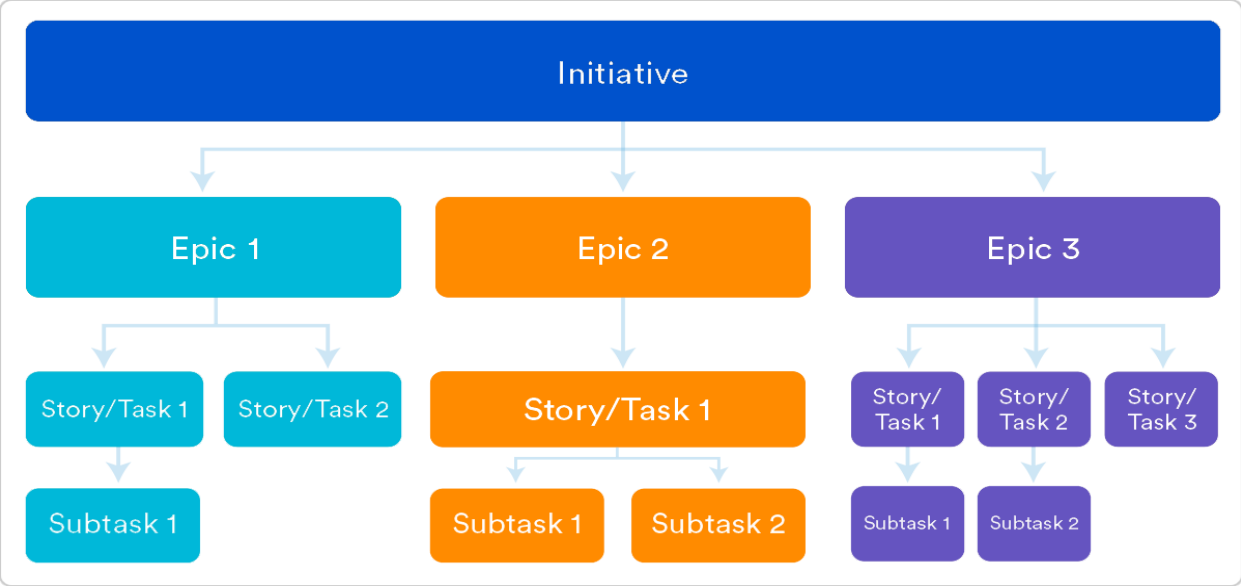


Ilustración 26 - Diagrama de *epics* y *user stories* [141]

Historias de usuario

Los requerimientos se especificaron a partir de la realización de historias de usuario con criterios de aceptación [57]. La utilización de las *user stories* permitió administrar de forma rápida los requisitos de usuarios ante la volatilidad de los requerimientos sin necesidad de escribir documentos formales y extensos, en lineamiento con los principios agiles. El hecho de comenzar la especificación a partir de las historias de usuario permitió mover el foco de la escritura de los requerimientos hacia la discusión de estos.

Para la narrativa de las *user stories* se siguió el siguiente formato [58]:

Como <tipo de usuario>, quiero <funcionalidad> para <valor o beneficio>

A su vez, se utilizó *"Desarrollo Guiado por Comportamiento"* o *Behaviour Driver Development (*BDD por sus siglas en inglés*)* para las historias que tenían complejidad alta de modo de describir la narrativa de la historia de usuario. La siguiente ilustración permite visualizar un ejemplo de formato de historia de usuario utilizando BDD:

ID: 1	**TÍTULO:** Procesamiento de datos del consumidor.	**PRIORIDAD: ALTA**	**ESTIMACIÓN: -**
COMO Cliente de Conecta361 **QUIERO** procesar los datos del consumidor final en el sistema **PARA** luego analizar si hubo una conversión o no.			
Dado un consumidor final que asiste a la tienda **CUANDO** realiza la compra **ENTONCES** el mismo es ingresado al sistema con sus datos correspondientes.			

Ilustración 27 - Historia de usuario con BDD

Para la gestión de las historias de usuario el equipó comenzó utilizando la herramienta *Google Sheets* y cuando estas empezaron a escalar y necesitar de una gestión más completa se pasó a utilizar *Jira*. El equipo consideró distintas herramientas para la gestión del trabajo, como por ejemplo *Trello*, pero terminó optando por Jira por distintas razones. En primer lugar, varios de los integrantes ya lo utilizaban en sus respectivos trabajos, por lo que contaban con conocimiento previo de la herramienta. Además, y como razón principal, se observó que *Jira* contiene muchas más funcionalidades de análisis, lo cual permitiría brindar mejores métricas de la gestión del proyecto.

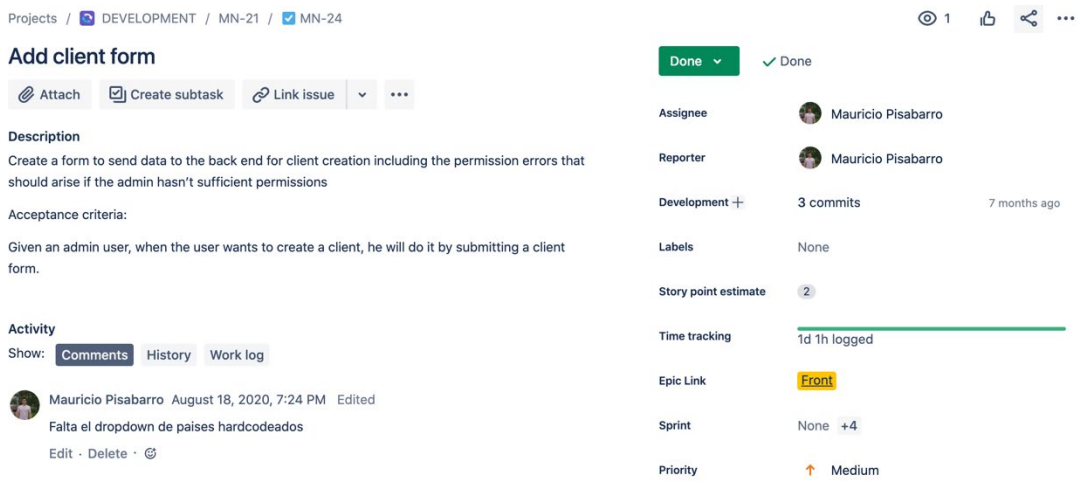


Ilustración 28 - Requerimiento con criterio de aceptación en *Jira*

Tal como se puede visualizar en la imagen anterior, se muestra un ejemplo un requerimiento especificado en *Jira* con su respectivo criterio de aceptación

A continuación, se detallan a las diferentes técnicas de relevamiento de requerimientos que permitieron bajar a tierra las historias de usuario:

Wireframes, mockups y prototipos

La búsqueda y análisis de sistemas similares al del equipo realizada durante la fase de investigación, permitió comprender a nivel visual como presentar algunas de las

funcionalidades del sistema. Además, fue una forma de validar los requerimientos con el cliente de manera de obtener retroalimentación.

Se utilizaron *wireframes* para poder plasmar la primera visión de la plataforma en un primer borrador. Este boceto representó de forma sencilla y esquemática la estructura de la aplicación [59].

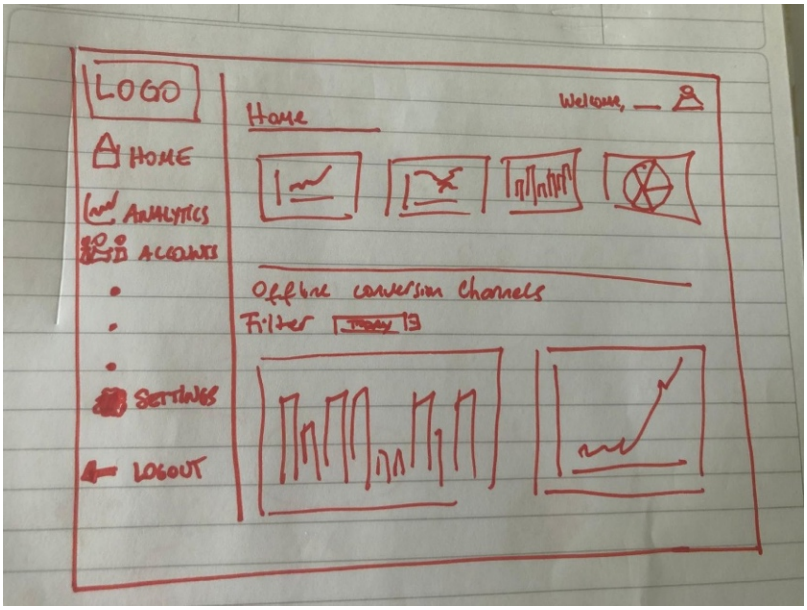


Ilustración 29 – Ejemplo de *wireframe* a lápiz y papel

Estos bocetos – que fueron hechos a lápiz y papel – ayudaron a organizar las funcionalidades y la información de cada interfaz. Gracias a estos, se tendieron puentes entre los requerimientos que se debían implementar y donde estos se debían colocar gráficamente.

Luego de realizar bocetos, se procedió a la creación de *mock-ups* y prototipos gracias a un mayor entendimiento de las pantallas por parte del equipo. Para la realización de estos se utilizó la herramienta Axure RP, debido a su facilidad de uso y conocimiento por parte de algunos de los integrantes del equipo. *Axure* permitió darles comportamiento a las pantallas mediante la interacción de sus componentes, así como la definición de estilos *CSS* y otras herramientas que ayudaron a validar con el cliente los flujos de la aplicación.

Estas validaciones tuvieron lugar en varias ocasiones, generalmente luego de la finalización de cada *milestone*, debatidas con el cliente de forma de discutir los próximos pasos o mejoras.

A continuación, se visualiza el prototipo realizado en base al *wireframe* mencionado anteriormente. Este pertenece a la pantalla de inicio de un administrador.

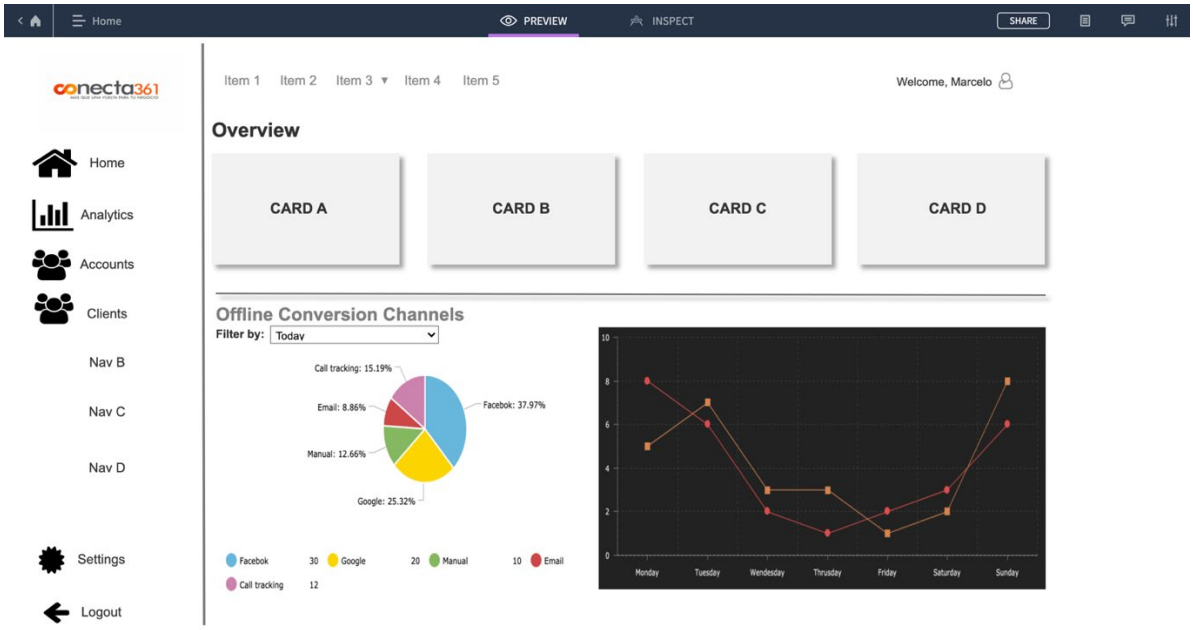


Ilustración 30 - Prototipo de pantalla de inicio de administrador

También se realizaron validaciones con uno de los clientes de Conecta361 que es MundoTrabajo. Se le pidió al administrador que navegue por las distintas pantallas con el objetivo de observar la interacción con el sistema y recolectar su *feedback* al respecto. Estas se pueden ver en detalle dentro de la sección Validaciones.

Brainstorming

El equipo tuvo varias instancias en donde se presentaban algunos desafíos a la hora de evaluar la viabilidad de ciertos requerimientos. Estos se daban por inquietudes del cliente, o por parte de las investigaciones realizadas por el equipo.

Por ejemplo, en la fase de investigación, el equipo se dio cuenta que, para automatizar la obtención de las ventas de todos los clientes, el sistema debía contemplar todas las plataformas que estos utilizaban. Sin embargo, algunas de estas no contaban con APIs, otras eran excesivamente caras y otras simplemente eran inviables para los casos de uso de los requerimientos.

Fue allí donde por primera vez, y luego en varias instancias posteriores, se utilizó la técnica de lluvia de ideas – mediante el uso de la herramienta *Miro* - para proponer soluciones y respuestas a un problema específico. En el ejemplo puntual, responder a la siguiente pregunta:

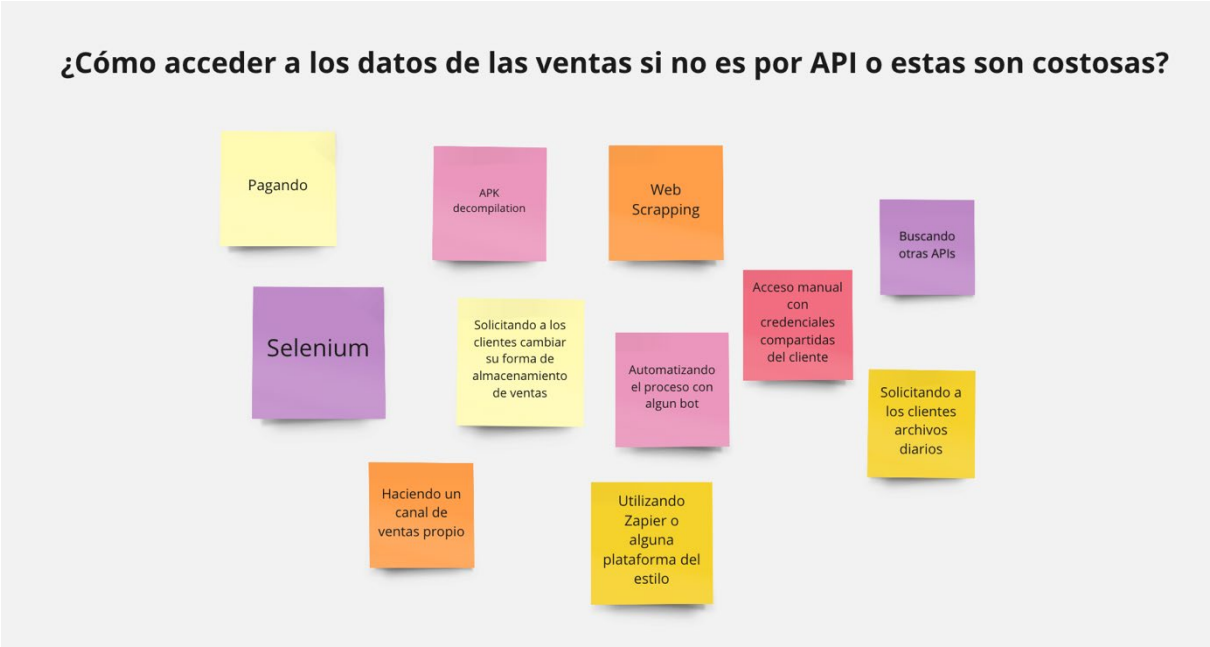


Ilustración 31 - *Brainstorming* con *Miro*

De esta instancia de interacción y construcción de ideas colectivas fue que surgió la solución al problema, mediante la utilización de *web scrapping*, detallado más adelante en el capítulo de Arquitectura y desarrollo.

Con esta misma técnica surgieron otros requerimientos tales como el desarrollo de una *SDK* así como también la creación de un API pública para los clientes de *Conecta361.*

Es importante destacar que el *brainstorming* no fue únicamente utilizado para los procesos de ingeniería de requerimientos, sino que fue una técnica adoptada por el equipo para otras como la gestión de riesgos.

5.2 Listado de requerimientos

En esta sección se la listan los requerimientos funcionales y no funcionales del sistema a alto nivel. Cabe destacar que los mismos fueron modificados durante el proyecto. Los requerimientos en formato de *user stories* se pueden encontrar en el Anexo 13.11

5.3 Requerimientos Funcionales

A continuación, un breve resumen de los principales requerimientos funcionales definidos al comienzo del proyecto. Los mismos están agrupados según las tres soluciones que ofrece el sistema: la aplicación web, la *SDK* y la API de público acceso.

5.3.1 Aplicación Web

Manejo de sesión

El sistema debe permitir a todo usuario registrado, ingresar a la web con correo electrónico y contraseña. A su vez, todos estos deben poder cerrar sesión y restaurar su contraseña. Debe poder reconocer las credenciales ingresadas por el usuario y determinar el rol de acceso al sistema. Estos deben ser:

- Super Admin / administrador de Conecta361
- Admin / administrador de cliente de Conecta361
- User / usuario de cliente de Conecta361

Administración de clientes

El administrador del sistema debe permitir dar de alta, modificar, ver detalles o eliminar clientes administradores. El alta del cliente debe ser realizada por invitación de un administrador de *Conecta361*, mediante el ingreso de un correo electrónico. Esta invitación es enviada por mail. Luego, el cliente invitado debe activar su cuenta creando una contraseña para acceder a la plataforma.

Administración de usuarios

El administrador del cliente debe permitir dar de alta, modificar o eliminar usuarios del cliente. El alta del usuario debe ser realizada por invitación del administrador del cliente ingresando únicamente el correo electrónico. Luego, el usuario debe activar su cuenta creando una contraseña para acceder a la plataforma. La invitación es realizada por email.

Administración de fuentes de datos

El sistema debe permitir a los administradores activar, desactivar y configurar las fuentes de datos para iniciar el procesamiento de ventas. Dentro de las configuraciones a realizar, se encuentran el manejo de credenciales y rangos de ejecución del procesamiento (cada 1, 6, 12 y 24 horas).

Administración de procesadores

El sistema debe permitir a los roles administradores activar, desactivar y configurar los distintos procesadores para poder enviar los eventos de las compras. Dentro de las configuraciones a realizar, se encuentra la vinculación a una cuenta publicitaria y creación o sincronización de la campaña digital a asignar los procesamientos.

Administración de compradores

El sistema debe permitir dar de alta, modificar, o eliminar compradores. El alta del comprador puede incluir nombre, apellido, teléfono, email, país, sexo, ciudad, código postal, siendo nombre o email campos requeridos. El sistema debe restringir la duplicidad de compradores.

Importación de compradores

El sistema debe permitir importar los compradores a través de un archivo. Debe admitir los siguientes formatos .csv, .xls, .pdf provenientes de plataformas populares como son Mailchimp y Zoho.

Importación de eventos

El sistema debe permitir importar los eventos a través de un archivo. Debe admitir los siguientes formatos .csv, .xls. El contenido del archivo debe cumplir con el formato del procesador seleccionado. En caso de que se desee utilizar un solo archivo para importarlo a todos los procesadores, el sistema cuenta con un formato personalizado configurable por cliente.

Obtención de eventos mediante web scraping

Para aquellas fuentes de datos que no tengan APIs disponibles para la consulta de sus datos, el sistema debe de extraer a estos mediante técnicas de *scraping*.

Ver estado de un evento en tiempo real

El sistema debe permitir ver el estado de un evento en tiempo real. Los posibles estados deben ser: pendiente o entregado.

Construcción de perfil de comprador

El sistema debe poder construir un perfil de comprador que permita mejorar los datos de los compradores a enviar a los procesadores.

Ver detalle de un comprador asociado a un evento

El sistema debe permitir ver el detalle completo del perfil del comprador de un evento. En este se debe visualizar si ya hubo eventos asociados a este, y la cantidad de conversiones atribuidos si aplica.

Estadísticas de *Conecta361*

El sistema debe permitir mostrar estadísticas relacionadas al rendimiento de las campañas de los clientes de *Conecta361*. Para el caso del administrador de *Conecta361*, este debe poder elegir ver cualquiera de las estadísticas de un cliente en particular. En estas se deben incluir detalles mensuales, anuales y de rangos personalizados de:

- Cantidad de conversiones atribuidas.
- Cantidad de eventos procesados.
- Porcentaje de conversiones atribuidas sobre eventos procesados.
- Porcentaje comparativo con respecto a los rangos seleccionados pasados.
- Porcentaje de uso y de conversión de las diferentes fuentes de datos.
- Porcentaje de uso y de conversión de los diferentes procesadores de eventos.

A su vez, se debe permitir ver la cantidad de dinero invertido en el uso de las campañas, y el dinero atribuido por las conversiones offline.

Rankings

El administrador de *Conecta361* debe poder visualizar rankings de los clientes más performantes de forma mensual, anual o personalizada.

Los rankings deben ser realizados según:

- Clientes con más conversiones.
- Clientes con mayores ganancias atribuidas.
- Mejor relación eventos enviados / conversiones atribuidas.

Listados

El sistema debe permitir a *Conecta361* visualizar listados de todos sus clientes y sus eventos. Los administradores de los clientes deben poder visualizar listados de sus compradores, usuarios y eventos. En la función ver detalles, se debe poder visualizar todos los datos del comprador. El orden de los elementos a listar depende del caso de uso de estos. Para el caso de los eventos, se deben ordenar por fecha de creación ascendiente. Para el caso de los clientes y usuarios, deben ser ordenados por estado activo y dentro de estos alfabéticamente ascendiente.

Filtrados

El sistema debe permitir realizar filtrados sobre los clientes, compradores y eventos con la finalidad de encontrar la búsqueda de forma más eficiente. Los clientes se deben poder filtrar por nombre, descripción y país. Los compradores por nombre, apellido, país, y sexo. Los eventos por proveedor, país, estado, fecha de creación y fecha de ejecución.

Paginados

El sistema debe poder paginar los listados de forma de estructurar el contenido agrupándolo en una cantidad fija de elementos por página.

Borrados lógicos

Se debe hacer un borrado lógico y no físico de los clientes, usuarios, fuentes de datos y compradores. De esta manera se puede llevar un historial de las diferentes acciones evitando eliminar la información.

5.3.2 API

Se debe exponer una API que permita a clientes de *Conecta361* integrarse directamente con el sistema de la aplicación. En este se deben permitir realizar las mismas funciones que a través de la aplicación web, mediante un conjunto de credenciales de desarrollo otorgadas previamente. La API debe permitir probar la solución en el ambiente de *staging,* como paso previo a la salida a producción.

5.3.3 SDK

Se debe proveer una *SDK* que permita a clientes de *Conecta361* integrarse directamente con el sistema de la aplicación. En este se deben permitir realizar las mismas funciones que a través de la aplicación web, mediante un conjunto de credenciales de desarrollo otorgadas previamente. La *SDK* debe permitir probar la solución en el ambiente de *staging,* como paso previo a la salida a producción.

5.4 Requerimientos No Funcionales

En la siguiente sección se describen los cinco atributos de calidad más relevantes de la solución. Además, se describen otros requerimientos no funcionales a destacar. En la sección Requerimientos de arquitectura se amplían los atributos de calidad junto con escenarios de uso.

La interoperabilidad es el atributo de calidad más importante que fue evaluado por el equipo. Este punto es fundamental ya que la solución se basa en la extracción de datos

y procesamiento de eventos de servicios externos por lo que el intercambio de información entre los mismos es constante.

Por otro lado, el cliente solicitó poder agregar a futuro nuevas fuentes de datos y administradores de anuncio por lo que la modificabilidad debe estar contemplada para soportar cambios en el sistema con el menor impacto posible.

La seguridad es fundamental debido a que los datos manejados en el sistema son reales y contienen información sobre documentos personales, direcciones y teléfonos, entre otros. Las manipulaciones de estos deben estar controlados por una correcta autorización al sistema.

La performance y la usabilidad no fueron explícitamente solicitados por el cliente, pero el equipo considero que son importantes en todo sistema de hoy en día. El sistema debe responder a tiempo y ser usable para que su navegación no se vuelva tediosa.

5.4.1 Principales atributos de calidad

RNF01 – Interoperabilidad

La interoperabilidad es el grado en que dos o más sistemas de intercambiar información a través de interfaces en un contexto en particular [60]. La solución del sistema consiste en intercambiar información con distintos servicios externos, desde fuentes de datos a procesadores. A su vez, el sistema expone APIs de acceso público, así como una *SDK* para una integración directa con la solución, lo que requiere que la solución de esta siga estándares ya conocidos y utilizados en la industria como lo es *Representational State Transfer* (REST por sus siglas en ingles).

RNF02 – Modificabilidad

La modificabilidad apunta a los cambios y se centra en el riesgo y el costo de hacer esos cambios [61]. Debido a que a futuro *Conecta361* van a querer extender a la solución

agregando nuevos procesadores, sean *TitkTok*, *Microsoft* u otro, este punto es fundamental. Además, el sistema debe ser adaptable a las fuentes de datos utilizadas por los clientes de *Conecta361*, que pueden variar desde plataformas populares como *MercadoPago*, a algunas más locales y desconocidas como *Memory.* En caso de que cualquiera de estos servicios cambie su versión, o se deban de modificar, el sistema debe poder sustituir unos con otros sin reacción al cambio.

RNF03 – Disponibilidad

La disponibilidad refiere a una propiedad del software que está listo para continuar con las tareas cuando es necesario [62]. Al ser un sistema planeado para estar en producción, se debe minimizar el tiempo en que el sistema este fuera de servicio mediante la mitigación de defectos.

RNF04 – Seguridad

La seguridad es la medida de la capacidad de los sistemas para proteger los datos y la información de accesos no autorizados y al mismo tiempo proporcionar acceso a las personas y sistemas autorizados [63]. La información que maneja el sistema debe ser administrada únicamente por aquellos usuarios autorizados a accederla. Debido a que es una plataforma multi-cliente, con muchos de estos siendo competencia, es importante distinguir mediante roles las capacidades de estos, imposibilitado que un cliente pueda obtener información de otro cliente.

RNF05 – Testeabilidad

"El testing lleva al fracaso, y el fracaso lleva al entendimiento" - Burt Rutan. La testeabilidad del software se refiere a la facilidad con que se puede lograr que el software demuestre sus fallas a través de pruebas [64]. Debido al constante cambio en los requerimientos del sistema, es importante identificar las fallas del sistema cuando estos cambios se producen. A su vez, hay que tener en cuenta que los datos calculados son luego utilizados para armar presupuestos e inversiones, por lo que se debe asegurar que

la información entregada sea la correcta. Es importante, a su vez, contar con diferentes ambientes de entorno que permitan realizar las pruebas correspondientes sin afectar datos del mundo real.

5.4.2 Otros atributos de calidad

RNF06 – Performance

La plataforma trata del tiempo y la capacidad de los sistemas de software en cumplir con los requerimientos de tiempo y cadencia [65]. Si bien la performance no es un atributo que deba de ser crítico en la arquitectura de la solución, todo sistema debe contar una performance acorde a sus necesidades. El objetivo de la solución es poder procesar eventos de múltiples clientes en simultaneo, desde muchas plataformas diferentes y de lugares diferentes. Es necesario, aunque no primordial, que la plataforma pueda tener una comunicación fluida con estos para procesar los eventos lo más cercano al tiempo real posible.

RNF07 – Usabilidad

La usabilidad se relaciona con lo fácil que es para el usuario realizar una tarea deseada y el tipo de soporte al usuario que le proporciona el sistema [1]. Se busca proveer una interfaz sencilla y fácil de usar para que *Conecta361* y sus clientes puedan interpretar bien los resultados de las campañas sin necesidad de contar con experiencia previa en la temática.

5.5 Restricciones

La única restricción expresamente solicitada por el cliente para con la plataforma fue que este soporte al menos los idiomas de español e inglés, con la opción a añadir más en caso de ser necesario.

6. Arquitectura y desarrollo

De modo de entender las decisiones y tácticas que fueron tomadas para definir la arquitectura, es importante atender los requerimientos de arquitectura previamente listados en el capítulo de Ingeniería de Requerimientos. Para su análisis, se describirán pequeños escenarios que ayuden a explicar el detalle de cada uno de estos. A su vez, se describen las diferentes tecnologías aplicadas, así como también aquellas que puedan resultar de interés para enriquecer la solución en el futuro.

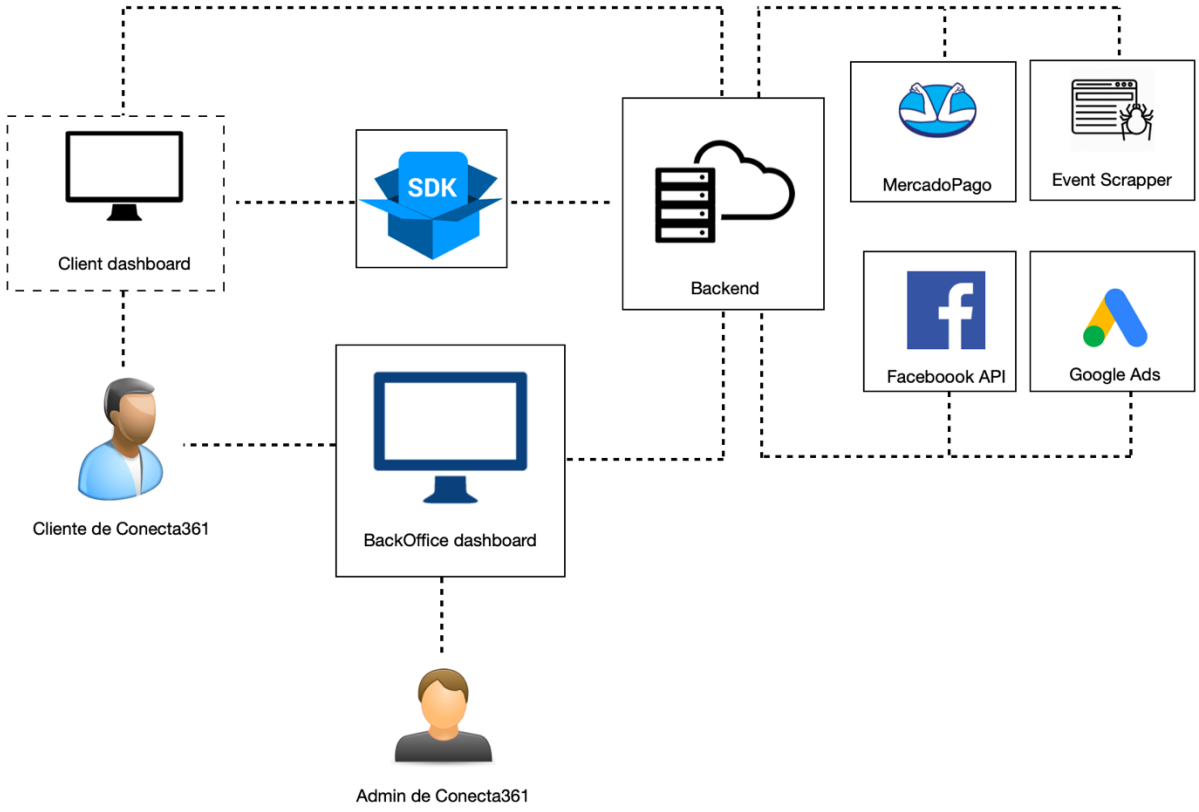


Ilustración 32 - Diagrama de alto nivel de arquitectura

Tal como se puede visualizar en la ilustración de arriba, la arquitectura propuesta consta de ocho componentes. Cuenta con un *backoffice* para que administradores y clientes de *Conecta361* puedan visualizar las estadísticas de sus campañas publicitarias, obtener datos de sus compradores y administrar la configuración de proveedores y fuentes de

datos. La *SDK* permite a los clientes de *Conecta361* desarrollar su propio *frontend* para la visualización y gestión de conversiones. El *backend* contiene toda la lógica y datos de la aplicación y una API de acceso público a los clientes. *Facebook API* y *Google Ads* como servicios para la generación y obtención de estadísticas de las campañas publicitarias. Por último, el *bot Event Scrapper* para obtener datos de los compradores a través de CRM's como alternativa al servicio de *MercadoPago.*

6.1 Requerimientos de arquitectura

Los requerimientos de arquitectura están compuestos por los atributos de calidad mencionados en el capítulo de junto con algunos requerimientos funcionales cuya implementación impacta en la arquitectura [66]. Algunos de estos fueron solicitados explícitamente por el cliente y otros fueron elegidos por el equipo, por considerarse relevantes para el dominio de la solución. A continuación, se detallan los principales atributos de calidad junto con escenarios basados en los requerimientos funcionales que permiten entender la importancia de su cumplimiento.

RNF01 – Interoperabilidad

- Se debe poder intercambiar información entre los servicios internos a través de diferentes canales de comunicación.
- Se debe proveer una API que pueda ser accedida por clientes de *Conecta361* para acceder a datos y estadísticas de sus campañas publicitarias.
- Se debe proveer una *SDK* para los clientes de *Conecta361* de modo de integrar su solución directamente al sistema sin pasar por la plataforma.
- Se deben poder generar y obtener datos desde y hacia las API's de proveedores externos como por ejemplo *Facebook Ads*.
- Se deben obtener datos de los compradores mediante distintas API's de terceros tales como *MercadoPago*.

- Se deben poder importar archivos de datos de los compradores de diferentes fuentes de datos y *CRM's.*

RNF02 – Modificabilidad

- Se deben poder agregar nuevas interfaces de APIs de administradores de anuncios para la generación y obtención de datos y estadísticas de las campañas publicitarias con el menor impacto posible. Estas interfaces podrían ser de otros procesadores tales como *Microsoft Ads* o *Tik Tok Ads.*
- Se deben poder configurar y sustituir los accesos a los proveedores externos desde un panel de administración sin costos de desarrollo.
- Se deben poder agregar nuevas interfaces de usuario para interactuar con la API de conversiones. Estas interfaces pueden ser personalizadas y desarrolladas por clientes o terceros.
- Se deben poder manejar diferentes formatos de eventos de ventas e identificación de compradores con el menor impacto posible sobre la lógica del sistema.
- El sistema debe permitir incorporar nuevas validaciones y transformaciones especificadas a los datos de venta.
- Se deben poder adaptar a cambios en los datos que se reciben.
- Se deben poder desplegar las aplicaciones en diferentes espacios físicos.

RNF03 – Disponibilidad

- En caso de que los eventos no se procesan por la indisponibilidad de alguno de los procesadores, estos deben ser reintentados hasta conseguir su procesamiento sin afectar la salud del sistema.
- Los clientes que utilicen la API de acceso público deben poder conocer el estado de salud de la aplicación. Se debe proveer un *endpoint HTTP* que indique al mismo.

RNF04 – Testeabilidad

- Se debe poder probar las funcionalidades críticas del sistema para minimizar los riesgos y probar que el sistema hace lo esperado.
- Se debe poder automatizar las pruebas de manera de no tener que requerir una intervención manual para probar cada componente del sistema.

RNF05 – Seguridad

- Los datos almacenados de los compradores deben ser únicamente modificados por usuarios autorizados (sistema basado en roles).
- Se debe poder asegurar la integridad de los datos.

Cabe destacar que también se cumplieron con aspectos de los atributos de calidad de performance y usabilidad, pero estos no se destacan entre los principales ya que se consideran relevantes en cualquier sistema.

6.2 Descripción de la arquitectura

Luego de haber mostrado la arquitectura a alto nivel y definido los principales requerimientos, se procederá a detallar la descripción de esta. El siguiente diagrama expone a los diferentes componentes del sistema y sus comunicaciones.

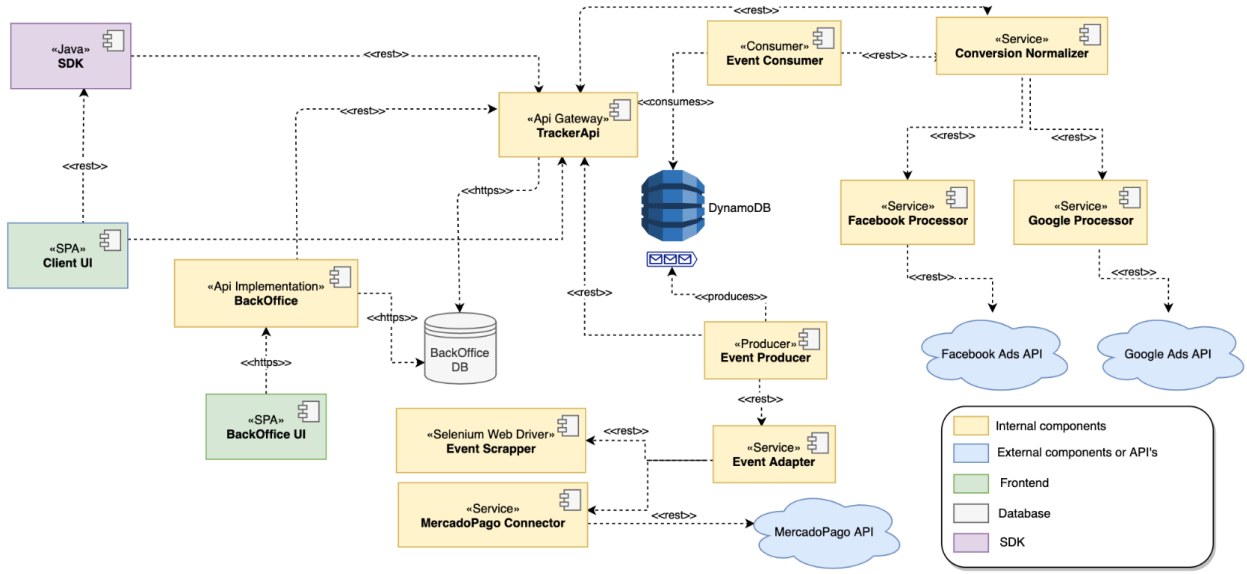


Ilustración 33 - Diagrama de componentes y conectores

A continuación, se listan los componentes detallados en el diagrama de arriba y una breve descripción de su función en el sistema:

Componente	Descripción
BackOffice UI	Interfaz de usuario para acceder a las funcionalidades provistas por *BackOffice*.
ClientUI	Interfaz de usuario de tercero para acceder a las funcionalidades provistas por la *SDK* o por Tracker API.
SDK	Kit de desarrollo de software que ofrece una interfaz para acceder a las funcionalidades del Tracker API.
BackOffice	API encargada de la gestión y mantenimiento de la plataforma web.
Tracker API	Es un API Gateway. Es el orquestador del sistema.
Event Producer	Servicio encargado de producir eventos de compra de las diferentes fuentes de datos.
Event Adapter	Adaptador que se encarga de procesar los datos provenientes del Event Producer y los transforma hacia formatos compatibles para el Event Scrapper o MercadoPago Processor.

Event Scrapper	*Bot* que se encarga de realizar *web scrapping* para la obtención de eventos de compra.
MercadoPago Connector	Procesador para la obtención de eventos de compra provistos por MercadoPago API.
MercadoPago API	API de MercadoPago.
Event Consumer	Servicio encargado de consumir eventos de compra para la posterior normalización de estos.
Conversion Normalizer	Servicio que cumple la función de normalizar los datos de eventos de compra en un formato genérico.
Facebook Processor	Servicio que obtiene datos de eventos en formato genérico, los transforma y posteriormente envía en formato compatible con la API de *Facebook Ads*.
Google Processor	Servicio que obtiene datos de eventos en formato genérico, los transforma y posteriormente envía en formato compatible con la API de *Google Ads*.

Facebook Ads API	API de *Facebook Ads.*
Google Ads API	API de *Google Ads.*
DynamoDB	Base de datos no relacional encargada de almacenar eventos de compra provenientes del Event Producer. Actúa como cola de mensajes asincrónica.
BackOfficeDB	Base de datos relacional que contiene toda la información de la plataforma web.

Tabla 2 - Descripción de los componentes de la arquitectura

Tras explicar la función de los principales componentes de la arquitectura, se profundizará sobre cómo esta permite resolver cada uno de los requerimientos mencionados teniendo en cuenta la comunicación entre los componentes ya graficados.

6.2.1 Interoperabilidad

La interoperabilidad es la capacidad de diversos componentes de un sistema o diferentes sistemas para operar con éxito mediante el intercambio de información, a menudo mediante el uso de servicios. Un sistema interoperable permite intercambiar y reutilizar información tanto interna como externamente [67].

El equipo consideró que la interoperabilidad es un requerimiento de arquitectura crucial dada las características del producto. El constante intercambio de información entre los

diferentes servicios del sistema obliga a definir estándares para su correcto funcionamiento e interpretación.

A su vez, la solución se destaca por su flexibilidad a la hora de comunicarse con las diferentes APIs de los servicios utilizados, sean Facebook, *Google Ads*, *MercadoPago* o cualquier otro en el futuro. El manejo de estas aporta un gran valor a la solución, por lo que saber utilizarlas de manera correcta es fundamental.

RESTful API como mecanismo de intercambio de información.

Todos los servicios del sistema fueron implementados bajo los principios de arquitectura REST, que se basan en el protocolo *HTTP* para el intercambio de información [68]. El equipo decidió que los mensajes deban ser transmitidos en formato *JSON*, y los *endpoints* compuestos en base a los recursos del sistema. Este mecanismo es ampliamente utilizado en la industria y permite una fácil integración para los servicios desconocidos que necesiten interactuar con este en un futuro [69]. Esto se debe a que las interfaces *REST* son simples, están hechas para ser autodescriptivas y manejan mecanismos de comunicación *stateless* [70]. Tal es el caso de posibles interfaces de clientes que tienen la posibilidad de integrarse directamente con *tracker-api*, como API gateway del sistema.

Servicios externos

La variada utilización de servicios externos al sistema implica que deba estar preparado para poder intercambiar información con estos, requiriendo que este sepa comunicarse e interpretar la información enviada y recibida entre ellos. Cada uno de estos servicios definen su contrato de comunicación y este debe ser correctamente utilizado por el sistema.

En la siguiente tabla se pueden visualizar los diferentes servicios externos con el cual interactúa el sistema y su tipo de comunicación.

Nombre	Descripción	Comunicación
Facebook Ads	API de administrador de anuncios de *Facebook*	API *REST*
Google Ads	API de administrador de anuncios de *Google*	API *REST*
MercadoPago	API de pagos de MercadoLibre	API *REST*
SendGrid	*Email Delivery Service*	API *REST*
MailChimp	*Cloud CRM*	*WebScrapping*
Zoho	*Cloud CRM*	*WebScrapping*

Tabla 3 - Comunicación con servicios externos

Kit de desarrollo de software (SDK)

La creación de una *SDK* permite a clientes externos operar con el sistema mediante una especificación sumamente restringida y sin posibilidad de cambio. De esta manera se les ofrece a los clientes la facilidad de integrarse con el sistema de manera fácil y rápida, con bajo riesgo de la existencia de un cambio en la SDK impactando sus sistemas. Actualmente la *SDK* está disponible en *Java* debido a la popularidad y robustez del lenguaje. No obstante, en el futuro se puede implementar la misma en nuevos lenguajes, según la demanda de clientes.

Algunas de las funcionalidades permitidas dentro de la *SDK* son:

- Envío de eventos de ventas para su procesamiento
- Creación, modificación y obtención de compradores
- Importación de archivos de pagadores de diferentes formatos

- Creación de usuarios administradores para el cliente
- Obtención de estadísticas según administrador de anuncios, fecha o comprador, entre otros.
- Vinculación de las cuentas publicitarias de los diferentes administradores de anuncios: campañas publicitarias y *datasets*.
- Habilitar/deshabilitar fuentes de datos tales como *MercadoPago* o *CRMs* específicas. También aquellas implementaciones que utilicen al *Event Scrapper*.

A continuación, se puede visualizar el *README.md* como guía para la implementación de la SDK en *Java*.

Tracker361 by Conecta361 Java client library

The official Tracker361 Java client library.

Installation

Requirements

- Java 1.8 or later

Gradle users

Add this dependency to your project's build file:

```
implementation "com.conecta361:tracker361-java-sdk:1.0.0"
```

Maven users

Add this dependency to your project's POM:

```
<dependency>
  <groupId>com.conecta361</groupId>
  <artifactId>tracker361-java-sdk</artifactId>
  <version>1.0.0</version>
</dependency>
```

Documentation

Please see the Java API docs for the most up-to-date documentation.

Usage

Credentials

```
String merchantKeySbx = "123456abcdef";
String secretKeySbx = "32weefs34534rfwesfsjyu67f";

Tracker361 tracker361Sandbox = new Tracker361.Sandbox(merchantKeySbx, secretKeySbx);
```

Create Event

```
public class CreateEventExample {

   public static void main(String[] args) {

      Address address = Address.builder()
                               .street("Rambla Republica del Peru")
                               .city("Montevideo")
                               .zipCode("11300")
                               .country("Uruguay")
                               .build();

      Payer payer = Payer
            .builder()
            .id("4-9934519")
            .address(address)
            .document("21329039050")
            .documentType("CI")
            .email("jhondoe@conecta361.com")
            .firstName("Jhon")
            .lastName("Doe")
            .phone("+598 98979695")
            .build();


      CreateEventRequest createEventRequest = CreateEventRequest
            .builder()
            .reference("123456789")
            .amount(new BigDecimal(500))
            .country("UY")
            .currency("UYU")
            .payer(payer)
            .provider("MP")
            .description("Compra QR MercadoPago")
            .creationDate("2020-11-26")
            .build();

      try {
         CreateEventResponse createEventResponse = tracker361Sandbox.client.createEvent(createEventRequest);

         // Handle response

      } catch (Tracker361Exception e) {
         // Handle errors
      }
   }
}
```

For more examples see the bundled tests.

Dependencies

Ilustración 34 - README.md de *SDK* en *Java*

API pública

De la misma forma que la *SDK* fue desarrollada para que clientes de *Conecta361* puedan conectarse con la solución sin necesidad de pasar por una interfaz gráfica, se expone una API que cumple con las mismas funcionalidades y brinda mayor flexibilidad que la *SDK* para su integración, pudiendo realizar más personalizaciones a su comportamiento. Los *endpoints* de la misma pueden ser encontrados dentro del siguiente Anexo 13.12.

Event Scrapper: *Bot* de extracción de eventos

Una de las dificultades que tuvo el equipo al intentar obtener eventos de las diferentes herramientas utilizadas por los clientes de *Conecta361*, fue que estas no tenían APIs para su uso público. El equipo comenzó utilizando la API de *MercadoPago* que permite extraer información de las ventas y compradores. Pero otras herramientas como *Zoho*, *MailChimp* o *Memory* no disponen de APIs o las mismas no son gratuitas para su uso público. Es por esta razón que el equipo analizó las diferentes formas posibles de no perder el rastreo de dichas ventas.

En primer lugar, el equipo implementó la posibilidad de que los usuarios puedan importar archivos de las diferentes plataformas de *CRMs* como las ya mencionadas. Si bien este requiere de un trabajo manual para obtener estos archivos, el sistema lograba incluir

Ilustración 35 - Vista de importación de archivos en *"Tracker361"*

estas ventas para su procesamiento de conversiones. En la siguiente ilustración se puede visualizar el proceso a través de la plataforma de *backoffice*.

Sin embargo, el equipo busco automatizar aún más el proceso de extracción de datos, llegando así a la implementación de un *bot* que tenga la responsabilidad de extraer información de los eventos y compradores.

El *Event Scrapper*, como fue denominado por el equipo, funciona en base a tecnologías de *web scrapping* como *Selenium Web Driver*. Este mismo es ejecutado cada un tiempo fijo configurado por cada cliente, y puede ser personalizado para que extraiga información de cualquier fuente de datos. De esta forma, el sistema abarca de manera automática la extracción de datos sin importar si sus fuentes son accesibles por API. A continuación, se puede ver el flujo del *bot*.

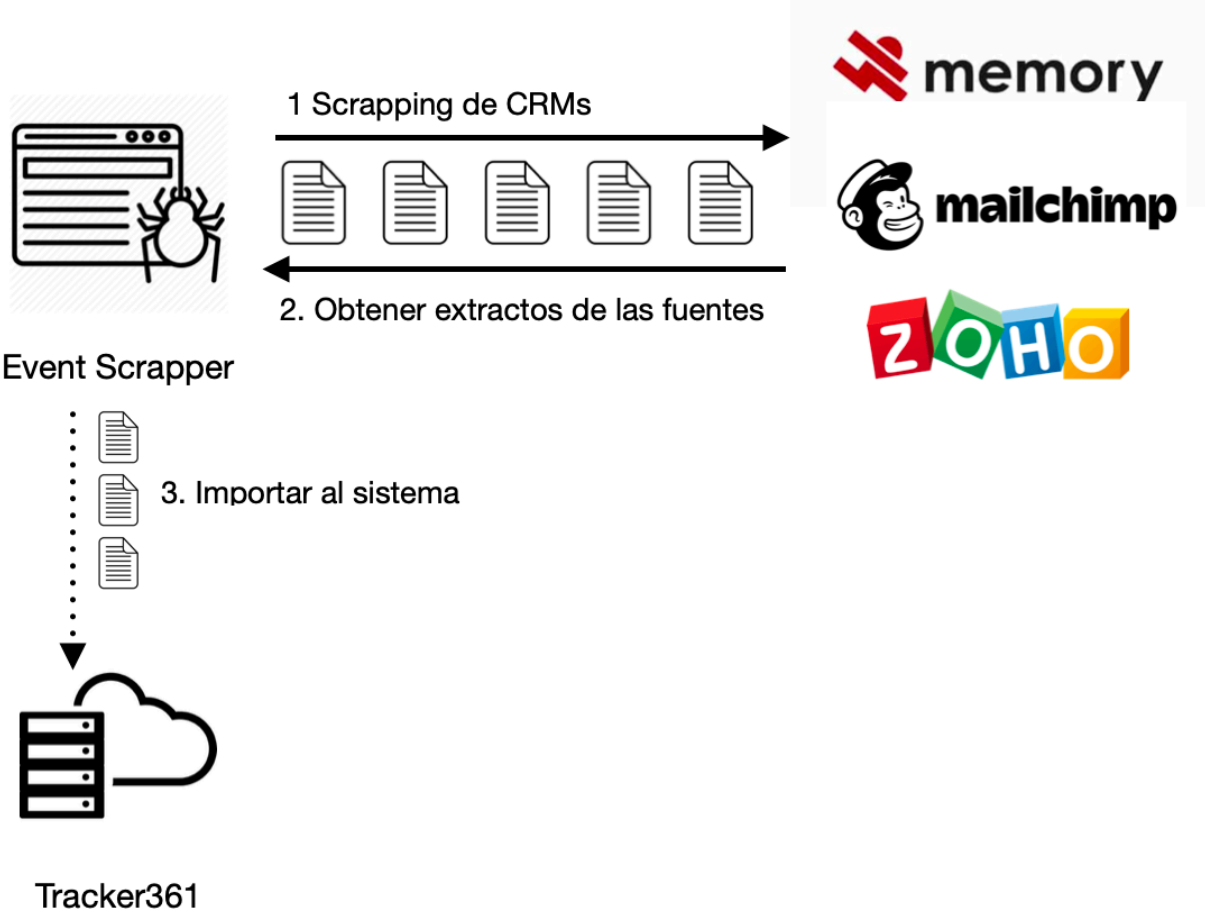


Ilustración 36 - Flujo de *scrapping* por *Event Scrapping*

6.2.2 Modificabilidad

"Si el cambio es la única constante en el universo, entonces el cambio en software no es solo constante sino ubicuo. (…) Estudios tras estudios demuestran que la mayor parte del costo del sistema de software típico se producen después de su lanzamiento en producción" [61]. La modificabilidad trata sobre como adaptarse al cambio, y qué tan preparado está el sistema para hacerlo [61] . Durante la etapa de desarrollo el equipo fue consciente de que debido a la incertidumbre en requerimientos la posibilidad de que surjan cambios en el software de manera constante era alta, incluso luego de su puesta en producción. Cambios en las *API's* de los proveedores, nuevas interfaces de fuentes de datos, alteraciones en las validaciones y formatos de eventos fueron algunas de las posibles modificaciones a realizar. Es por este motivo que el equipo hizo especial énfasis en construir una arquitectura que aumente la cohesión y reduzca el acoplamiento, además de utilizar variadas tácticas para mitigar el riesgo al cambio.

Se debe poder agregar o sustituir proveedores y fuentes de datos

Como se mencionó anteriormente, es necesario que los diferentes procesadores o fuentes de datos del sistema puedan ser sustituidos, modificados o agregados con un impacto de cambio bajo.

Para lograr esto, se determinó que la interacción entre el Event Consumer y los procesadores o fuentes de datos deban ser mediados por un intermediario especifico que actuará como un *wrapper.* Por ejemplo, para el caso de los procesadores, este se encarga de convertir los datos normalizados enviados por el *Conversion Normalizer* de manera que los diferentes procesadores puedan entender y viceversa. En el caso de que se quisiera agregar, modificar o sustituir un procesador por otro que provea el servicio mediante APIs, por ejemplo, *TikTok*, es necesario crear un nuevo componente

intermediario. En caso de querer eliminar uno, simplemente deberá tocarse el *Conversion Normalizer* para que no realice más peticiones al *wrapper* especifico.

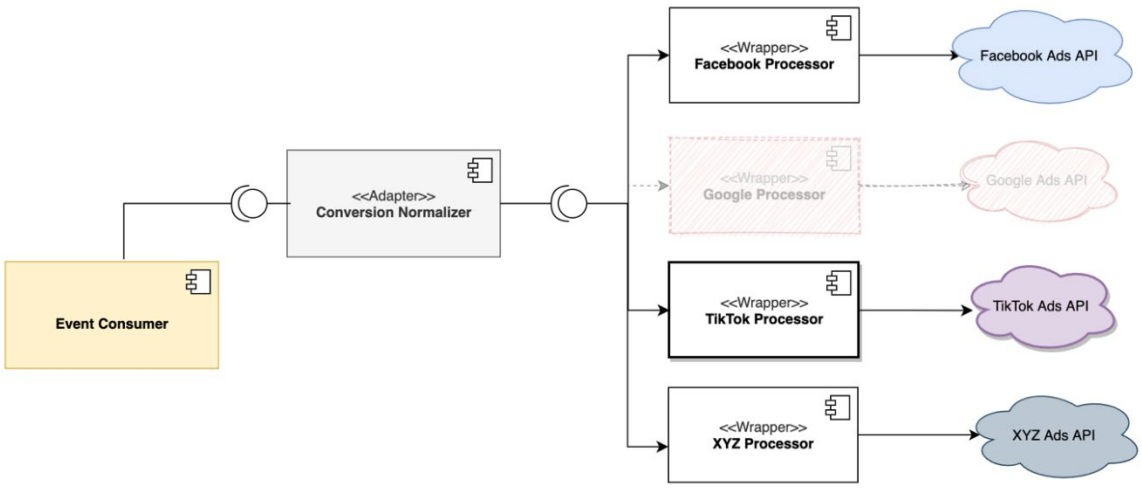


Ilustración 37 - Escenario de como sustituir, modificar o agregar un procesador

En la ilustración de arriba, se puede visualizar como el procesador de *Google Ads* es removido sin impacto de cambio, mientras que otro es sustituido bajo la misma premisa.

De esta forma, se termina reduciendo el acoplamiento entre los procesadores, lo que permite realizar cambios en uno sin que afecte a los otros.

Abstracción de servicios comunes

A medida que se fueron construyendo los primeros componentes del sistema, el equipo fue identificando comportamientos similares que se podían abstraer en un único módulo, utilizando la táctica de *refactoring* y abstracción de servicios comunes. De esta forma, cualquier tipo de cambio al servicio común ocurriría en un único lugar, reduciendo el costo de modificación. Por ello mismo, se decidió crear un módulo que agrupe comportamientos reutilizables entre los demás servicios del sistema. Este módulo, denominado *commons,* fue construido para ser utilizado como librería por los diferentes

servicios que necesiten de estos comportamientos. Algunas de estas abstracciones son servicios de clientes http, validaciones, excepciones, mails, *logging* y constantes.

De manera similar, se construyó luego otro modulo denominado *root-pom,* que alberga a las dependencias comunes de todos los proyectos del sistema, la especificación de sus versiones y algunos *plugins* provistos para la ejecución de los deploys correspondientes. Cabe destacar que este módulo consta de un único archivo llamado *pom.xml*, que utiliza a *Maven* como manejador de dependencias. De esta manera, utilizando la táctica de diferir enlaces, los procesos automáticos de la integración y despliegue continuo importaban sus dependencias de manera automática y facilitaban la ejecución de sus *pipelines.*

En el siguiente diagrama se pueden apreciar los usos entre los distintos módulos de alto nivel. En el mismo se puede ver como el *commons* contiene a los módulos de *constants, exceptions, models, spring* y *utils.*

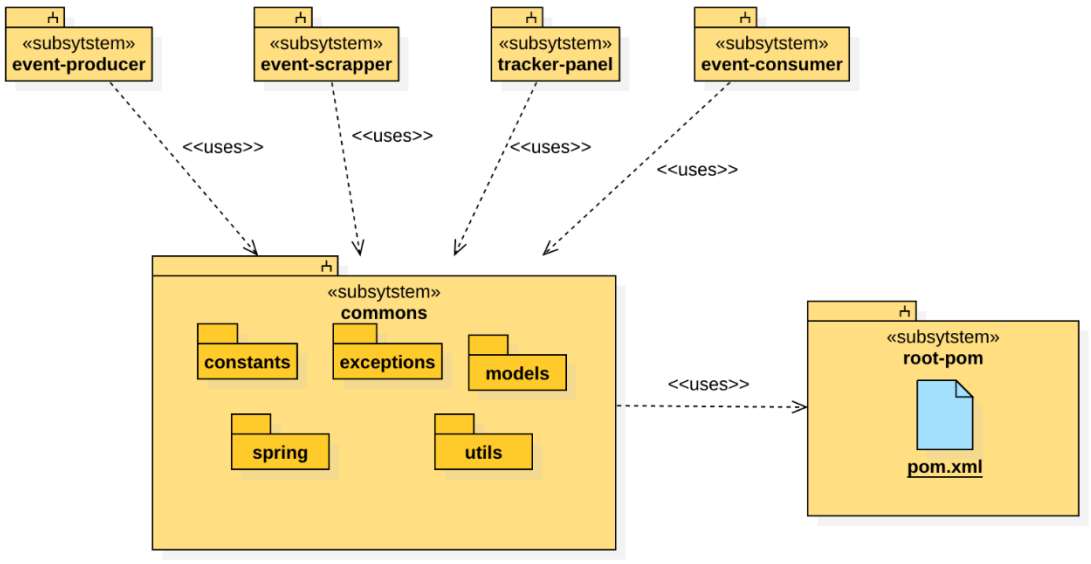


Ilustración 38 - Diagrama de usos del *commons*

Guardar la configuración en el entorno

El sistema desarrollado puede ser ejecutado en diferentes ambientes: local, *staging* y producción. Esto significa que los accesos a bases de datos, credenciales a servicios externos y otros *backing services* varían según su entorno y pueden ser modificados en el tiempo.

Siguiendo con los lineamientos de *The Twelve Factor*, las variables de entorno fueron configuradas en un archivo *application.yml* [71]. Estos están almacenados en los servidores físicos para no dar visibilidad en los repositorios. Esta táctica, denominada diferir enlaces, se utilizó para minimizar el impacto de cambio [61].

También es posible configurar las variables de entorno dentro del panel de *Travis CI*[4], modificando los valores como se muestra a continuación:

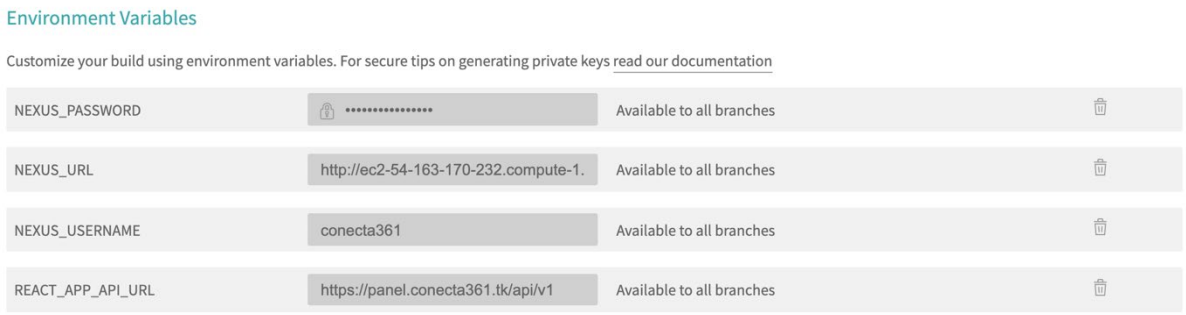


Ilustración 39 - Variables de entorno en *Travis CI*

[4] https://travis-ci.com/

Este punto también mejora la seguridad ya que todas las credenciales de base de datos, claves secretas de encriptación, *tokens* de accesos a servicios externos (*backing services*) y *urls* de servicios internos son manejadas a nivel de variables de configuración y segregadas según su entorno. Estas variables son cargadas en tiempo de ejecución mediante un archivo de configuración ubicada internamente en el servidor físico y a través de la herramienta *Travis CI*. De esta manera se mitiga el riesgo a robos de identidad y ataques a base de datos. El detalle de esta táctica es también descripto en Automatización del proceso de ingeniería de software: un acercamiento a *DevOps*.

Base de datos no relacional

Debido a que la información de los eventos de las ventas no mantiene una estructura estable, se optó por utilizar una base de datos *NoSQL* que permite almacenar información en formato *JSON* sin una estructura definida. Por ejemplo, algunos eventos de venta presentan la dirección, historial, producto y cedula de identidad del comprador y otros no.

Utilizar bases de datos documentales ofrecen la ventaja de permitir agregar nuevos campos con un impacto de cambio mínimo. Si en un futuro se decide que el procesador de eventos de compra valide un nuevo campo de este, basta con insertar la compra en formato *JSON* con este nuevo campo [72]. No hay necesidad de modificar la estructura de la base de datos, como habría que hacerlo en una base de datos relacional. El equipo utilizó en un primer lugar a *Couchbase*[5] como base de datos *NoSQL*, pero al darse cuenta

[5] https://www.couchbase.com/

de la complejidad y costos para su administración, se optó por una herramienta *cloud* y gratuita.

Luego de investigar distintas alternativas, realizar pruebas de concepto y validar su uso, el equipo inclinó por utilizar *DynamoDB*.

DynamoDB es una base de datos no relacional y clave-valor de *Amazon Web Services*, que permite el almacenamiento de información en formato *JSON* y un rendimiento de milisegundos a cualquier escala. Es una base de datos orientada a documentos, lo que quiere decir que, en lugar de guardar datos en registros, los guarda en documentos [73].

6.2.3 Disponibilidad

El equipo se puso como objetivo poder tener el producto en producción antes de la entrega final. Esto significa que, una vez productivo, se deberán monitorear fallas o defectos del sistema para poder mitigar el impacto causado de la manera más efectiva posible. Un producto defectuoso en producción puede causar problemas al cliente si los mismos no son identificados y tratados a tiempo. Por ejemplo, si existen fallas en los servicios de procesamientos de conversiones, las estadísticas de las campañas publicitarias no arrojarán datos válidos y pueden afectar a la estimación de presupuesto, causando perdida de dinero.

Recuperación de defectos

De manera de recuperarse de defectos causados por, por ejemplo, los servicios externos, es importante que el sistema pueda revertir a un estado conocido que funcione correctamente.

Es por este motivo que se utilizó la táctica de *rollback* y *exception handling*, el cual permitió tener un mayor control de las excepciones ocurridas en el sistema. Para esto se crearon traducciones de errores internos mediante una clase denominada *ExceptionHandler.*

A continuación, se puede visualizar un ejemplo de la aplicación de ambas tácticas mediante *annotations* de *spring*:

```
@Transactional(rollbackFor = Exception.class)
public void editClient(Integer clientId, ClientDto clientDto) throws Conecta361Exception {
    Client client = Precondition.checkNotNull(clientRepo.findById(clientId), ExceptionCode.CLIENT_NOT_FOUND);
    Precondition.checkTrue(clientId.equals(clientDto.getId()), ExceptionCode.INCORRECT_CLIENT);
    client = ClientMapper.INSTANCE.toEntity(client, clientDto);
    clientRepo.save(client);
}
```

Ilustración 40 - *Exception handling* en *Java Spring*

Los clientes deben poder conocer el estado de salud de la API pública

Con motivo de ofrecer una mayor calidad en el servicio a los clientes, estos deben poder conocer en tiempo real si existen problemas en el sistema. Es por esta razón que se tomó la decisión de realizar un *endpoint* que permita informar sobre el correcto funcionamiento del sistema.

A su vez, de manera de ser proactivos ante posibles defectos en el sistema, se utilizó la táctica *ping/echo* que permita realizar peticiones a ese *endpoint* cada 5 segundos. Esta fue configurada dentro de la infraestructura del sistema, que es detallada más adelante.

Monitoreo en la nube como servicio (CMaaS)

La decisión de utilizar un servicio en la nube que ofrezca herramientas de monitoreo a nivel de aplicación se debió a que ofrecen facilidades en la configuración y centralizan todas las herramientas bajo un único acceso y punto de entrada.

La herramienta elegida fue DataDog[6] dada su popularidad y acceso gratuito en su plan educativo. DataDog es una plataforma basada en *Software as a Service (SaaS* por sus siglas en ingles) para aplicaciones que ofrece supervisión, seguridad, analíticas y registro de logs [74]. Ante la incertidumbre sobre la futura mantenibilidad de parte del equipo, esta plataforma permite revocar accesos y crear otros de manera segura.

6.2.4 Seguridad

Al manejar información sensible de los usuarios, la integridad y confidencialidad de los datos es muy valiosa. Es imperativo restringir toda información sensible a aquellos que no estén permitidos a acceder o modificar esta [63]. La seguridad cobra aún más relevancia teniendo en cuenta que este es un producto que maneja datos de personas reales, pudiendo traer consecuencias financieras y legales.

Tokens de acceso para la autenticación y autorización de usuarios.

Para la utilización de la plataforma web se implementaron tokens de acceso basados en el standard *JWT (JSON Web Tokens)* que permiten decodificar, verificar y generar *tokens* con información relevante para la sesión tales como identificación y rol del usuario y fechas de creación y expiración de sesión. Son estos *tokens* de acceso son generados al iniciar sesión en la aplicación y enviados dentro de una *cookie* de sesión. Luego, para cada petición se envía la *cookie* con el *token* como contenido, siendo este un mecanismo de autorización *stateless.* Esto permitió identificar, autenticar y autorizar a los usuarios de la plataforma. A su vez, para los servicios que requieran ser llamadas únicamente por

[6] https://www.datadoghq.com/

ciertas interfaces se utilizó la táctica de limitar exposición mediante el mecanismo de seguridad *CORS* que permite configurar dominios permitidos para acceder a los *endpoints*.

En la siguiente ilustración se visualiza el flujo de *login*:

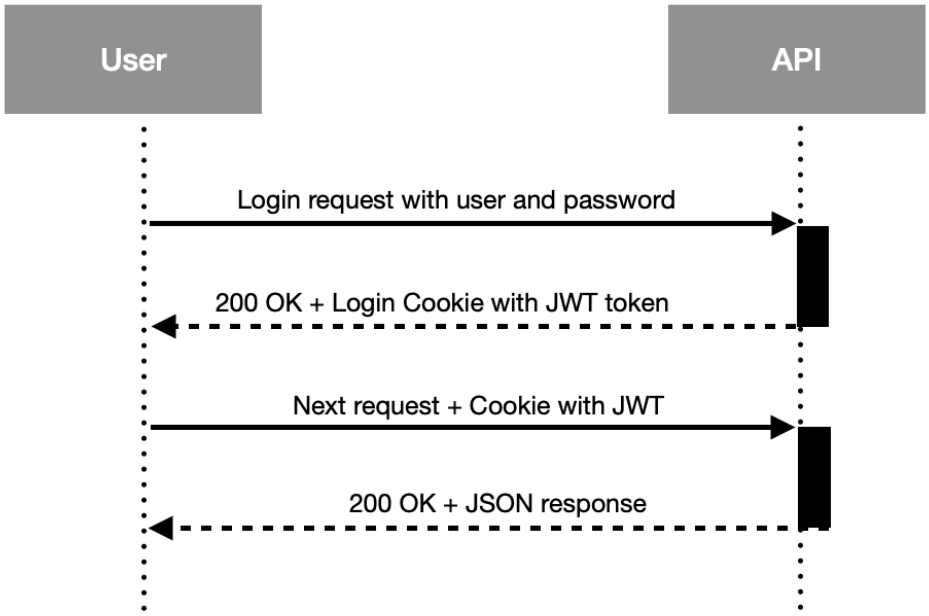


Ilustración 41 - Flujo de *login*

Sistema basado en roles

Se cuenta con la implementación de un sistema basado en el manejo de roles según la acción a realizar en el sistema. Cada *endpoint* del sistema restringe o habilita el acceso a los recursos según el rol y tipo de acción (lectura, escritura y edición). Esta implementación fue realizada mediante la utilización de *annotations* e *interceptors* provistos por el *framework* de *Java Spring* [75], que permite de manera fácil customizar comportamientos a nivel de entrada a la aplicación.

En la siguiente ilustración se puede visualizar el uso de estas *annotations* con sus roles en forma de enumerados.

```java
@PutMapping("/{id}/edit")
@Role(roles = EAccessRole.CLIENT)
public void editClient(@PathVariable Integer id, @Valid @RequestBody ClientDto clientDto) throws Conecta361Exception {
    clientService.editClient(id, clientDto);
}

@DeleteMapping("/{id}/delete")
@Role(roles = EAccessRole.SUPER_ADMIN)
public void deleteClient(@PathVariable Integer id) throws Conecta361Exception {
    clientService.deleteClient(id);
}
```

Ilustración 42 - Manejo de roles mediante *annotations*

Protocolo HTTPS y utilización de SSL

Todas las peticiones realizadas entre los servicios del sistema utilizan protocolo de transporte seguro mediante HTTPS. A su vez, estos cuentan con certificados SSL para intercambiar la información de manera segura y evitar ataques de *man-in-the-middle.* Los servicios públicos como el *tracker-panel-api* y *tracker-api* contienen dominios y certificados asociados a estos mediante los servicios cloud de *AWS* como *Route53*, *CloudFront* y *AWS Certificate Manager* y el gestor de dominios *Freenom*. Incluso, si la comunicación entre los servicios es interna se utilizan otros dominios de forma de diferenciar el acceso a los recursos. Por ejemplo, la comunicación entre el *tracker-panel* y el *tracker-api* se hace a través del siguiente dominio: https://internal.conecta361.tk y cuando la comunicación es externa se hace a través de https://api.conecta361.tk.

Encriptación de datos sensibles

Los datos sensibles de la aplicación son encriptados mediante un algoritmo de *hashing* HMAC SHA-512 y un *salt* autogenerado previo a su almacenamiento en base de datos. Tal es el caso de las contraseñas de los usuarios.

6.2.5 Testeabilidad

Para poder cumplir con una buena testeabilidad, es necesario que se cumplan dos condiciones: controlabilidad y observabilidad. La primera es la capacidad del sistema de poder controlar el estado interno de los componentes y la segunda es poder observar las salidas [76].

Limitar complejidad

Se realizó una arquitectura basada en servicios de forma de limitar la complejidad estructural y facilitar el *testing* del sistema [76]. Al realizar servicios como unidades independientes, ayudó a realizar el proceso de automatización de construcción, prueba y despliegue del sistema. Por ello mismo se utilizó la táctica de limitar complejidad, en donde se tomaron diferentes medidas:

Por un lado, se controlaron las dependencias cíclicas mediante el uso de controles por *Maven.* A su vez, se encapsularon las dependencias en un ambiente externo – *Nexus Repository* – para reducir las dependencias entre componentes.

Además, las tácticas utilizadas para satisfacer el atributo de calidad de modificabilidad ayudaron a mantener una buena testeabilidad, aumentando la cohesión y reduciendo el acoplamiento.

Sandboxing

Se utilizó la táctica *sandboxing* para mejorar el control y la observabilidad del estado del sistema [76]. El sistema cuenta *pipelines* de integración y entrega continua, por lo que fue importante tener un ambiente el cual sea independiente al de producción para realizar las pruebas pertinentes. En este proceso, se realizaron pruebas unitarias, funcionales y de integración. Esta táctica es detalla más adelante dentro del capítulo de Automatización del proceso de ingeniería de software: un acercamiento a *DevOps*.

6.2.6 Otros atributos de calidad

Si bien los atributos de calidad *performance* y usabilidad no estuvieron dentro de los principales, se destacan en esta sección ya que igualmente fueron tenidos en cuenta. Hoy en día, todo sistema debe tener buen rendimiento y ser fácil de usar. A continuación, se detalla cada uno.

Performance

- Single Page Application (SPA)

Para la interfaz de usuario del cliente y del *backoffice* se hizo una *single page application* (SPA por sus siglas en inglés). Se eligió este tipo de arquitectura de manera de lograr una mejor experiencia de usuario a través de tiempos de carga y respuesta más rápidos en comparación con otras arquitecturas *server side*. Esto es porque la aplicación web corre en el lado del cliente, y se va cargando de manera dinámica a demanda según las interacciones con el usuario, a diferencia de arquitecturas más tradicionales en donde la página entera tiene que ser cargada entre interacción e interacción [77].

Por otro lado, este tipo de arquitectura permitió una clara separación de responsabilidades y dependencias, lo cual facilitó avanzar en paralelo sin conflictos con el *backend*, efectivamente acelerando el desarrollo.

- Cola de mensajes asincrónica para el procesamiento de ventas

Se decidió utilizar una cola de mensajes que actúe como intermediario entre el *event-producer y el event-consumer* para que ambos módulos no se comuniquen entre sí y reducir el acoplamiento y mejorar la performance del sistema mediante la comunicación asincrónica. Fue implementada y desarrollada de manera nativo por el equipo mediante el uso de *DynamoDB* [78].

La decisión de elegir esta tecnología y no otras como *Amazon SQS*, se debe a que el equipo considero necesario buscar una cola de mensajes que a su vez sirva como unidad de almacenamiento para una auditoria futura en caso de ser necesaria. Las colas de mensajes tradicionales como la de Amazon SQS borran los registros una vez consumidos.

Sin embargo, al utilizar DynamoDB, la consumición de los registros de la cola ocurre a nivel lógico y no a nivel físico, mediante una propiedad llamada *dequeue* que alberga los valores 0 o 1. El componente event-producer es el encargado de producir nuevos eventos, los prioriza por fecha de creación de la compra y los inserta en la cola de mensajes. Luego, de manera asincrónica y paralalelamente, el event-consumer escucha sobre nuevos registros en la cola de mensajes y en caso afirmativo los consume para ser procesados. Una vez procesados, el event-consumer le cambia el valor de la propiedad *dequeue* en 1 pero sin borrar el registro dentro de la base de datos no relacional. Se procuró de realizar índices y optimizaciones de tabla para que se puedan realizar las filtraciones necesarias de manera instantánea.

El tener estos dos módulos separados, permite que, si en un futuro cambia la lógica de la generación o consumición de eventos, esta no impacte en el consumer y viceversa.

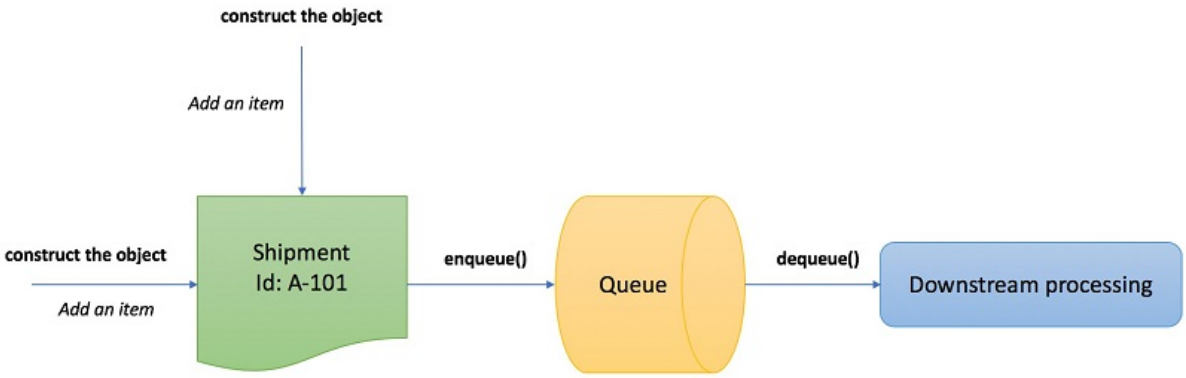


Ilustración 43 - Cola de mensajes con *DynamoDB* [73]

Usabilidad

En esta subsección se discute la elección de utilizar *Material Design* [79] como lenguaje de diseño por defecto por sobre diseñar todo de cero, o, dicho de otra manera, a mano.

"*Material* es un sistema de diseño - respaldado por código fuente- que ayuda a equipos crear experiencias digitales de alta calidad" [79]. En otras palabras, *Material Design* es un conjunto de pautas, patrones, y buenas prácticas de diseño que se ven implementadas en componentes de diseño creados por Google. Para este proyecto en particular se utilizó la librería de React *Material-UI* [80], la cual contiene componentes presentacionales que cumplen con el lenguaje de *Material*.

Material-UI permitió al equipo utilizar componentes de diseño ya probados, dejando como única dificultad encontrar una manera de integrarlos para que se mantenga la usabilidad y experiencia de usuario que tiene cada uno de los componentes por separado.

Como se mencionó en la introducción, la decisión de utilizar *Material* fue opuesta por la idea de diseñar toda la interfaz y experiencia de usuario de cero. Si bien utilizando esta técnica seguramente se lograse una interfaz completamente personalizada a los deseos del cliente, esto tenía como contrapartida un gran costo de desarrollo en cuanto a esfuerzo, y riesgo aumentado de llegar a producir algo poco usable con mala experiencia de usuario. Con esto en mente fue que se optó por la seguridad de utilizar algo ya probado, y la agilidad de no tener que crear diseños propios que provee *Material Design*, implementado en la librería *Material-UI* para React.

6.3 Tecnologías

Esta sección tiene como objetivo listar y justificar las tecnologías utilizadas para la construcción del sistema, así como el análisis de los principales problemas tecnológicos enfrentados.

El éxito en el diseño de la arquitectura no fue agnóstico a las tecnologías, y muchos de los requerimientos de arquitectura fueron logrados exitosamente gracias al uso y combinación de estas.

Se tuvo que decidir las tecnologías para utilizar en el *backend, frontend e infraestructura,* ya que el cliente dio libertad en la elección de estas. Las decisiones que llevaron a la selección de estas se basaron en los siguientes factores:

- Gran comunidad y respaldo
- Experiencia del equipo
- Populares y de vanguardia
- Curva de aprendizaje
- Recursos disponibles (presupuesto)

6.3.1 Backend

Lenguaje y framework

Para el desarrollo de la lógica del *backend* se utilizó el lenguaje *Java* bajo el *framework Spring*, utilizando *Apache Tomcat* como servidor *web y Maven* como gestor del proyecto. Se utilizó a este debido a la experiencia previa de algunos integrantes del equipo, cantidad de librerías disponibles y agregan adaptabilidad y estabilidad a la solución. Se utilizaron varios de las funcionalidades que ofrece *spring*, como el uso de *annotations* e *interceptors* para varias satisfacer varias de las tácticas de arquitectura, así como también manejo

asincrónico de tareas. A su vez, se utilizó *Maven* como manejador de dependencias, lo que permitió facilidad en la automatización del despliegue a los diferentes ambientes.

También se permitió la configuración de logs a través de diferentes librerías como *logback* y *log4j*.

6.3.2 Frontend

Framework

Para el *frontend* se buscó un *framework* que permitiera desarrollar una SPA, como se menciona en la sección de performance dentro de Otros atributos de calidad. El equipo consideró *React* y *Angular*, ya que son los *frameworks* más utilizados para esto, y al menos un integrante del equipo tenía experiencia en ellos. La decisión final fue *React* gracias a su superior flexibilidad, ya que técnicamente es una librería y no un *framework*, lo que significa que es posible realizar cosas de manera más sencilla, al no estar atado al flujo de un *framework* [81].

Lenguaje

React permite ser utilizado con *Javascript* por defecto, pero además brinda soporte completo para *super-sets* de *Javascript* como *Typescript* o *Flow*, que agregan tipado estático. El tipado estático sirve para auto documentar el código, haciendo la colaboración más fácil, y permitiendo detectar errores de tipo de manera temprana, evitando largas y complicadas sesiones de debugging, y sets extra de pruebas [82].

Por estas razones es que se decide utilizar uno de los *super-sets* de *Javascript* para desarrollar en *React*, más específicamente *Typescript*, ya que el equipo ya lo conocía.

Redux

Todas las aplicaciones web que siguen la arquitectura SPA deben mantener el estado de la navegación del usuario en la web. Parte de este estado suele ser local a la porción de

la web que el usuario está navegando, por ejemplo, los datos de las entradas de un *form*. La otra parte del estado debe ser persistido de manera transversal a todos los componentes de *React*, para poder ser accedida por cada componente sin tener que realizar peticiones al *backend,* como información del usuario logueado en sí. Esta persistencia global no debe generar renderizaciones innecesarias, y debe permitir ser leída y modificada sin agregar dependencias innecesarias a nivel de código.

Redux es una librería que provee al proyecto de React de un contenedor para el estado global, y cumple con todas las características previamente mencionadas. Esta utiliza el patrón *publish subscribe* para publicar y distribuir la información manteniendo las dependencias a nivel de código al mínimo, y no genera nuevas renderizaciones en el algoritmo de React a menos que se publique o renueve la información del estado.

Material Design

Como se menciona en la descripción de la arquitectura en la sección de usabilidad, se utilizó la librería *Material-UI* en *React*, que sigue con el lenguaje de diseño conocido como *Material Design*. Esto le facilitó al equipo el desarrollo de componentes, ya que no fue necesario implementar el aspecto visual de cada uno, asegurando la consistencia visual y en la experiencia de usuario.

6.3.3 Infraestructura

Una de las principales decisiones que el equipo tomó fue decidir qué infraestructura construir para el alojamiento del sistema. Si bien los integrantes del equipo habían realizado el curso de “Arquitectura de Software En La Practica”, el nivel de exigencia necesario para albergar un proyecto de tal tamaño sobrepasaba cualquier conocimiento académico previo.

PaaS vs IaaS

Se dedicaron días para investigar diferentes alternativas para la correcta elección de las tecnologías que conformasen a la infraestructura del sistema. Luego de analizar factores como el costo, complejidad de uso y control de la plataforma el equipo termino decidiéndose por entre dos candidatos: *Heroku* y *Amazon Web Services*. Si bien ambos presentaban una inmensidad de beneficios a contemplar, la principal diferencia en ambos era conceptual; *Heroku* es catalogada como un *Platform as a Service* (PaaS por sus siglas en inglés), y *AWS* como un *Infraestracture as a Service* (IaaS por sus siglas en inglés). Luego del análisis realizado que se encuentra en el Anexo 13.13, el equipo terminó optando por *Amazon Web Services*.

Amazon Web Services

Amazon Web Servies es un *IaaS* y fue elegido ya que sus costos son muy convenientes y al mismo tiempo brinda una escalabilidad única.

Elegir un *IaaS* tiene muchos beneficios, ya que el equipo es responsable de toda la infraestructura. En primer lugar, permite acceder a un sistema con costos económicos, más bajos en comparación con otros de los modelos propuestos por *Cloud Computing* [83].

Por otro lado, tener todo el sistema dentro de *Amazon Web Services* genera una gran ventaja, ya que todos los servicios de *AWS* están hechos para trabajar correctamente en conjunto.

Otra gran ventaja de este modelo es la flexibilidad y escalabilidad que permite. Bajo este modelo es muy sencillo escalar un sistema y se puesden elegir entre distintos servidores con distinta capacidad de cómputo para poder elegir el que se adapte más a la solución. Además del escalado vertical tradicional, el cual es muy sencillo de manipular en *AWS*, entre los servicios de *AWS* encontramos los *Elastic Load Balancers* los cuales nos

permiten balancear la carga de la plataforma entre varias instancias de servidores de manera simple haciendo que el escalado horizontal no tenga límites.

Vale la pena mencionar que los servicios *IaaS* suelen tener una disponibilidad bastante superior a las provistas por otros modelos de *Cloud Computing*. En el caso de *AWS*, la disponibilidad es mayor al 99.99 %.

A continuación, se describen los servicios *cloud* utilizados:

Almacenamiento

El almacenamiento del sistema requirió de la utilización de dos servicios *cloud* de *Amazon*, según los requerimientos especificados.

En primer lugar, las bases de datos relacionales son implementaron utilizando el servicio de *Amazon RDS* (Relational Database System) [84]. Este servicio permite la escalabilidad de forma sencilla, ofrece alta disponibilidad y buena performance y seguridad. También permite realizar respaldos de la base de datos de forma automática y programada, y en caso de necesitar uno en un momento especifico, se lo puede hacer manualmente mediante *snapshots.* Esto significa una gran ventaja en caso de pérdidas de información. Por cada base de datos necesaria se creó una instancia *MySQL*.

La elección de esta se debe a su facilidad de integración con *Java Spring y* experiencia del equipo. Además, brinda todos los beneficios de una base de datos relacional, tales como la integridad y facilidad de los datos, que son aspectos fundamentales para garantizar la seguridad de esta [85].

Por otro lado, la base de datos *NoSQL* se utilizó mediante *Amazon DynamoDB.* Esta es una base de datos no relacional y clave-valor completamente administrada que ofrece desempeños rápidos, alta disponibilidad, durabilidad e integridad de los datos y una escalabilidad optima. A su vez, también ofrece facilidad de integración para con *Java Spring* mediante la *SDK* provista por el servicio en tal lenguaje [73].

Se utilizó el servicio de *S3* (*Amazon Simple Storage Services*) para el almacenamiento de imágenes [86]. Presenta un excelente rendimiento y permite realizar copias de seguridad de los archivos en cualquier momento. Algunas ventajas que presenta son su facilidad de eso, escalabilidad y seguridad.

Alojamiento

Para el alojamiento de los servidores del *backend* se utilizó el servicio de *EC2* (*Elastic Cloud Computing*), mediante la creación de instancias con características físicas y de rendimientos apropiadas a las necesidades del proyecto [87]. Se crearon cuatro servidores para alojar a los componentes del *backend* y otro para la utilización de un repositorio privado de *Nexus* [88].

Para los servidores del *backend* se crearon instancias con el sistema operativo *Linux Amazon 2* y dentro de estas se instaló la versión de *Java Correto11*. Las instancias se encuentran alojadas en la región de Us-East-1/Virginia. Si bien *Amazon* ofrece una instancia *Free Tier,* es decir gratuita, el equipo se vio en la necesidad de aumentar su capacidad y pasarse a un plan pago para cumplir con los requerimientos de un producto que pretendía estar en producción.

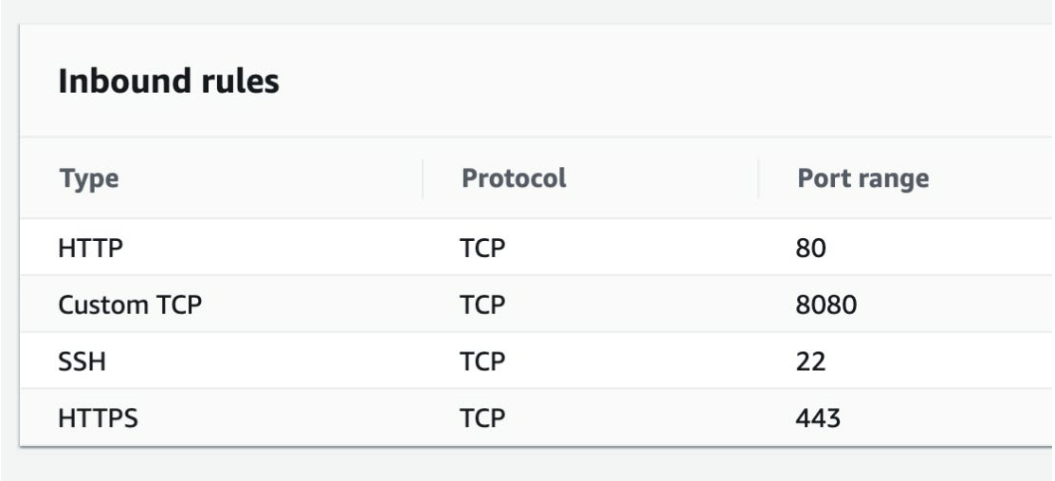
Una vez creados y configurados las instancias, el equipo implementó varias tácticas para mejorar la disponibilidad, performance y seguridad de su aplicación:

- Seguridad

Las instancias del *backend* se crearon dentro de una red virtual privada (VPC por sus siglas en inglés), lo que significa que sus accesos están restringidos por el alcance de red configurado. Además, estas fueron configuradas con reglas a nivel red (ACL) y a nivel de aplicación (*security groups*) que permitieron restringir o habilitar los accesos de entrada y salida hacia las instancias y exterior (internet).

De esta manera se aplicaron las tácticas de resistir ataques, así como limitar exposición y acceso. Por ejemplo, las instancias del *backend* fueron habilitados para servir a las peticiones que lleguen por el puerto 8080 mediante protocolo *https.*

Además, las instancias pueden ser únicamente accedidas mediante *SSH* y utilizando un par de llaves pública y privada, utilizando la táctica de encriptar. Los accesos a estas fueron restringidos a un único integrante del equipo para mitigar el riesgo a una filtración de seguridad, además de crear un usuario para el despliegue continuo con accesos de lectura limitados.



Inbound rules

Type	Protocol	Port range
HTTP	TCP	80
Custom TCP	TCP	8080
SSH	TCP	22
HTTPS	TCP	443

Ilustración 44 - Reglas de seguridad en *AWS*

Por otro lado, se utilizó el servicio de *Amazon Certificate Manager* (*ACM* por sus siglas en inglés) para la creación de certificados *SSL* y permitir una comunicación segura entre los componentes del *backend* y con el *frontend* [89].

Name	Domain name	Additional names	Status	Type
-	api.conecta361.tk	-	Issued	Amazon Issued
-	panel.conecta361.tk	-	Issued	Amazon Issued
-	app.conecta361.tk	-	Issued	Amazon Issued
-	conecta361.tk	www.conecta361.tk	Issued	Amazon Issued

Ilustración 45 - Configuración de dominios en *AWS*

- Disponibilidad

Los servicios utilizados para cumplir con una buena disponibilidad a nivel de infraestructura fueron varios. Por un lado, las instancias creadas fueron reagrupadas en determinadas *subnets* y módulos lógicos para luego poder ser utilizadas como un conjunto indiferente (escalabilidad horizontal). De esta manera, al configurar luego el balanceador de carga mediante el *Elastic Load Balancer (ELB* por sus siglas en inglés*)*, se balanceó la carga de tráfico del internet y distribuyo en forma uniforme hacia las instancias. De esta manera se utilizaron las tácticas de disponibilidad de recuperación de fallas. Además, el balanceador de carga sirvió para asegurar la disponibilidad en los momentos de despliegue de las aplicaciones. Utilizando la táctica *ping/echo* el *load balancer* realiza peticiones de verificación de salud de la aplicación cada cinco segundos. En caso de que alguna de las instancias no esté saludable, las peticiones siguientes no serán tomadas en cuenta por el balanceador para el ruteo de tráfico hacia esa instancia.

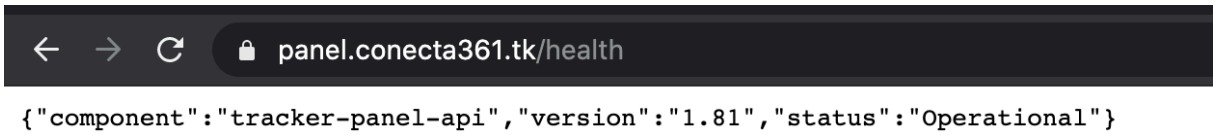
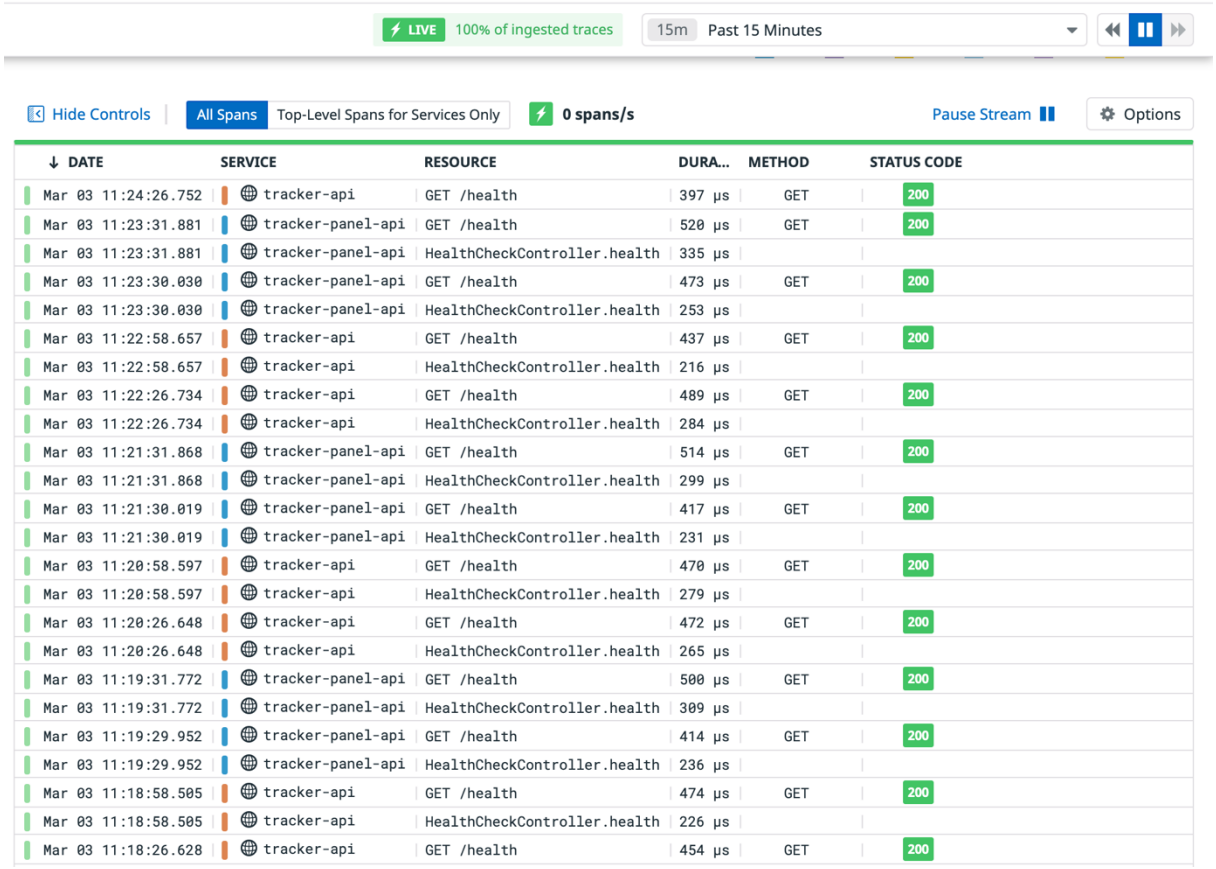


Ilustración 46 - *Health check*



DATE	SERVICE	RESOURCE	DURA...	METHOD	STATUS CODE
Mar 03 11:24:26.752	tracker-api	GET /health	397 µs	GET	200
Mar 03 11:23:31.881	tracker-panel-api	GET /health	520 µs	GET	200
Mar 03 11:23:31.881	tracker-panel-api	HealthCheckController.health	335 µs		
Mar 03 11:23:30.030	tracker-panel-api	GET /health	473 µs	GET	200
Mar 03 11:23:30.030	tracker-panel-api	HealthCheckController.health	253 µs		
Mar 03 11:22:58.657	tracker-api	GET /health	437 µs	GET	200
Mar 03 11:22:58.657	tracker-api	HealthCheckController.health	216 µs		
Mar 03 11:22:26.734	tracker-api	GET /health	489 µs	GET	200
Mar 03 11:22:26.734	tracker-api	HealthCheckController.health	284 µs		
Mar 03 11:21:31.868	tracker-panel-api	GET /health	514 µs	GET	200
Mar 03 11:21:31.868	tracker-panel-api	HealthCheckController.health	299 µs		
Mar 03 11:21:30.019	tracker-panel-api	GET /health	417 µs	GET	200
Mar 03 11:21:30.019	tracker-panel-api	HealthCheckController.health	231 µs		
Mar 03 11:20:58.597	tracker-api	GET /health	470 µs	GET	200
Mar 03 11:20:58.597	tracker-api	HealthCheckController.health	279 µs		
Mar 03 11:20:26.648	tracker-api	GET /health	472 µs	GET	200
Mar 03 11:20:26.648	tracker-api	HealthCheckController.health	265 µs		
Mar 03 11:19:31.772	tracker-panel-api	GET /health	500 µs	GET	200
Mar 03 11:19:31.772	tracker-panel-api	HealthCheckController.health	309 µs		
Mar 03 11:19:29.952	tracker-panel-api	GET /health	414 µs	GET	200
Mar 03 11:19:29.952	tracker-panel-api	HealthCheckController.health	236 µs		
Mar 03 11:18:58.505	tracker-api	GET /health	474 µs	GET	200
Mar 03 11:18:58.505	tracker-api	HealthCheckController.health	226 µs		
Mar 03 11:18:26.628	tracker-api	GET /health	454 µs	GET	200

Ilustración 47 - *Health check* logueado en *DataDog*

Dominios

"Dado que existen muchas direcciones IP, los nombres de dominio se crean para formar una identidad única para cada sitio web. Sin un nombre de dominio, su dirección web se traducirá en números de IP. Imagínese la terrible experiencia de recordar una serie de números" [90].

El equipo se vio en búsqueda de registrar un dominio para la aplicación web del sistema, así como también para la API de acceso público. Por este motivo, en primer lugar, se recurrió al gestor de dominios *Freenom*[7], que ofrece el registro de dominios sin costo. Una vez registrados, se utilizó el servicio de *Route53* para la creación de una *hosted zone* y asociar el dominio al balanceador de carga de la aplicación. Luego se crearon los certificados de seguridad correspondientes para poder acceder a la aplicación de forma segura.

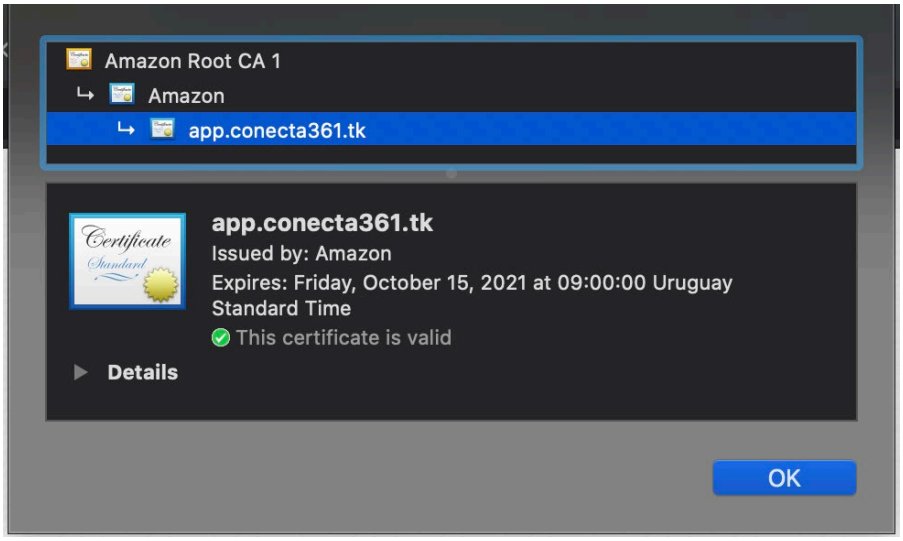


Ilustración 48 - Certificado SSL

[7] https://www.freenom.com/en/index.html?lang=en

7. Gestión de proyecto

En el presente capítulo se describen los procesos de gestión utilizados a lo largo de las diferentes fases del proyecto. Se explicará cómo se llevó a cabo la misma y las adaptaciones a las metodologías utilizadas según su marco. A su vez se describen y analizan las métricas obtenidas y la gestión de riesgos realizada.

7.1 Fases e hitos del proyecto

En la siguiente ilustración se pueden visualizar las distintas fases e hitos que atravesaron el proyecto.

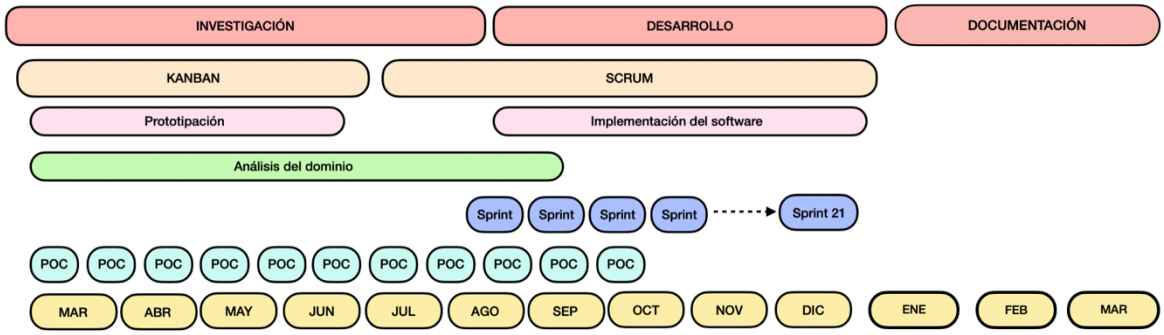


Ilustración 49 - Fases e hitos del proyecto

7.2 Fase de investigación

Como se mencionó en la sección de Fases del proyecto, el equipo dedicó una fase a investigar sobre el dominio del problema, tecnologías y las herramientas a utilizar. Este permitió adquirir un conocimiento más amplio acerca del negocio. En la siguiente sección se justificará la metodología utilizada *Kanban*.

7.2.1 Implementación de *Kanban*

Se aplicó la metodología ágil *Kanban* debido a que se trataba de un dominio desconocido por los integrantes del equipo como lo es el área del marketing digital. Esta presentaba gran incertidumbre y requerimientos cambiantes. La misma permitió controlar la cantidad de trabajo realizado y llevar un flujo organizado y continúo visualizando el estado de cada tarea dentro del tablero. Una de las grandes ventajas que presenta *Kanban* es la flexibilidad en los tiempos y priorización de tareas [49].

Tablero de Kanban

El tablero de *Kanban* fue utilizado a través de la herramienta *Jira* que permitió la visualización de las tareas de forma *online* mediante una interfaz gratuita, amigable y colaborativa [91].

Dentro del tablero se establecieron tres columnas que representan a los diferentes estados de las tareas:

- *To do*: Tareas que todavía no fueron comenzadas y ordenadas por prioridad.
- *In progress*: Tareas en proceso.
- *Done*: Tareas que fueron completadas.

En la siguiente imagen se permite ver las columnas mencionadas:

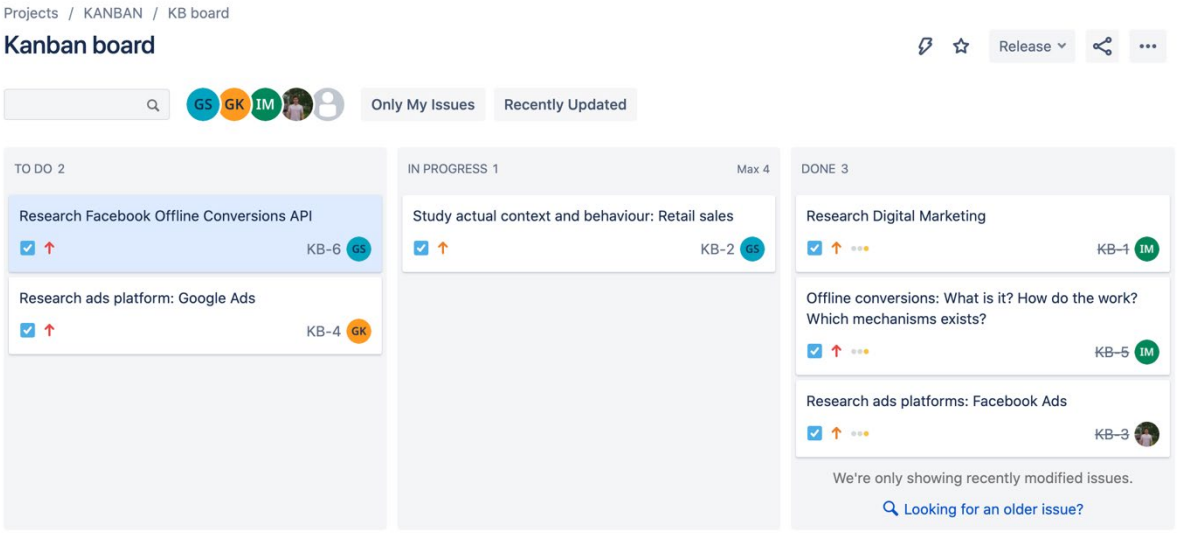


Ilustración 50 - *Kanban board*

Las primeras tareas para realizar surgieron luego de una lluvia de ideas entre el equipo debido a la incertidumbre del dominio del problema y a la inexperiencia de este. Cada integrante tenía la flexibilidad de ir agregando nuevas tareas a la columna *"To do"* a medida que se desarrollaba la investigación. Sin embargo, se estableció que el estado *"In progress"* podía tener un límite máximo de trabajo (*WIP* por sus siglas en inglés) de cuatro tareas en simultaneo con el objetivo de evitar cuellos de botella lo cual bloquearía el avance del proyecto. Un integrante podía tomar una nueva tarea solo cuando finalizaba con la tarea anterior.

Tareas que eran consideradas muy relevantes o que requerían de mucho tiempo de investigación, fueron asignadas a más de un integrante con el fin de resolverlas en un menor tiempo. En la reunión de los miércoles con el tutor, se realizaba un resumen de lo investigado en la semana y se priorizaban las tareas.

7.2.2 Distribución de esfuerzo

En el tablero de *Kanban* se asignaron categorías a las diferentes tareas con el objetivo de medir la distribución de esfuerzo realizadas a lo largo de la fase de investigación.

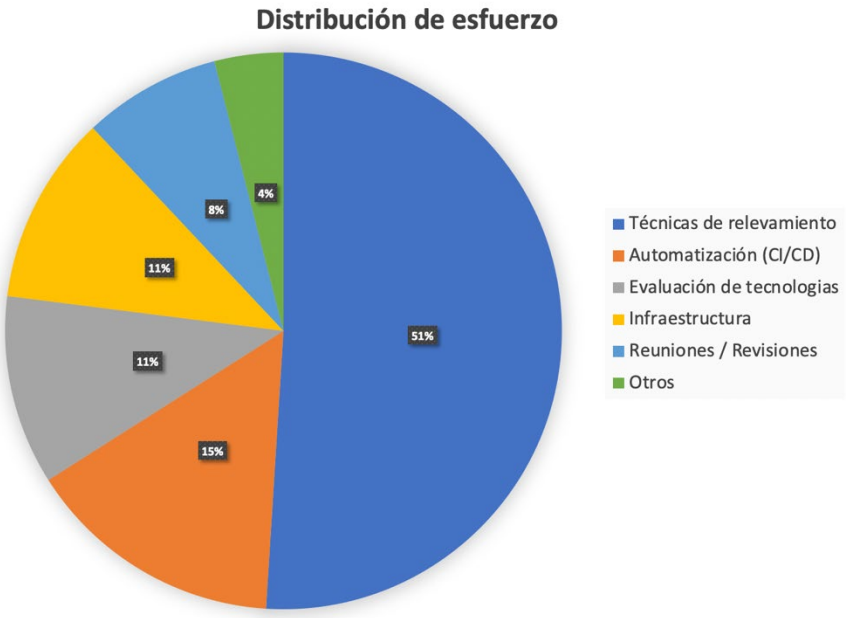


Ilustración 51 - Distribución de esfuerzo por categoría en fase de investigación

En la anterior grafica se puede visualizar el porcentaje aproximado de la distribución de esfuerzo por categoría, ya que, si bien hubo un control en estas, no todas las tareas quedaron categorizadas.

De esta grafica se desprenden las siguientes observaciones:

En primer lugar, se puede ver como la técnica de relevamiento de requerimientos demandaron más de la mitad del esfuerzo en la fase de investigación. Esto se debe principalmente a las características del proyecto y desconocimiento del equipo en la temática, que llevaron a los integrantes a buscar todas las formas posibles de adquisición de conocimiento previo a la etapa de desarrollo.

A su vez, también se puede observar que se realizó una investigación en la automatización de los procesos, así como en evaluación de tecnologías e infraestructura.

La metodología ágil utilizada por parte del equipo requiero de entregar software frecuente al cliente, por lo que tener tecnologías compatibles para ellos fue fundamental.

7.3 Fase de desarrollo

La segunda fase del proyecto fue el desarrollo de la solución. Si bien el equipo siguió con un enfoque ágil, se optó por cambiar la metodología y pasar a utilizar *Scrum* por las ventajas que presenta sobre *Kanban*. Algunas de ellas son el poder definir roles respecto a este dentro del equipo, establecer iteraciones de tiempos fijos (*sprints*), tener flexibilidad en agregar valor a los usuarios pudiendo interactuar frecuentemente de manera de priorizar y validar de forma constante. También permite recolectar métricas como la velocidad del equipo y esfuerzo de las tareas. A su vez, es flexible en el sentido que permite agregar nuevas funcionalidades y realizar modificaciones sin que tengan gran impacto. Cabe destacar que todos los miembros del equipo ya contaban con experiencia previa trabajando con esta metodología por lo que la decisión fue unánime.

7.3.1 Implementación de *Scrum*

A continuación, se detalla la implementación de *Scrum* [92]. Se definirán los roles, artefactos utilizados y la adaptación de los eventos.

Roles de Scrum

Cada integrante tuvo su rol y se asignó de la siguiente manera:

- *Equipo de desarrollo*

Este rol fue asumido por los cuatro integrantes del equipo. Todos colaboraron en la construcción del producto.

- Scrum Master

Este rol fue asumido por Mauricio Pisabarro. Tuvo la responsabilidad de actuar como protección entre el equipo y cualquier influencia que pueda distraer. Se ocupó de que el proceso *Scrum* se cumpla de manera correcta.

- *Product Owner*

Se designó como *Product Own*er a Marcelo Wilkorwsky, CEO de *Conecta361* por ser el experto en el dominio. Encargado de marcar los objetivos. Con él fue que se validó los requerimientos y priorización de las tareas del backlog.

Artefactos

- *Product Backlog*

Lista dinámica de todas las tareas e historias de usuario que conforman los requerimientos del producto y fue priorizada por el *product owner*. Los cambios se debieron por nuevos requerimientos que fueron surgiendo. La lista se puede ver en el Anexo 13.11.

- *Sprint Backlog*

Lista de trabajo para implementar las historias de usuario que el equipo elaboró en la *Sprint Planning* para completar los objetivos seleccionados de la próxima iteración. Las mismas fueron seleccionadas del *product backlog*.

- *Producto Incrementado*

Al final de cada iteración se obtuvo un producto incrementado.

- *Tablero Scrum*

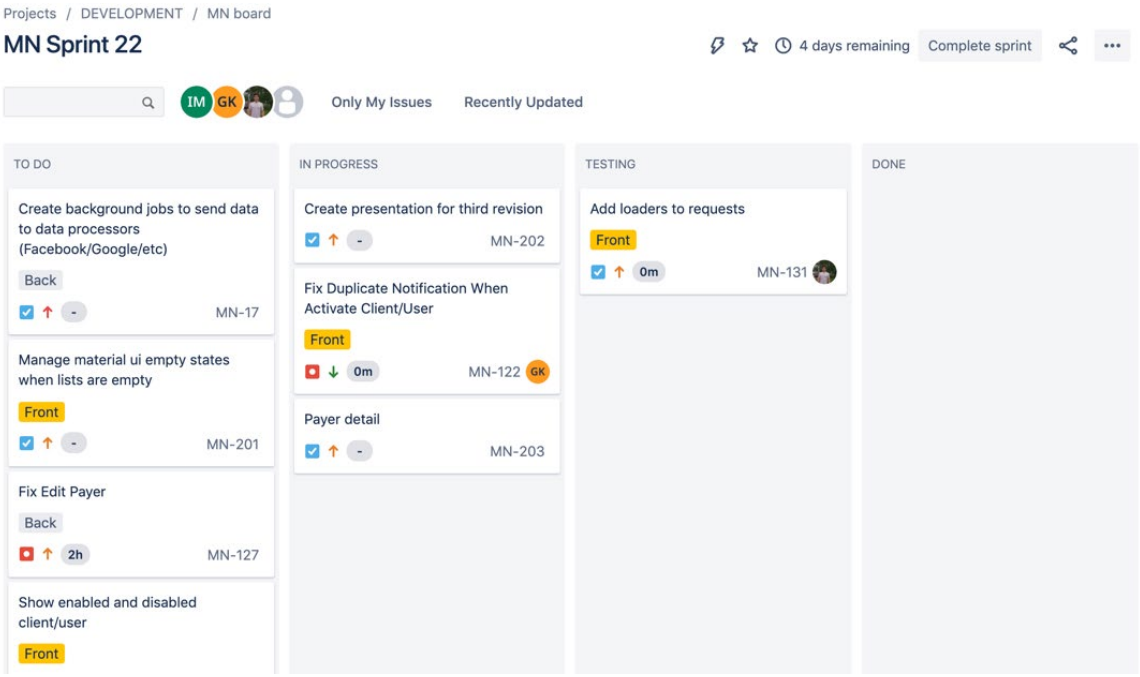


Ilustración 52 - Tablero *Scrum*

Se utilizó la herramienta *Jira* para hacer el seguimiento de las tareas. En la ilustración se puede visualizar las diferentes historias de usuario en el *sprint backlog*. Se observa también los diferentes estados por los cuales transita una tarea: *"To do", "In progress", "Testing", "Done"*. Cada historia de usuario contiene quien es el responsable, estimación en *story points* y si pertenece al *frontend* o al *backend*. Dentro de cada una se encuentra la lista de criterios a cumplir para que la misma pueda pasar a *Done*.

Los criterios de aceptación de cada historia se definieron en la *planning meeting*. Todos los domingos se llevó a cabo la reunión para dar por finalizado el sprint. Todas las tareas que no quedaban en "*Done*" se pasaban al *product backlog* y se volvían a estimar y priorizar para ver si se incluían en la próxima iteración.

Flujo de trabajo

"Sprint son eventos de duración fija de un mes o menos para crear coherencia. Un nuevo sprint comienza inmediatamente después de la conclusión del sprint anterior" [93]. Se

optó por hacer iteraciones de una semana para poder corregir de forma temprana ante requerimientos cambiantes. También sirvió para medir de forma rápida la velocidad del equipo y poder estimar de una mejor manera las tareas del *sprint backlog*.

Se tuvieron algunas de las consideraciones durante los sprints que son mencionadas a continuación.

En primer lugar, al inicio de cada iteración se realizó la *Sprint Planning* con el objetivo de seleccionar las historias de usuario para el desarrollo teniendo presente la velocidad del equipo.

Además, en el último día de cada sprint se realizó la *sprint review* para validar el producto con el cliente y *sprint retrospective* con el objetivo de evaluar el trabajo del *sprint* y proponer mejoras para la siguiente iteración.

A su vez, se definieron ciertas reglas para cumplir durante su ejecución y estas fueron:

- No agregar *user stories* una vez iniciado el *sprint*.
- Una tarea solo puede pasar a estado *"Done"* si y solo si cumplió con los criterios de aceptación definidos.
- Registrar cuanto tiempo llevo completar la historia de usuario. De esta manera podremos saber el esfuerzo al finalizar el *sprint*.
- Llevar en tiempo real el estado de cada tarea. Esto permitió saber cuándo una *user story* está en estado *"To do"*, cuando esta "*In Progress"*, cuando está en "*Testing*" y cuando quedó completada.
- Cada tarea debe tener su prioridad asignada. Esta se fijó para establecer de forma clara el objetivo y prioridad de cada una de manera de saber su incidencia en el proyecto.

Una vez definido el flujo el flujo de trabajo, se procede a explicar los respectivos eventos:

- *Scrum Planning Meeting*

Todos los domingos se realizó la *planning meeting* con el objetivo de planificar el siguiente *sprint*. Los actores que formaron parte de la misma fueron el *scrum master* y el equipo de desarrollo [93]. Durante esta reunión de planificación, se tenía en cuenta lo conversado con el *product owner* quien priorizaba las tareas del backlog. Luego, el equipo estimaba cada historia de usuario utilizando la técnica de *planning poker* ya que es sencilla y eficaz [94]. Se tomó en cuenta la velocidad del equipo y el esfuerzo a dedicar de cada integrante para saber cuántas historias de usuario poner en el sprint backlog. La dedicación del equipo fue menor en épocas de parciales, entrega de obligatorios o compromisos laborales.

- *Daily Meeting*

Este evento consiste en una reunión entre los integrantes del equipo para contar los avances del día anterior y ver si necesitan algo del día actual [93]. No se cumplió de forma estricta con este evento debido a que los integrantes del equipo manejaban diferentes horarios de disponibilidad.

- *Sprint Review*

Este evento consiste en el incremento realizado en el sprint anterior. Al finalizar cada iteración, el equipo realizo validaciones con el Product Owner a través de demos para verificar que el desarrollo cumpla con los criterios de aceptación definidos. En la misma, obtuvieron *feedback* de parte del cliente con mejoras o cambios que deseaba hacer [93]. En el siguiente Anexo 13.14 se puede encontrar evidencia del *feedback* recolectado de algunas de las reuniones con el cliente.

- *Sprint Retrospective*

En este evento, el equipo después de cada *sprint* realizaba una autoevaluación del *sprint* pasado. Todos los integrantes del equipo dejaron su reflexión de lo que se hizo bien, lo que se hizo mal y que se podía mejorar para la próxima iteración [93]. Que la crítica sea constructiva les ayudó a aumentar la productividad a lo largo de los diferentes *sprints*.

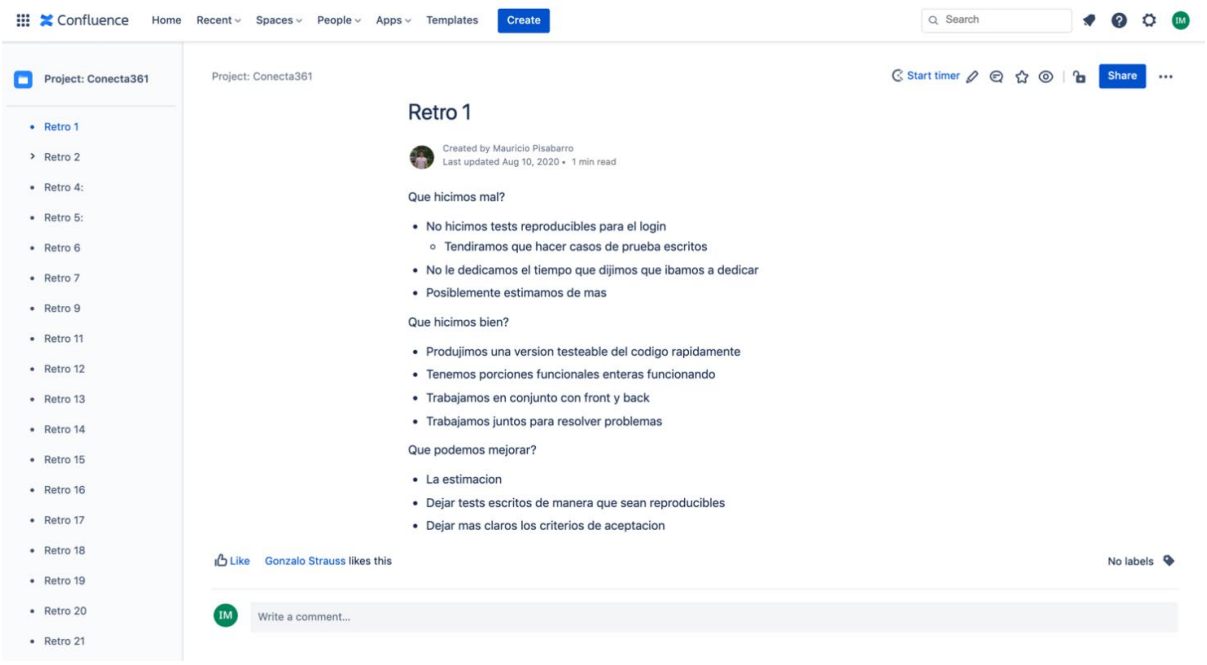


Ilustración 53 - *Sprint retrospective*

7.3.2 Release plan

"Un release plan o plan de proyecto es un conjunto de historias de usuario agrupadas por release o versiones del producto que se ponen a disposición de los usuarios incrementando el valor para estos respecto de la anterior" [95].

El equipo consideró útil realizar el mismo ya que ayuda al mismo a enfocarse en los objetivos propuestos. Permitió al equipo y a *Conecta361* tener una noción de las fechas estimadas de cada *milestone* y definir el alcance del producto a desarrollar. Sirvió como guía para saber en todo momento en que etapa se está del proyecto y visualizar el progreso.

El plan de proyecto se armó teniendo en cuanta las funcionalidades a implementar luego de que el *product owner* priorizó las mismas. Cabe destacar que esta fue dinámica ya que fue cambiando a lo largo del proyecto y a medida que se iba ajustando la velocidad del equipo.

Los sprints fueron agrupados en cuatro *milestones* según los objetivos de este. Estos fueron:

Numero	*Milestone*
1	Integración y despliegue continuo
2	Panel administrativo funcional
3	Procesamiento de conversiones *offline*
4	Integración completa de las conversiones con datos procesados

Tabla 4 - Milestones

En la siguiente tabla se puede observar el inicio y fin de cada *sprint*, así como también el cumplimiento del mismo.

Milestone	# Sprint	Duración	Status
Milestone 1	1	03/08 - 10/08	Completado
Milestone 1	2	10/08 - 17/08	Completado
Milestone 1	3	17/08 - 24/08	Completado
Milestone 1	4	24/08 - 31/08	Completado
Milestone 1	5	31/08 - 07/09	Completado
Milestone 2	6	07/09 - 14/09	Completado
Milestone 2	7	14/09 - 21/09	Completado
Milestone 2	8	21/09 - 28/09	Completado
Milestone 2	9	28/09 - 05/10	Completado
Milestone 3	10	05/10 - 12/10	Completado
Milestone 3	11	12/10 - 19/10	Completado
Milestone 3	12	19/10 - 26/10	Completado
Milestone 3	13	26/10 - 02/11	Completado
Milestone 3	14	02/11 - 09/11	Completado
Milestone 3	15	09/11 - 16/11	Completado
Milestone 3	16	16/11 - 23/11	Completado
Milestone 4	17	23/11 - 30/11	Completado
Milestone 4	18	30/11 - 07/12	Completado
Milestone 4	19	07/12 - 14/12	Completado
Milestone 4	20	14/12 - 21/12	Completado
Milestone 4	21	21/12 - 28/12	Completado

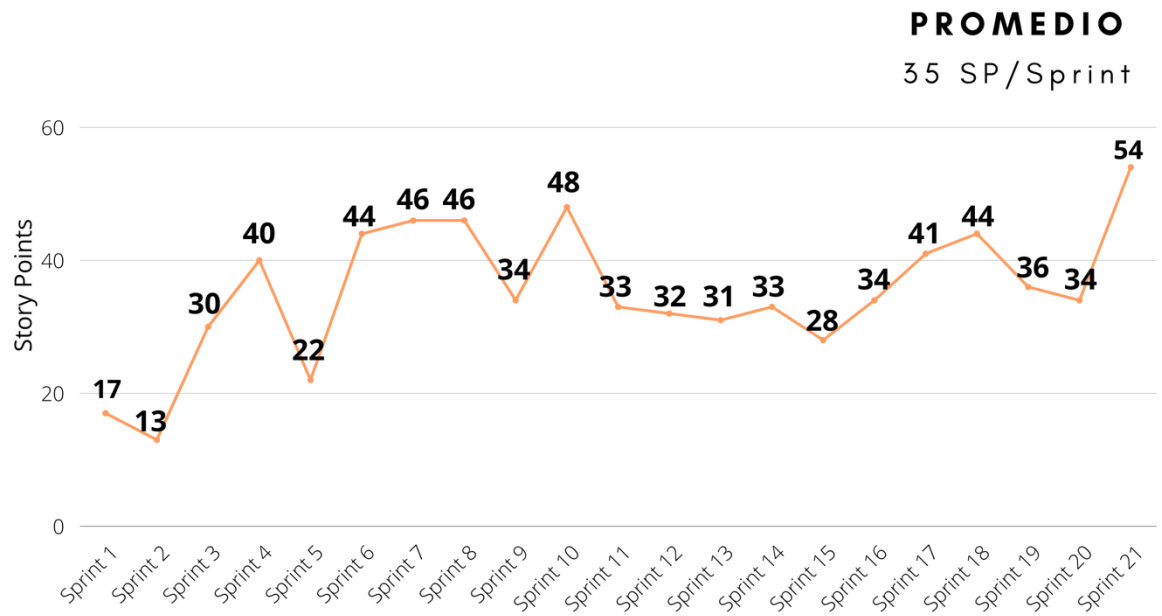
Tabla 5 - Sprints por milestone

En el Anexo 13.15 se pueden visualizar los principales hitos logrados y el cumplimiento de cada *milestone*.

7.3.3 Métricas de gestión

A continuación, se analizaron distintas métricas de gestión que permitieron al equipo evaluar el proceso llevado a cabo para cumplir con los objetivos del proyecto. Las mismas permitieron tener un mayor control sobre cada fase del proyecto y ver el progreso del mismo. A partir de los resultados obtenidos, se sacaron conclusiones al respecto.

Velocidad del equipo



Gráfica 6 - Velocidad del equipo

La velocidad por *sprint* es una métrica que permite visualizar la cantidad de *story points* que el equipo completaba luego de terminar un *sprint*. Para la evaluación de esta métrica de manera correcta, se deben tener en cuenta todos los *sprints*, y las condiciones particulares con las que se trabajó en cada uno. A continuación, se presenta una gráfica con la velocidad del equipo durante el desarrollo del proyecto.

Como se puede ver en la gráfica anterior, la velocidad del equipo fue variando a lo largo de todo el proyecto, siempre por distintos motivos. El principal motivo es que el equipo no contaba con la posibilidad de dedicarle el tiempo exclusivamente al proyecto, sino que también debía dedicarles a los trabajos particulares de cada uno de los integrantes, así como también a otras materias de la facultad.

Al comienzo del proyecto se pueden ver *sprints* de velocidad muy bajas, debido a que el equipo tuvo que adaptarse a la metodología de trabajo del proyecto, y no estaba acostumbrado a trabajar en un proyecto tan grande y teniendo que gestionarse solos. Otro factor fue que, al principio del proyecto, el equipo no tenía muy claro cómo realizar

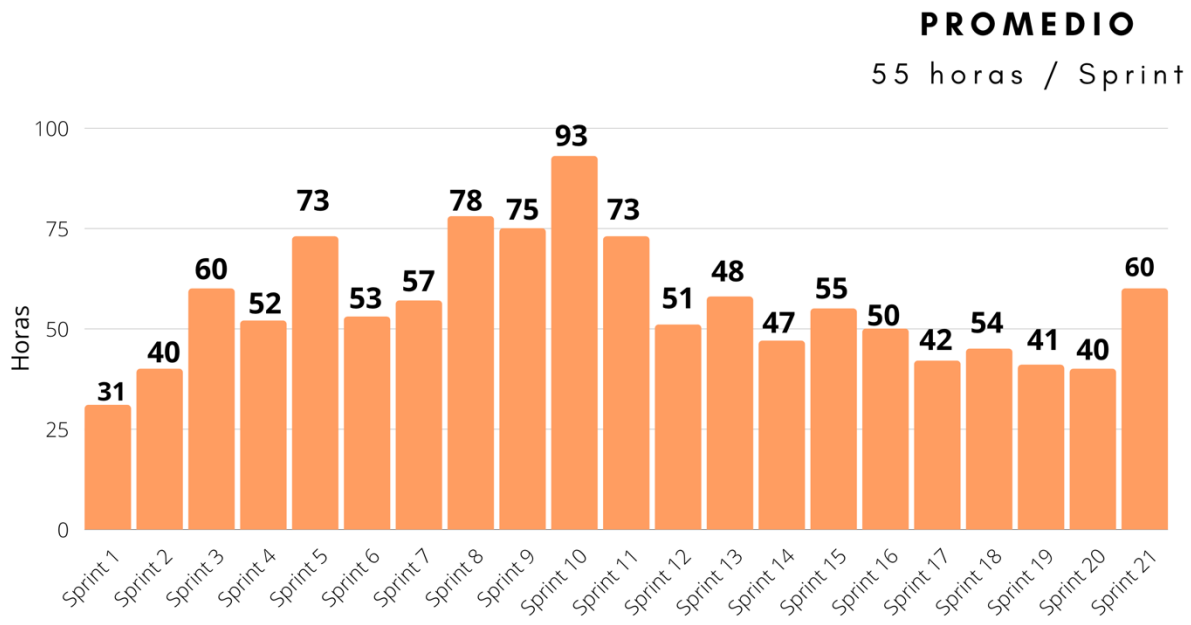
las estimaciones de las tareas, pero con el correr de las semanas, se fue adaptando a trabajar de manera conjunta y estimando mejor las tareas.

A partir del sprint 6 ya se puede ver que el equipo toma un ritmo constante con respecto a los *story points*, logrando llegar a un promedio de 35 *story points* por sprint.

Se pueden observar casos particulares, como lo son el sprint 5 o el sprint 15, donde el equipo decidió bajar la velocidad y definirse tareas de menor importancia, debido a distintas situaciones ajenas al proyecto, ya sea por entregas de obligatorios, parciales, u obligaciones laborales que no permitieron que el equipo le dedique al proyecto de la misma manera que el resto de los *sprints*.

Esfuerzo en desarrollo

El esfuerzo es otra métrica que permite demostrar la dedicación del equipo con el proyecto. A diferencia de los *story points* por *sprint*, los cuales pueden ser una métrica más variable, las horas dedicadas por *sprint* son algo fijo y conocido, lo cual demuestra cuanto trabajó el equipo en cada sprint, ignorando el peso de las tareas realizadas.



Gráfica 7 - Esfuerzo en desarrollo

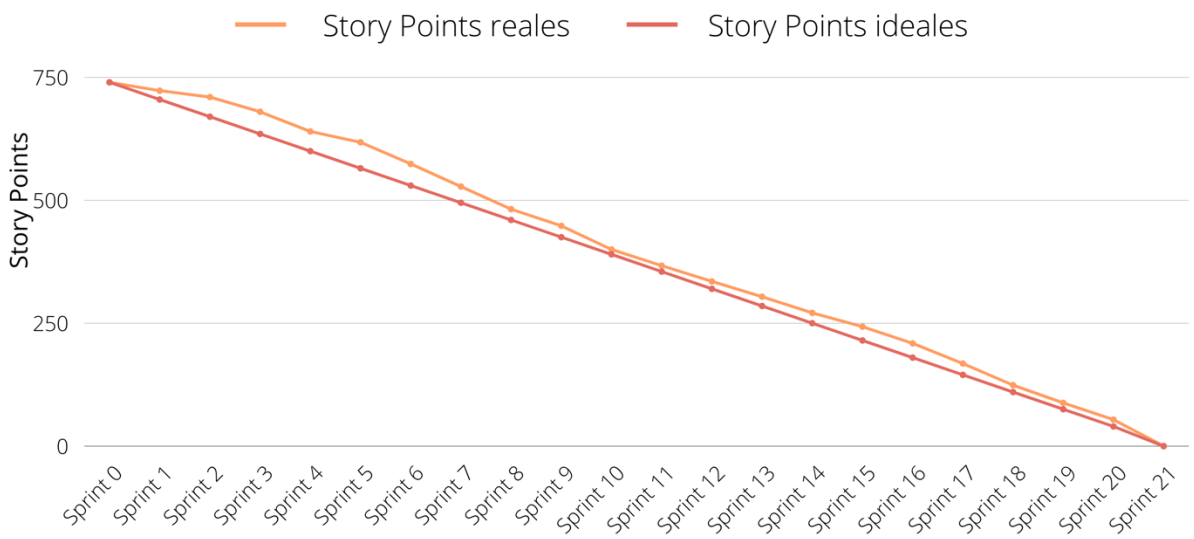
Como se puede visualizar en la gráfica anterior, la misma se alinea bastante con la velocidad del equipo. Al igual que esta, también tiene un incremento luego de los primeros *sprints* del proyecto, por el hecho principal de que el equipo se fue acostumbrando a la metodología de trabajo autogestionado, y fue agarrando tracción con las distintas tecnologías, lo cual permitió que se le pueda dedicar más.

Así como en la gráfica anterior, también se ven reflejados en esta los *sprints* en los que el equipo no le pudo dedicar tanto tiempo, aunque no de una manera tan diferenciada como con la de los *story points*. Se puede ver claramente que en el sprint 10, el equipo le dedicó casi el doble del promedio semanal, por motivos de organización y preparación de otros aspectos del proyecto.

Más sobre el final, se puede apreciar que el equipo ya estaba bien entendido con las tecnologías y el desarrollo del proyecto, por lo que lograba completar una mayor cantidad de *story points*, dedicándole un promedio más bajo de horas por semana.

Burndown chart

La gráfica de *burndown chart* fue de gran utilidad para permitir al equipo realizar un seguimiento de la manera en la que iba avanzando el proyecto, además de reflejar el trabajo restante del mismo. En la misma se puede observar la cantidad de *story points* restantes luego de cada sprint.



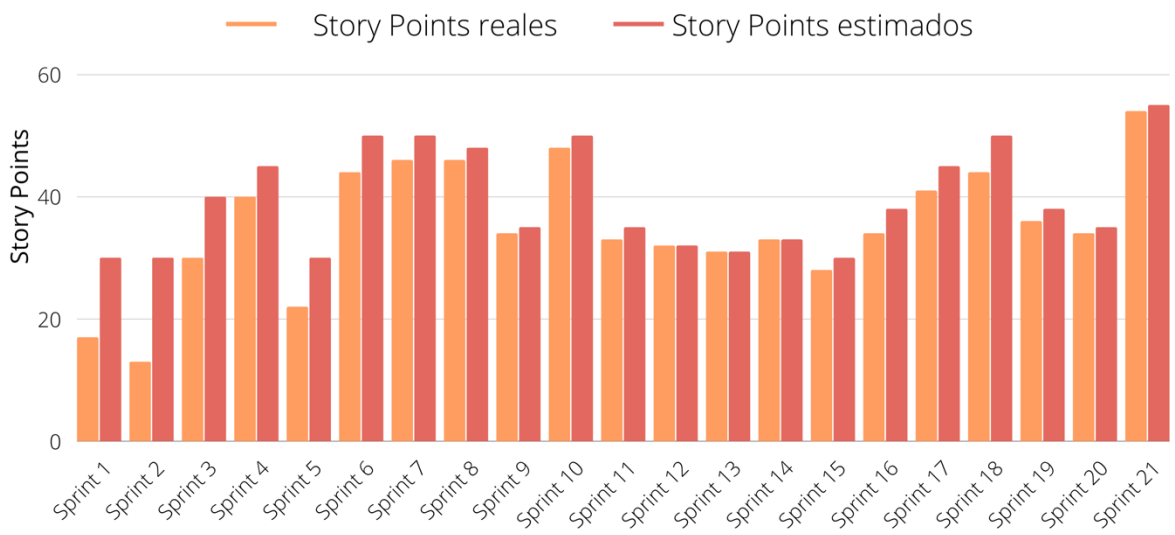
Gráfica 8 - Burndown chart

Se puede observar una diferencia entre los story points ideales y reales en las primeras iteraciones del proyecto, lo cual se debe a la inexperiencia del equipo con algunas tecnologías, así como la dedicación del equipo al comienzo. A medida que se fue avanzando en el proyecto, esta diferencia fue disminuyendo, lo cual significa que el equipo estaba trabajando a un buen ritmo y con posibilidades de lograr terminar a tiempo el alcance del proyecto.

Entre los sprints 15 y 17, se vuelve a ver una desviación entre los *story points* reales y los ideales, lo cual se atribuye a las distintas obligaciones académicas de los integrantes, los cuales le tuvieron que dedicar menos tiempo al proyecto.

En los últimos 4 *sprints*, se logra apreciar que prácticamente se alinean los *story points* ideales con los reales, debido a que el equipo ya había adquirido una amplia experiencia en el proyecto, y logró trabajar de una mejor manera.

Story points reales vs estimados

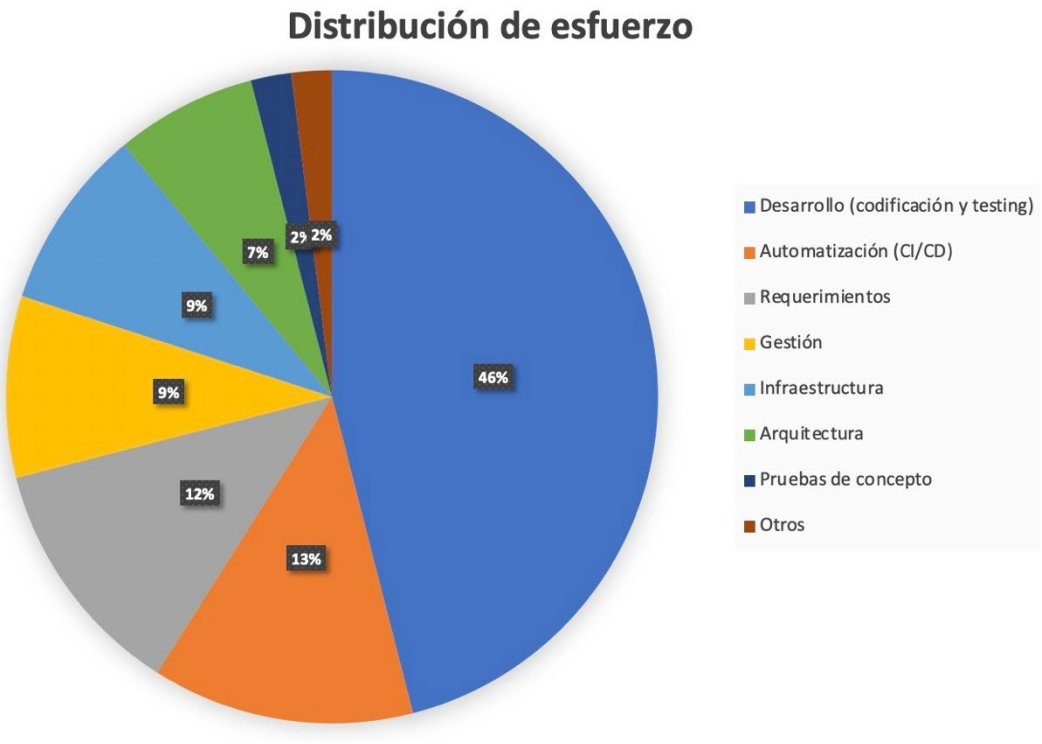


Gráfica 9 - *Story points* reales vs estimados

En la gráfica anterior se puede ver la comparación entre los *story point*s estimados por el equipo, y los completados para cada *sprint*. Como se puede apreciar, al principio el equipo no lograba completar con la estimación, ya sea por querer estimar tareas de más, o por la inexperiencia tanto en el proyecto como con las tecnologías. A medida que fue avanzando el proyecto, el equipo logró alinear de mucho mejor manera la estimación con la cantidad de *story points* a completar.

Distribución de esfuerzo

En el tablero de *Scrum* se asignaron categorías a las diferentes tareas con el objetivo de medir la distribución de esfuerzo realizadas a lo largo de la fase de desarrollo.



Gráfica 10 - Distribución de esfuerzo en fase de desarrollo

Tal como se puede visualizar en la anterior gráfica, la dedicación conjunta de desarrollo, automatización y relevamiento de requerimientos conforman más de un 70% de la dedicación en la fase de desarrollo.

Por un lado, esto cobra sentido ya que se dio lugar al desarrollo del sistema, que ya de por si ocupó casi la mitad de esfuerzo de toda la fase. Además, el equipo se centró en aplicar las practicas asociadas a DevOps como la integración y entrega continua, que requirieron de un gran esfuerzo para que quede funcional. Por otro lado, al ser un proyecto ágil que necesitó de cambios de requerimientos y validaciones con el cliente, es normal ver que el relevamiento de requerimientos siguió ocupando un lugar considerable en el esfuerzo del equipo.

También, analizando la gráfica, se puede ver como la infraestructura requirió menos esfuerzo debido a la investigación realizada en las etapas previas.

Por último, al realizar una arquitectura evolutiva debido a los requerimientos cambiantes, fue posible mantener constancia en el desarrollo de esta sin que esta requiera de un esfuerzo fuera de lo necesario.

7.4 Gestión de la comunicación

Una buena comunicación entre los diferentes actores del sistema fue necesaria para cumplir con los objetivos y metas establecidas. Permitió estar en contacto durante todo el proyecto y obtener retroalimentación de cada uno.

Por un lado, la comunicación dentro del equipo ayudó a trabajar de forma ordenada y mantenerse alineados. Con el cliente sirvió para entender sus necesidades y no desviarse del producto a construir y con el tutor para despejar todas las dudas que fueron surgiendo y aprovechar su trayectoria y conocimiento. A su vez, se establecieron los medios de comunicación utilizados entre las diferentes partes.

7.4.1 Comunicación entre el equipo

En un principio del proyecto, se conversó sobre la posibilidad de juntarse de forma presencial tres veces por semana, pero el surgimiento del COVID-2019 impidió que así lo sea. La mayoría de los encuentros se dieron de forma remota y las herramientas utilizadas fueron *WhatsApp* y *Google Meet*.

7.4.2 Comunicación con el cliente

Los encuentros con el cliente ocurrieron de forma presencial en su empresa hasta el comienzo de la pandemia. Durante esta, se acordó que por motivos sanitarios la comunicación iba a ser remota y mediante *Skype*. Antes de cada reunión, se le mandaba un mail al cliente con un breve resumen de cuál iba a ser la agenda de los temas a tratar durante la misma. Esto se hizo con el objetivo de que el cliente sepa de antemano cuales eran las inquietudes del equipo. En las primeras reuniones se habló sobre el dominio del

problema y a medida que se fue avanzando en la investigación se relevaron los requerimientos del sistema. Entrando en la fase de desarrollo se le fue entregando incrementos del producto resultante de los diferentes sprints. Cabe resaltar que durante todo el proyecto el cliente respondió de forma muy rápida y siempre estuvo predispuesto a ayudar en lo que el equipo necesitó.

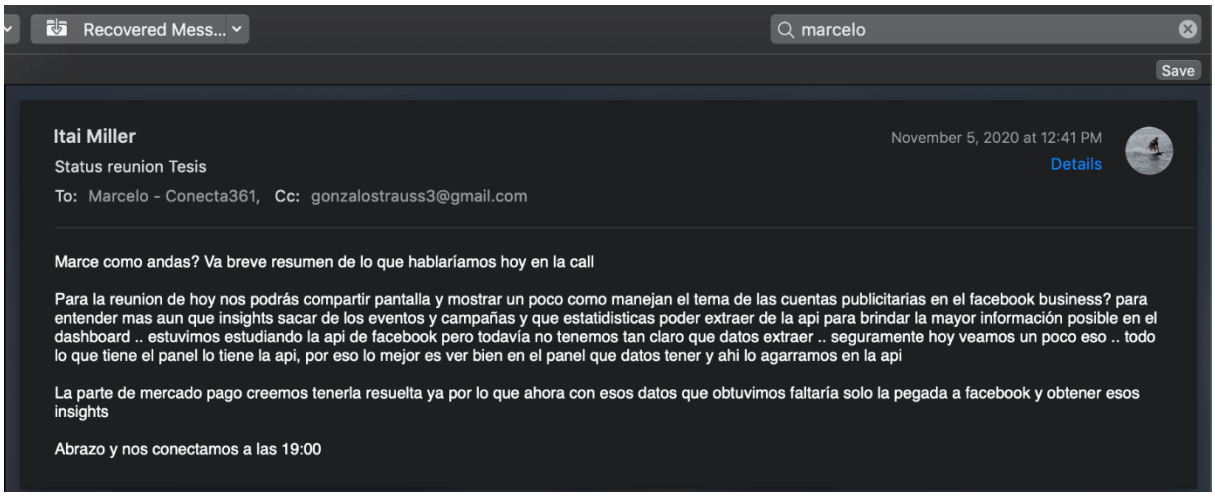


Ilustración 55 - Ejemplo de comunicación con el cliente

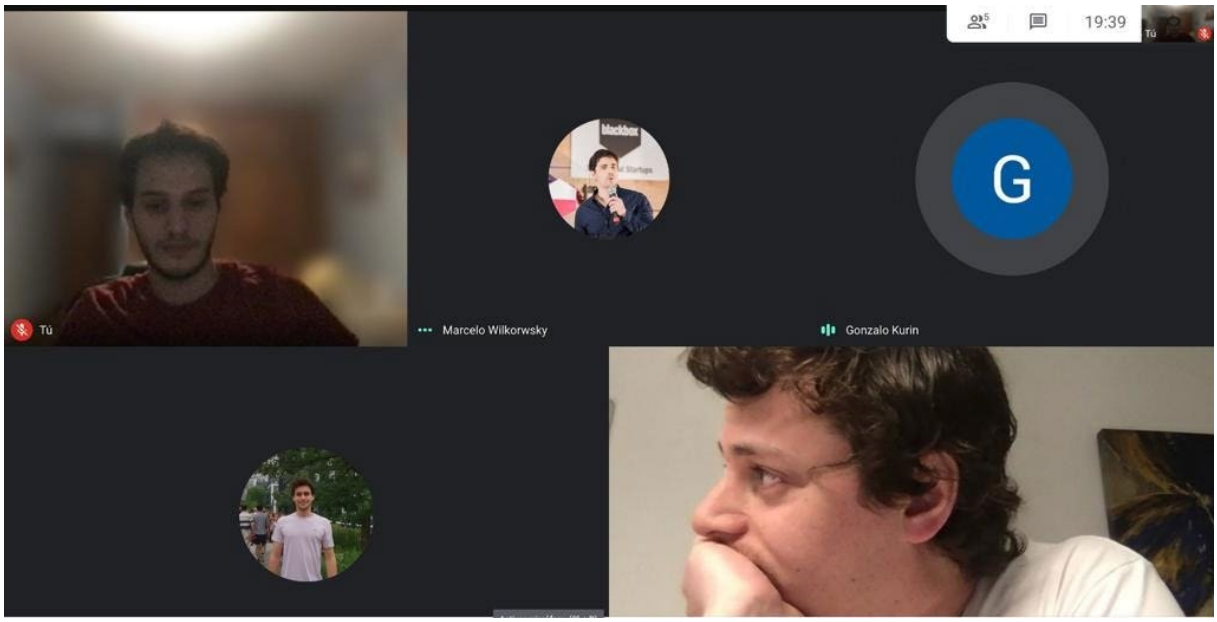


Ilustración 54 - Video llamada con el cliente

7.4.3 Comunicación con el tutor y docentes de ORT

La comunicación con el tutor fluyó durante todo el proyecto. Se realizaron reuniones semanales los días miércoles en la cual se hacía un breve resumen del progreso de la misma. La herramienta utilizada fue *Microsoft Teams*. Se realizaron 27 reuniones entre el 20/04 y 21/12 por *Microsoft Teams.* También se utilizó *WhatsApp* para todas las dudas que fueron surgiendo en el día a día.

Cabe destacar que, a lo largo del proyecto, se tuvo tres revisiones formales con docentes de ORT. El detalle de las mismas se puede ver en el Anexo 13.16. Además, se tuvo reuniones como parte del proceso académico con docentes que son referentes en diferentes áreas. Se tuvieron reuniones con Amalia Álvarez y Martin Solari con la finalidad de tener la opinión de un tercero.

7.5 Gestión de riesgos

Un riesgo es *"un evento o condición incierta que, si se produce tiene un efecto positivo o negativo sobre al menos un objetivo del proyecto como tiempo, costo, calidad o alcance"* [96]. El equipo decidió elaborar un plan de gestión de riesgos que permitió enfocar, planificar y ejecutar las actividades a lo largo del proyecto.

Desde un principio del proyecto, el equipo hizo hincapié en afrontar los riesgos identificados y reducir la probabilidad de los mismos durante el proyecto con el propósito de no desviarse de los tiempos del mismo. En primer lugar, se detallarán los riesgos identificados. Luego se hará un análisis cuantitativo de cada uno junto con los planes de respuesta y contingencia. Finalmente, se explicará el control y seguimiento realizado.

7.5.1 Metodología utilizada

El equipo realizó una metodología para gestionar los riesgos de manera que estos no se conviertan en problemas. Para esto, se desarrolló un plan de respuesta y contingencia

para cada uno de los riesgos identificados. Una vez por mes se reevaluaron todos los riesgos identificados, así como también se incluyó alguno nuevo.

Como primer paso, se identificaron todos los posibles riesgos del proyecto. Una vez identificados, se los pondero indicando la probabilidad e impacto de estos.

Los valores utilizados para medir el impacto fueron 1, 3 y 5 siendo 1 el que menos afectaba al proyecto y 5 el que más incidencia tenía. Los valores de probabilidad oscilaron entre 0 y 1 siendo 1 la probabilidad máxima de convertirse en problema.

A continuación, se muestran los valores en las siguientes tablas:

Impacto	Descripción
1	Impacto bajo en el proyecto
3	Impacto medio en el proyecto
5	Impacto alto en el proyecto

Tabla 6 - Escala de impacto

Probabilidad	Descripción
0	No probable
0.2	Poco probable
0.4	Probable
0.6	Muy probable
0.8	Altamente probable

1	Se convierte en problema

Tabla 7 - Escala de probabilidad

Luego de haber obtenido los valores de impacto y probabilidad de cada riesgo, se obtuvo el valor de la magnitud multiplicando el impacto por la probabilidad (M = I * P). Luego, se categorizaban los riesgos ya que facilita el seguimiento de los mismos. A su vez, se estableció una estrategia de respuesta siendo estas: mitigar, aceptar y evitar. Por último, se planteó un plan de contingencia en caso de que el riesgo se materializara.

7.5.2 Identificación de riesgos

Sobre el inicio del proyecto, el equipo decidió utilizar la técnica lluvia de ideas para pensar todos los posibles riesgos que podía tener el proyecto. El propósito de utilizar esta técnica fue porque no se tenía claro el dominio del problema y herramientas a utilizar. A medida que iba avanzando el proyecto, se fueron identificando nuevos riesgos que fueron surgiendo. Fue muy importante detectar los riesgos que podían afectar el proyecto ya que mientras más temprana es su detección, más rápida es su mitigación.

Se agruparon los riesgos por categoría ya que facilitaron el seguimiento de los mismos. A continuación, se detallan algunas de las principales:

Categoría	Descripción
Producto	Conocimiento del negocio y problemas del dominio que impactan en el producto y en el uso de este.
Equipo	Poco compromiso de los integrantes del equipo.
Tecnología	Involucra mala elección o uso de las tecnologías, o problemas al integrarse con las distintas plataformas como Facebook, Google o

	Mercado pago. También problemas de aplicación de la automatización
Planificación	Involucra tener una mala gestión del proyecto.
Estimación	Riesgos relacionados a cumplir con las fechas estipuladas para no alterar el alcance y tiempos del proyecto.
Tecnología	Que la arquitectura del proyecto no esté preparada para soportar la automatización de la integración y despliegue continuo

Tabla 8 – Categoría de riesgos

7.5.3 Análisis cuantitativo

Durante todo el proyecto se fueron identificando nuevos riesgos. Esto se debe a los requerimientos cambiantes. Una vez al mes se llevó a cabo un análisis cuantitativo.

A continuación, se puede observar el riesgo más relevante de cada una de las categorías mencionadas anteriormente.

Riesgo	Categoría	Estrategia de respuesta	Plan respuesta	Plan contingencia
Poco conocimiento del equipo sobre marketing digital	Producto	Mitigar	Investigación y reuniones con el cliente	Aumentar la cantidad de horas en investigación
Poco compromiso del equipo	Equipo	Evitar	Acuerdo de compromiso entre	Reunión entre el equipo para buscar una

			integrantes según intereses y objetivos personales	solución y motivar al integrante para que aumente su compromiso
Cambios de API de Google y Facebook	Tecnología	Aceptar	Fijarse continuamente en la documentación para detectar de forma temprana los cambios.	Aceptar
Mala gestión del proyecto	Planificación	Mitigar	Planificación en etapa temprana y evaluación de herramientas y buenas prácticas para la gestión	Si el equipo no puede gestionarse, se pedirá ayuda al tutor para una mejor organización
Mala estimación de historias de usuario	Estimación	Mitigar	Realizar investigaciones antes de comenzar a estimar. Dejar margen al estimar.	Volver a estimar las historias de usuario a realizar.

Que la arquitectura del proyecto no esté preparada para soportar la automatización de la integración y despliegue continuo	Tecnología	Mitigar	Planificar la arquitectura de acuerdo a los requerimientos impuestos por la tecnología de automatización	Priorizar los aspectos del sistema que necesiten ser automatizados, y limitar la automatización a los más prioritarios

Tabla 9 - Riesgos principales

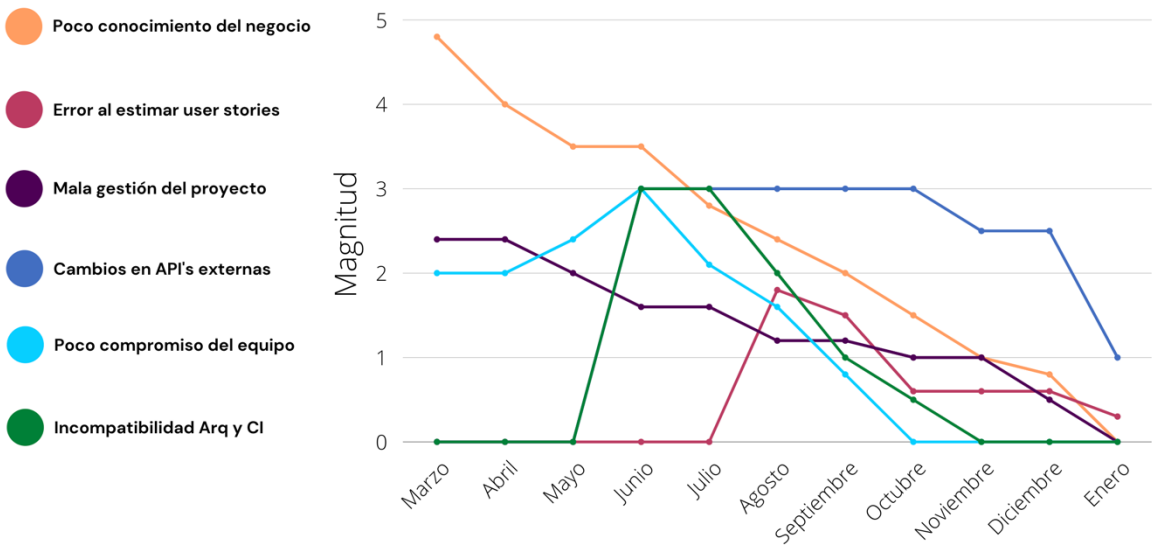
En el Anexo 13.17 se pueden ver el detalle de cada uno de estos riesgos. Se presenta la el nombre, descripción, categoría, estrategia de respuesta, plan de respuesta y plan de contingencia.

7.5.4 Control y seguimiento

A lo largo del proyecto, se realizó una vez por mes una evaluación y actualización de los riesgos identificados con el propósito de tener un mayor control y seguimiento de los mismos. Los que tenían mayor magnitud se monitoreaban antes que los que tenían menor magnitud.

Al tratarse de un proyecto complejo y con gran incertidumbre en el dominio del problema, la magnitud de los requerimientos fue cambiando e incluso fueron surgiendo algunos nuevos. Desde el inicio del proyecto se buscó evitar que los riesgos lleguen a materializarse.

La herramienta utilizada para el seguimiento fue *Google Sheets*. A continuación, se puede visualizar una gráfica de cómo fueron evolucionando los principales riesgos.



Gráfica 11 - Evolución de los riegos principales

En la ilustración anterior se puede visualizar como fueron evolucionando mes a mes la magnitud los principales riesgos. En el siguiente Anexo 13.18 se puede visualizar el seguimiento en detalle de cada uno de estos.

7.6 Resumen

Las características del proyecto determinaron la forma en la que el equipo realizó la gestión del mismo. Por un lado, el desconocimiento total del dominio por parte de los integrantes y la incertidumbre de una solución viable requirieron de una fase de investigación extensa. En esta se realizaron diferentes técnicas de validación del problema, como pruebas de concepto, ingeniería inversa, prototipos y otras técnicas de relevamiento de requerimientos junto al cliente. Luego de transitada esta, con la metodología ágil como base y *Kanban* como marco de gestión, se dio lugar a una fase de

desarrollo mediante *Scrum y* varios de sus eventos. La fase de desarrollo consistió en codificar la solución teniendo a la automatización como punto de referencia, tanto para las pruebas, como para el despliegue de la solución. Esta fue en concordancia a las necesidades del cliente, que precisaba de una entrega de software de valor rápida y eficaz. En paralelo al desarrollo se identificaron posibles riesgos del proyecto de forma de realizar un seguimiento de cada uno de ellos y evitar que se materialicen. En todo momento se siguieron utilizando técnicas de relevamiento de requerimientos, en donde el equipo siguió proponiendo ideas y validándolas con el cliente.

8. Automatización del proceso de ingeniería de software: un acercamiento a *DevOps*

En este capítulo se explicarán las practicas utilizadas para automatizar los procesos de la ingeniería de software y su importancia dada las características del proyecto (ver Marco metodológico). El equipo se enfocó en utilizar varias de las buenas prácticas de DevOps como manera de acercarse a dicha metodología y poder implementarla en el futuro. Esta es una metodología que se ha vuelto un estado de la practica en la industria, por lo que captó el interés y curiosidad del equipo para tomarlo como un aprendizaje dentro de los objetivos académicos del proyecto [97]. A su vez, se hará un especial énfasis en la implementación de la integración y entrega continúa debido al valor que agregaron a los procesos de desarrollo.

El software y la Internet lograron transformar al mundo y sus industrias. El uso del mismo ya no está únicamente limitado a respaldar un negocio, sino que es parte integral de cada aspecto de este. Al igual que la revolución industrial del siglo XX implicó una automatización en el modo en que las compañías de productos físicos diseñaban, creaban y entregaban sus productos, las compañías de hoy en día deben transformar el modo en que crean y entregan software [98].

La incertidumbre del dominio del problema, junto con los requerimientos cambiantes por parte del cliente, requirió al equipo adoptar un enfoque que le permita realizar entregas frecuentes de forma de validar el software rápidamente. La elección de una metodología ágil (véase en Metodología y ciclo de vida) implica ser – valga la redundancia - ágil en todos los aspectos del desarrollo.

8.1 DevOps

El término de *DevOps* surge como un acrónimo en inglés de *development* (desarrollo) y *operations (*operaciones*)* por primera vez en la conferencia *Agile* 2008 Toronto en una

charla sobre "*Agile infraestracture and operations*" [99]. Allí los interlocutores plantearon los diferentes malentendidos que ocurren entre los equipos de desarrollo y operaciones causando retrasos en la entrega de sus proyectos [100].

"DevOps es una metodología de desarrollo de software basada en la integración entre desarrolladores y administradores de sistemas, que permite que los desarrolladores puedan enfocarse sólo en desarrollar y puedan desplegar su código en segundos [101]*".*

A continuación, se puede visualizar el ciclo de vida de DevOps, diferenciando entre los procesos a cargo de los desarrolladores y los procesos a cargo de los integrantes del equipo de operaciones:

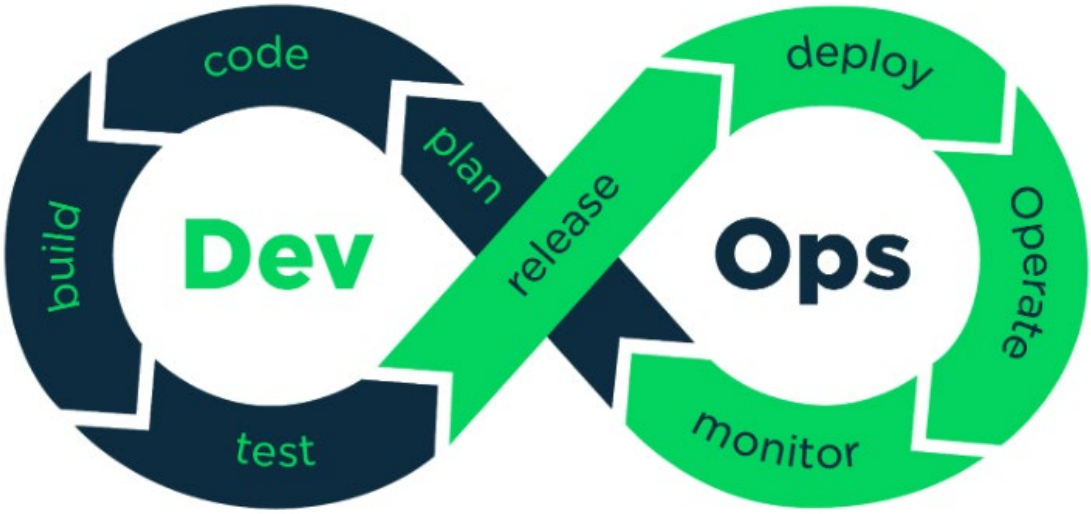


Ilustración 56 - Ciclo de vida de *DevOps*

Es importante destacar que el equipo no utilizó *DevOps* como tal, ya que para ello era necesario un seguimiento de las practicas mucho más estrictas que las viables en el equipo. *DevOps* tiene como objetivo mejorar la colaboración entre desarrolladores y operaciones mediante la automatización de las tareas desde el plan de trabajo hasta la puesta del software en producción [102]. Sin embargo, el equipo no contaba con roles de operaciones dado el tamaño del proyecto. De todas maneras, se utilizaron varias de las practicas que marcan a la cultura de *DevOps* [103]. Estas permitieron automatizar los

procesos y estándares del equipo, mejorar la productividad y maximizar la eficiencia, entregando software frecuentemente al cliente.

8.2 Integración y despliegue continuo

La integración continua (CI por sus siglas en inglés) es una práctica de desarrollo de software en la que los desarrolladores fusionan los cambios de código en la rama máster con frecuencia. En la integración continua se utilizan pruebas automáticas, análisis de código y otras prácticas de aseguramiento de calidad, que se ejecutan cada vez que se hace "*commit*" de código nuevo. De este modo, el código de la rama principal siempre es estable [104].

La entrega continua (CD por sus siglas en inglés) es la implementación automática y frecuente de nuevas versiones de una aplicación en un entorno de producción mediante el pasaje de diferentes *stages* que incluyen pruebas, liberación y despliegue de versión. Al automatizar los pasos necesarios para la implementación, los equipos reducen los problemas que pueden surgir en ese proceso y permiten actualizaciones más frecuentes.

Cuando se establecen estas dos prácticas, el proceso resultante es CI/CD, que incluye la automatización completa de todos los pasos desde que se hace "commit" del código hasta que su despliegue en el entorno de producción. La implementación de CI/CD permite a los equipos centrarse en la creación de código y elimina la carga y la posibilidad de errores humanos en los pasos cotidianos que se realizan manualmente. CI/CD también agiliza el proceso de implementación de nuevo código y reduce los riesgos que conlleva. Por tanto, las implementaciones son más frecuentes y se realizan en incrementos más pequeños, lo que ayuda a los equipos a ser más agiles y productivos, y a confiar más en el código que se ejecuta.

En la siguiente ilustración se visualiza la combinación entre estas dos practicas con los flujos explicados anteriormente:

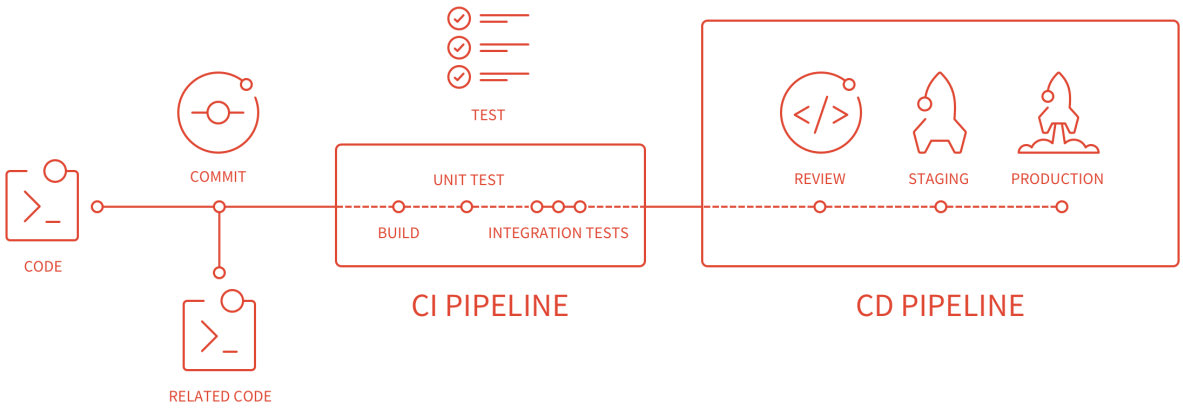


Ilustración 57 - Flujo de CI/CD [142]

A continuación, se detallará el proceso que el equipo realizó para aplicar la integración y despliegue continuo en los diferentes sistemas del proyecto. Se describirán los pasos necesarios previos a su ejecución, partiendo desde la investigación, construcción y elección de herramientas hasta la propia ejecución de los *pipelines*.

8.2.1 Investigación y construcción

El equipo tuvo que tomar varias decisiones a nivel de desarrollo, tecnologías y construcción de la arquitectura para poder soportar y asegurar que la automatización de las prácticas de integración y despliegue continuo se cumplan de la manera deseada.

Una de las primeras decisiones que el equipo tuvo que tomar fue sobre la elección de la herramienta de integración continua. Para ello se realizó un *spike* al comienzo del proyecto para analizar las diferentes herramientas que existen en el mercado y evaluar su viabilidad en la aplicación sobre el proyecto.

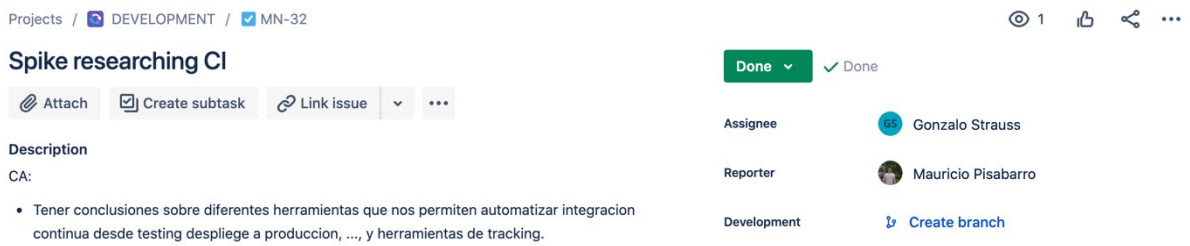


Ilustración 58 - *Spike de Integracion Continua*

El equipó comenzó utilizando *Bitbucket Pipelines*[8] debido a que los repositorios del proyecto estaban almacenados en *Bitbucke*t *y* a su vez este permitía integrarse directamente con Jira. Luego de varios días de investigación y desarrollo, realizando pruebas de concepto y simulando los pasos necesarios del *pipeline*, se optó finalmente por dejar de utilizar la herramienta debido a, principalmente, los costos luego del periodo de prueba finalizado.

Luego de una extensa búsqueda de herramientas disponibles, entre las que se encontraban *Jenkins*[9]*, CircleCI*[10]*, GitHub Actions*[11] y *Travis CI*[12]*,* la última mencionada fue la elegida debido a su paquete de educación gratuito y flexibilidad en la construcción de los pipelines (ver Anexo 13.19).

[8] https://bitbucket.org/product/es/features/pipelines
[9] https://www.jenkins.io/
[10] https://circleci.com/
[11] https://docs.github.com/es/actions
[12] https://www.travis-ci.com/

Travis CI

Como plataforma de integración continua, Travis CI se encarga de proveer *cloud virtual machines* en diferentes sistemas operativos para la ejecución de sus *pipelines.* Estos son configurados para la creación y pruebas automáticas de cambios de código, así como el despliegue de las aplicaciones hacia los servidores de *Amazon Web Services.* La utilización de Travis permitió configurar de manera específica acorde a las necesidades del proyecto, en donde la automatización de las dependencias, liberación de versiones, *gitflow,* pruebas y despliegue se hicieron de forma centralizada bajo una plataforma.

Cabe destacar que el hecho de utilizar *Travis CI* tuvo un impacto directo en la gestión de configuración del proyecto. Debido que al momento de la implementación de *Travis CI* este no contaba con soporte con *Bitbucket* (a fecha de hoy si lo tiene), el equipo tuvo que migrar todos sus repositorios de *Bitbucket* a *GitHub*.

Diseño y arquitectura

El equipo tuvo que diseñar la arquitectura del sistema y tomar decisiones relacionadas al desarrollo e infraestructura de este en base al objetivo de automatizar los procesos de pruebas y despliegue. Es por esta razón que se tomaron varias decisiones tecnológicas al respecto como forma de mitigar los riesgos que pudiesen estar asociados a construir un software que no esté preparado para tal automatización.

A nivel de infraestructura, el equipo creó servidores independientes para los ambientes definidos; *staging* y producción. Estas son instancias de *Amazon Web Services* con las mismas características a nivel de sistema operativo y de hardware. Incluso se crearon en las mismas regiones físicas para no tener diferencias en la latencia con estas. Esta misma decisión se aplicó para las bases de datos y otros servicios *cloud* ofrecidos por *AWS* como *S3*. De esta forma se logró tener sistemas idénticos pero independientes con respecto a los datos, lo que permite que las pruebas en una no afecten a la otra. Como una posible mejora a futuro, el equipo consideró la evaluación de utilizar contenedores

con tecnologías similares a *Docker* para asegurar la concordancia de los ambientes de manera automática. Esta no se aplicó debido al escaso conocimiento del equipo en estas tecnologías y a la priorización de aprendizaje de otras sobre esta.

En lo que respecta al desarrollo de software, se realizaron dos *templates* de desarrollo; uno para los servicios de *backend* y otro para los *frontend.* Estos *templates* se construyeron de forma de no perder tiempo en configurar cada uno de los proyectos para su integración con las pruebas y despliegue continuo. Los archivos de configuración y *scripts* necesarios fueron construidos de forma genérica y solo debiendo modificar parámetros puntuales al momento de la construcción de un servicio específico.

Dentro del Anexo 13.20, se desprende la explicación de los principales archivos de configuración dentro del *template.*

Herramientas de apoyo

A lo largo de toda la automatización de las prácticas de la ingeniería de *software*, y en específico de la integración y despliegue continuo, se utilizaron diversas herramientas que, de manera directa o indirecta, interactuaron en algunos de los pasos del proceso para dejar el software en producción. A continuación, se describen brevemente las más importantes:

Nexus

Nexus es un repositorio privado que funciona como un manejador de dependencias de los proyectos que utilizan *Maven* [88]. La herramienta también es utilizada como sistema de versionado para los diferentes componentes de un sistema. En el proyecto se utilizó a la misma para realizar las liberaciones de versión de las diferentes aplicaciones, así como también para importar las dependencias privadas de los mismos.

SonarQube

SonarQube es una plataforma que permite evaluar código fuente [105]. La misma fue utilizada para escanear a las diferentes aplicaciones del sistema e informar métricas respecto a código duplicado, *code smells,* estándares de codificación, deuda técnica, complejidad ciclomática entre otros. Fue integrado automáticamente con *Travis CI* para definir el éxito de un *build.* La herramienta fue específicamente diseñada para *Java*, lenguaje utilizado para el desarrollo del sistema en cuestión.

JUnit

JUnit es un conjunto de bibliotecas utilizadas para realizar pruebas unitarias en aplicaciones *Java*. Su utilización ayudó a definir los estándares de calidad integrados de manera automática en los *pipelines* de integración continua [106].

Jira

La herramienta fue integrada con *GitHub* para el seguimiento en tiempo real de las tareas asociadas a las *user stories.* Al momento de hacer *merges* y/o *builds* exitosos, *Jira* automáticamente sincronizaba los estados dentro de los *tickets* internos de su panel.

DataDog

La herramienta sirvió como plataforma para monitoreo de salud del sistema luego de cada despliegue a producción. En este se configuraron alertas y *dashboards* de monitoreo que permitieron enmendar fallas del sistema rápidamente [74].

8.2.2 Ejecución

A continuación, se explicarán los pasos aplicados por el equipo para la ejecución de la integración y despliegue continuo luego de una correcta configuración del proyecto. En el siguiente diagrama se pueden visualizar las fases necesarias para los diferentes *pipelines* de ejecución.

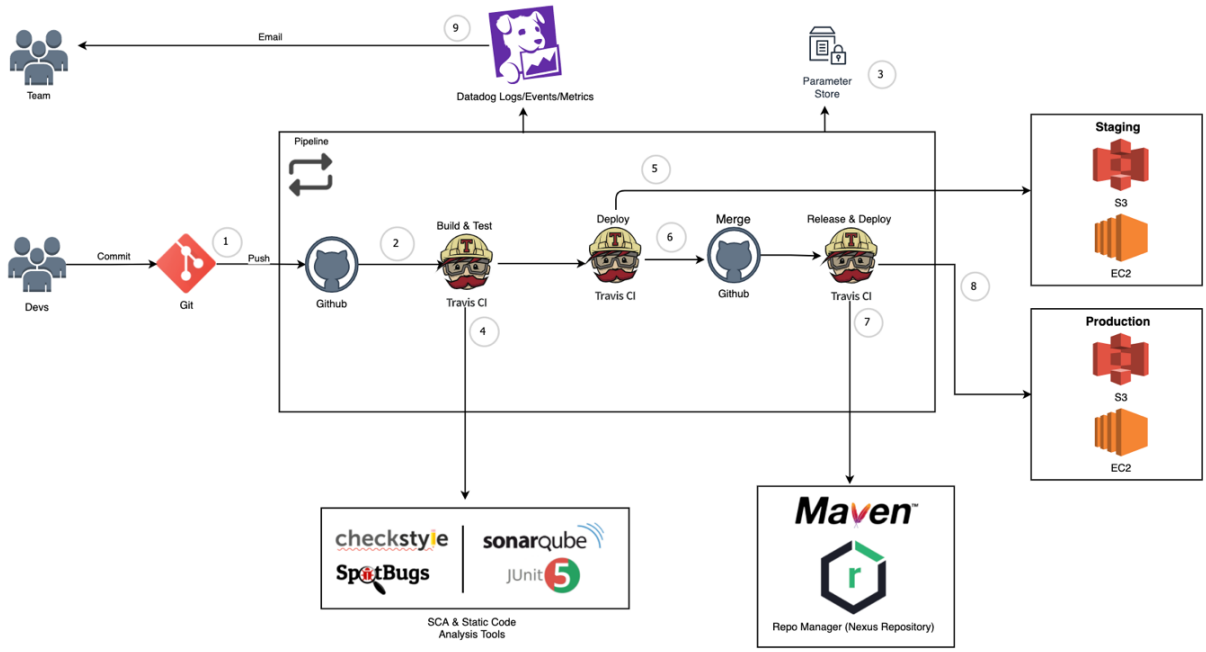


Ilustración 59 - Flujo de prácticas de DevOps

El paso 1 comienza al terminar una tarea de desarrollo, cuando se realiza un *commit* del código y este es compartido en el repositorio *cloud* de *GitHub*. El desarrollador a cargo realiza una *pull-request* con el objetivo de solicitar mergear su código a develop, como se visualiza en el paso 2.

Al momento de su realización – durante el paso 4 - se acciona un proceso automático en *Travis-CI* que compila el código fuente y ejecuta una serie de checks de código, pruebas unitarias, de integración y end-to-end con el objetivo de identificar cualquier regresión (véase en Automatización de pruebas y un acercamiento a DevOps). Previo a estas *Travis*

CI difiere las variables de entorno necesarias segun lo configurado dentro de su plataforma (paso 3).

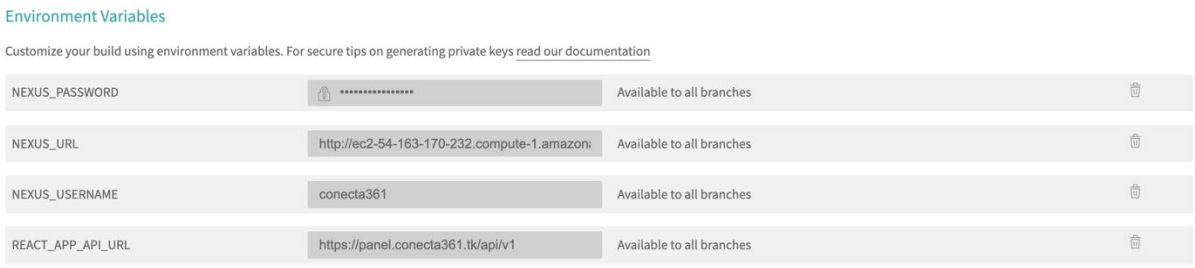


Ilustración 60 - Variables de entorno en *Travis CI*

Si el *build* falla, el *pull-request* también lo hará y notificará al desarrollador a tomar las acciones correspondientes para arreglar su código. En caso de ser exitoso, entonces otro proceso – visualizado como el paso 5 - será accionado de manera automática para realizar un *deploy* al ambiente de staging y realizar pruebas funcionales manuales y de integración.

Tal como ya fue mencionado anteriormente, el ambiente de *staging* es una réplica idéntica al de producción ya que el código desplegado es el mismo, pero independientes entre sí en datos e infraestructura.

Una vez realizadas las pruebas en el ambiente de *staging*, el *pull-request* deberá ser aprobada por otro desarrollador de forma de cumplir con *peer-reviewing* [107]. De esta forma se minimizan los problemas de integración que surgen al trabajar en un sistema compartido de código fuente, así como re-enmarcar los *bugs* en una instancia temprana del ciclo de vida del desarrollo.

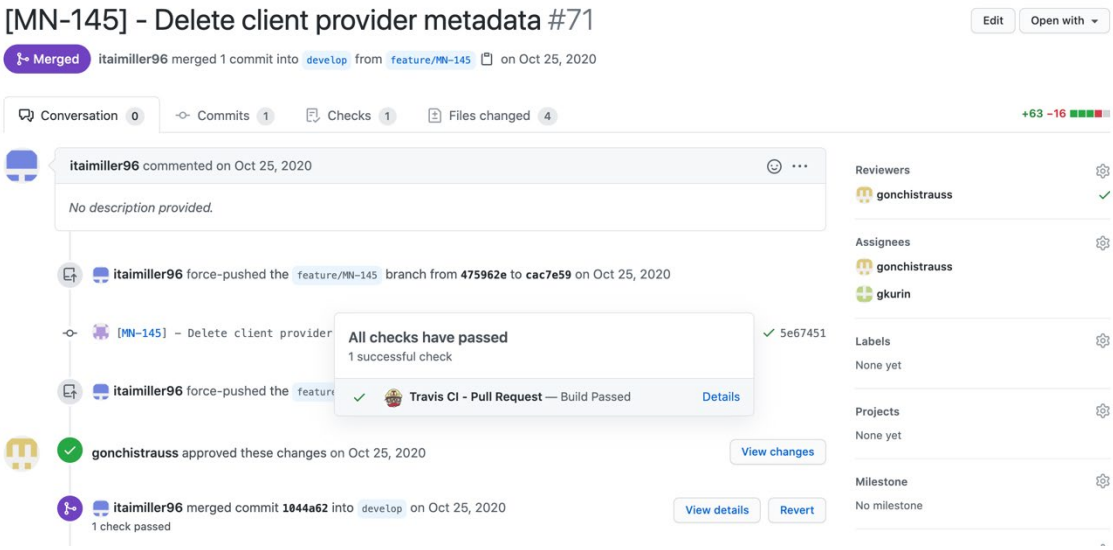


Ilustración 61 - *Pull Request* y *build checks*

Al mergear el *pull-request* en el paso 6, significa que los cambios de código pasaron todas las pruebas de manera exitosa y están prontas para ser desplegadas al ambiente de producción. Aquí comienza el proceso de despliegue continuo.

Tras este, se desencadena un proceso automático ejecutado por *Travis*; el *pipeline* para la liberación y despliegue a producción en el paso 7. Este proceso es encarga de generar una nueva versión y *tag* del proyecto, culminando en un *merge* hacia las ramas master y *develop*. Este se realiza utilizando *Maven* mediante los *plugins* de *gitflow*.

A su vez, el código generado es desplegado a *Nexus* como repositorio de versionado privado del equipo, de forma de centralizar las dependencias desprendidas de los diferentes proyectos.

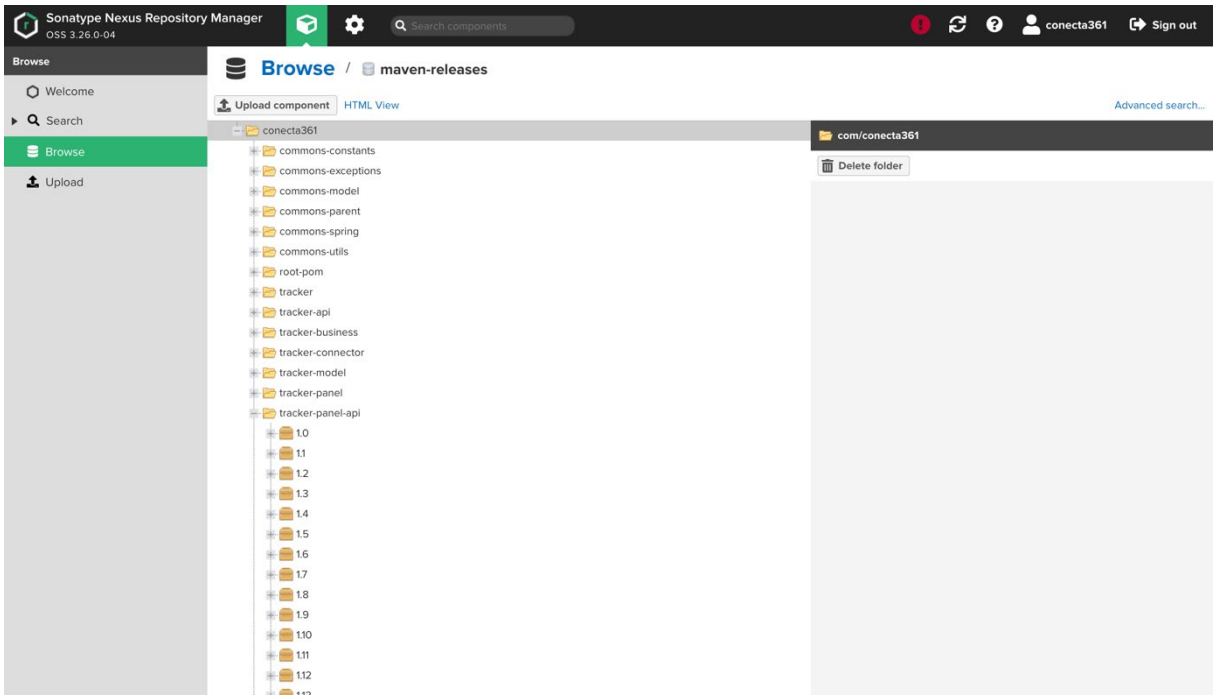


Ilustración 62 - Liberación de versiones en Nexus

Cuando la nueva versión del sistema queda disponible en los repositorios de *Github* y *Nexus*, el *build* es finalmente desplegado a producción con sus respectivas configuraciones en otro proceso automático independiente, visualizado como el paso 8. Al culminar el despliegue, un tick verde indicará que el despliegue fue exitoso y no hubo problemas en su ejecución.

Si por alguna razón la ejecución en producción tiene alguna falla, el sistema guarda la versión anterior en el sistema de manera de disponer de segundos para realizar un *rollback* mientras se realizan los cambios necesarios.

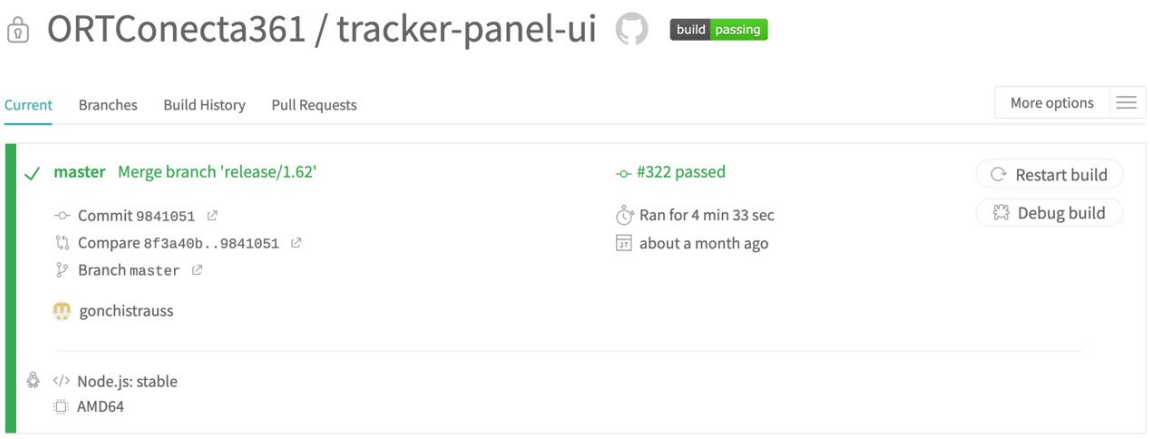


Ilustración 64 - Build success en Travis CI

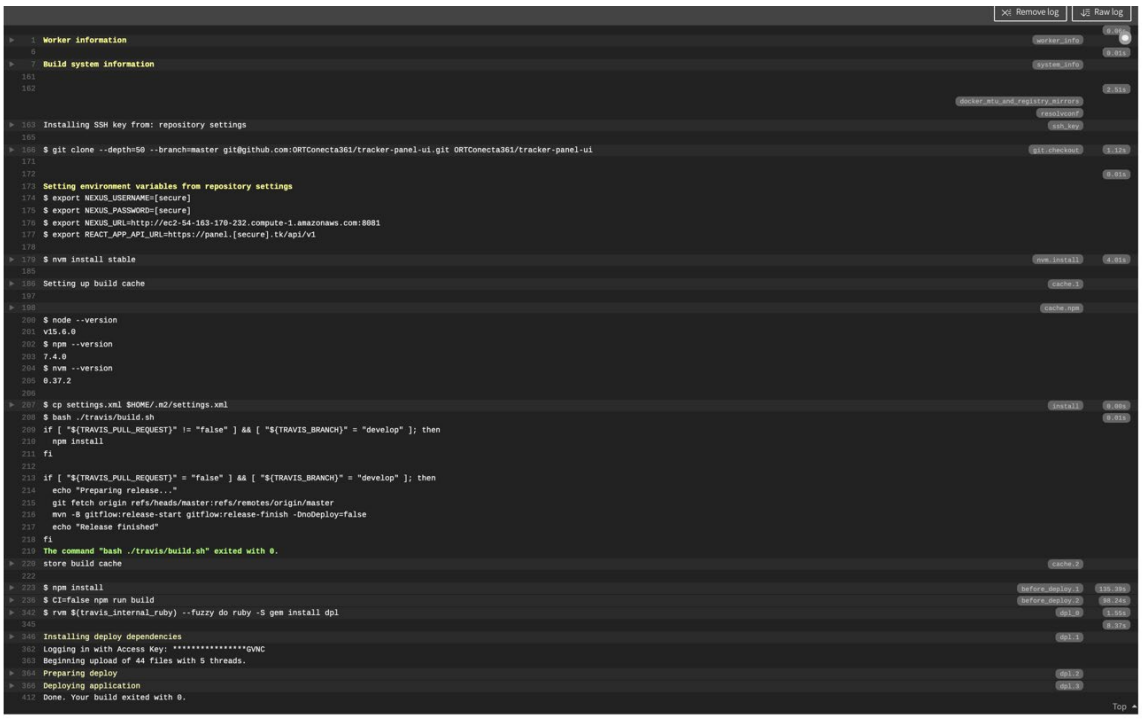


Ilustración 63 - Pipeline de despliegue en Travis CI

Luego de desplegado el código, el equipo se dedica a monitorear el estado de salud del sistema a través de DataDog.

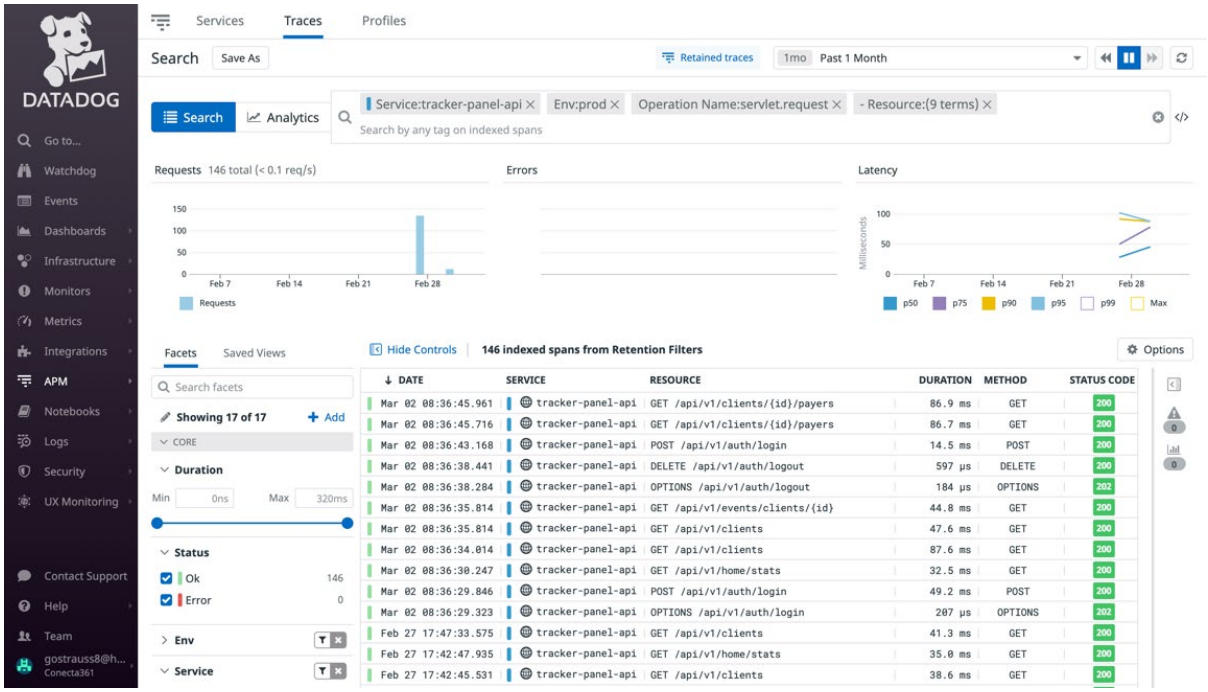


Ilustración 65 - Monitoreo en DataDog

A su vez, se programaron alertas automáticas enviadas por email en caso de que la salud de la aplicación no sea la correcta, visualizado como el paso 9.

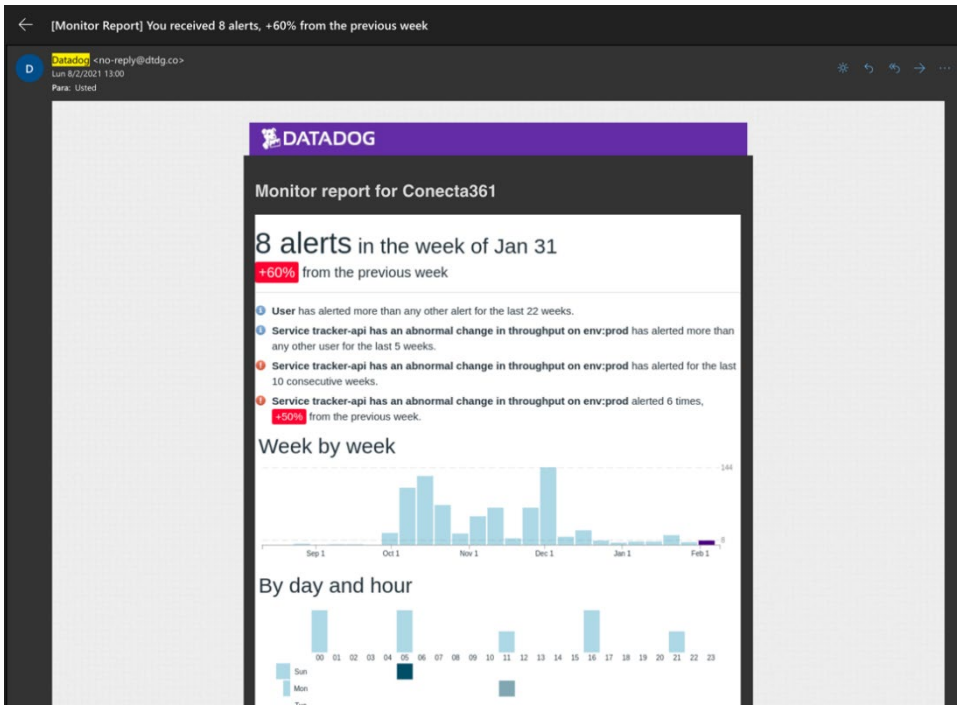


Ilustración 66 - Notificaciones de DataDog

8.3 Buenas prácticas utilizadas

Existen varias prácticas que ayudan a los equipos a agilizar, automatizar y mejorar una fase especifica. Varias de estas prácticas pertenecen a varias fases, homogenizando y aumentando la productividad de sus procesos [108].

A continuación, se detallarán las practicas fundamentales que el equipo decidió aplicar en el proyecto.

Desarrollo ágil de software

La utilización de un enfoque de desarrollo ágil implica reforzar el trabajo en equipo y requiere de una gran capacidad de adaptación a los cambios mediante ciclos cortos de lanzamiento de versiones. El equipo utilizó tanto *Kanban* como *Scrum* como marcos de gestión ágil dependiendo de la fase en la que se encontraba el proyecto.

Control de versiones

Esta práctica se encarga de administrar el código por versiones. El uso de esta ayudó a los desarrolladores a trabajar en conjunto, facilitando la división de tareas de programación y almacenamiento del código en caso de que una recuperación fuese necesaria. Para la aplicación de esta se utilizó el sistema de control de versiones de *Git y* repositorios en la nube de *Github* [109]. Si bien el control de versiones es una práctica en sí misma, también fue utilizada como elemento de otras prácticas tales como la integración continua.

Supervisión continua

Implica obtener el rendimiento y el estado de todo el *stack* de aplicaciones en tiempo real; desde la infraestructura en donde estos operan hasta los propios componentes de software. Consiste en recopilar datos de eventos y registros de varias partes del sistema para luego poder analizarlos y consultarlos.

8.4 Resumen

La automatización de algunas de las fases asociadas a *DevOps*, así como la aplicación de buenas prácticas, fortaleció los procesos de ingeniería de software. Esta garantizó construir, validar, desplegar el código en los distintos ambientes y realizar pruebas para que el sistema finalmente quede en producción en el menor tiempo posible. A su vez, este acercamiento deja abierta la posibilidad a que, si en un futuro se tienen los recursos necesarios y si se quisiese realizar *DevOps*, se pueda hacerlo. No es menor destacar que la investigación de esta metodología por parte del equipo se basó en gran parte a la curiosidad académica que este generaba. Tal como se menciona anteriormente, *DevOps* es una metodología que cada vez es más utilizada como estándar de los equipos agiles, y los beneficios de utilizarla abundan. A su vez, las practicas que englobaban a estas eran compatibles e incluso necesarias dados los objetivos planteados por el equipo al comienzo del proyecto y las características del mismo (véase Objetivos del proyecto)

9. Gestión de la configuración

En el presente capitulo se detalla el proceso realizado para la gestión de la configuración del *software* (SCM por sus siglas en inglés).

Su propósito fue que el equipo pueda identificar, organizar y controlar modificaciones en el proyecto de manera que sea escalable sin comprometer la calidad de este.

La gestión de la configuración del *software* es un conjunto de actividades desarrolladas para gestionar los cambios a lo largo del ciclo de vida. Es una actividad de garantía de calidad de software que se aplica en todas las fases del proceso de ingeniería de *software* [110].

En este proyecto, caracterizado por sus requerimientos cambiantes y entregas frecuentes, fue muy importante gestionar y automatizar la evolución del sistema a lo largo de todo el proceso de desarrollo.

A continuación, se identificarán los elementos de configuración, herramientas utilizadas, organización de los diferentes repositorios, metodología de trabajo y gestión de versionado a lo largo del proyecto.

9.1 Elementos de configuración

En la siguiente sección, se describirán los diferentes elementos de configuración. Estos fueron clasificados según su tipo de aplicación.

9.1.1 Software

La gestión de configuración de *software* del proyecto incluye a todos los componentes del sistema con sus respectivos repositorios. Estos fueron creados en *GitHub* y almacenados como código fuente (véase en el Anexo 13.21).

Elemento de configuración	Repositorio
Backoffice web	*tracker-panel-ui*
Backend	*tracker-panel*
API de conversiones	*tracker-api*
SDK de conversiones	*tracker-sdk*
Productor de eventos	*event-producer*
Consumidor de eventos	*event-consumer*
Bot de *web scrapping*	*event-scrapper*
Módulo de dependencias comunes	*root-pom*
Módulo de utilidades comunes	*commons-parent*

Tabla 10 - Elementos de *software*

9.1.2 Librerías

La solución del sistema requirió de diferentes librerías y dependencias para el correcto funcionamiento del mismo. Muchos de los componentes del sistema necesitaban de las mismas dependencias por lo que el equipo decidió crear un módulo cuya única responsabilidad sea proveer dependencias comunes a los distintos componentes del sistema. Este fue el *root-pom* – mencionado como elemento de software – que albergó a las diferentes dependencias comunes, siendo las principales de estas listadas a continuación (véase en Anexo 13.22).

Librería	Versión
Lombok	1.18.12
mapstruct	1.3.1 Final
apache-commons	50
spring-framework	5.2.6 *Release*
jsonwebtoken	7

Tabla 11 - Librerias

9.1.3 Infraestructura

Como ya fue mencionado a lo largo del proyecto, el equipo buscó la automatización del proceso de ingeniería de la forma más eficiente y rápida posible. Esto llevó a que, de entre otras decisiones, se utilicen a los servicios *cloud* de *Amazon Web Services* como *IaaS* y a su vez a *TravisCI* como servidor de CI/CD.

Una de las practicas más importantes asociadas a estas dos, y relacionadas estrechamente a la metodología de DevOps, es la *Infraestructure as a Code* (*IaC* por sus siglas en inglés) [111].

La *IaC* es la automatización de la infraestructura a través de código [112]. Esta automatiza, con la ejecución de código fuente, la provisión de componentes de infraestructura (código que define como queda configurada la infraestructura) [113].

Las aplicaciones pueden contener scripts para crear y organizar sus propias máquinas virtuales (VM por sus siglas en inglés). Se trata de una parte fundamental de la computación en la nube y es esencial para DevOps [114].

Si bien existen herramientas como *Ansible*[13] o *Terraform*[14] que se enfocan en la aplicación de IaC, el equipo se acercó a la aplicación de esta práctica mediante otro tipo de *scripts* o archivos de configuración más manuales. A cotinuación se detallan los elementos asociados junto con una breve explicación de su uso (véase en Anexo 13.23):

Elemento	Descripción
.travis.yml	Archivo de configuración del *pipeline* de *TravisCI*. En este se configuró el sistema operativo de la VM, así como el lenguaje, restricciones de seguridad y otras características a nivel de infraestructura.
application.yml	Archivo de configuración de la aplicación almacenado en el servidor. Contiene configuraciones de la base de datos, y reglas para el *reboot* automático del servidor en caso de una *memory leak.*
rpm > preinstall.sh / postinstall.sh / service.sh / run.sh	Los *scripts* dentro de la carpeta rpm son utilizados para la construcción de y despliegue de la aplicación a los servidores linux de AWS. Dentro de

[13] https://www.ansible.com/
[14] https://www.terraform.io/

	estos se configuran reglas de seguridad, de estado de salud de la aplicación, memoria y otras más anterior a la ejecución del artefacto.
travis > *build.sh* / *clean.sh* / *deploy.sh* / *run.sh*	*Scripts* de configuración creados para la puesta en modo de mantenimiento y operacional de los servidores que alojan a la aplicación.

Tabla 12 - Elementos de infraestructura

Cabe destacar que, si bien varios de estos entran en el proceso de IaC, este puede ser mejorado mucho más en el futuro. Ansible, por ejemplo, es un *software* hecho para automatizar el aprovisionamiento de software, la gestión de las configuraciones y el despliegue de las aplicaciones, por lo que utilizarla permitiría centralizar las configuraciones en un único lugar y de forma aún más automática. Otra posible mejora podría ser con la utilización de contenedores de *docker* o kubernetes, que permiten configurar sus características físicas y de infraestructura una sola vez y desplegadas en conjunto a los diferentes ambientes.

9.1.4 Documentación

Para la gestión, versionado y seguimiento de los que son de tipo documento, se optó por Google Drive para tener almacenados todos los documentos en la nube y por la cantidad de funcionalidades que la misma ofrece. También se utilizó OneDrive para el almacenamiento del documento final del proyecto (véase en Anexo 13.24).

Carpeta	Descripción
Anteproyecto	Contiene todos los documentos que formaron parte de la fase de anteproyecto.
Arquitectura	Contiene todos los documentos relacionados al diseño y arquitectura del sistema junto con sus diagramas.
Calidad	Contiene todo lo relacionado a las actividades de calidad. Incluye el plan de calidad, aseguramiento de la calidad, métricas y gestión de incidentes.
Demo	Contiene todo el contenido audiovisual que se presentara en la defensa final.
Diagramas	Contiene todos los diagramas del sistema.
Documentación	Contiene todo lo relacionado a la documentación de la entrega final.
Gestión Proyecto	Contiene todo lo relacionado a la gestión del proyecto.
Informe de Avance	Contiene el documento presentado en el informe de avance.
Infraestructura	Contiene todo lo relacionado a la infraestructura del sistema

Ing. Requerimientos	Contiene todo lo relacionado al relevamiento de requerimientos y prototipos presentados al cliente.
Integraciones	Contiene todo lo relacionado a las distintas integraciones que fuimos realizando junto con las pruebas de concepto.
Investigación	Contiene el trabajo realizado durante la etapa de investigación.
Prototipos	Contiene todos los prototipos creados para validar los requerimientos.
Reuniones	Contiene todo lo relacionado a las distintas reuniones que se hizo durante el proyecto.
Revisiones	Contiene todo lo relacionado a las tres revisiones que se hizo durante el proyecto.
Riesgos	Contiene todo lo relacionado a la identificación y mitigación de los riesgos.

Tabla 13 - Elementos de documentación

9.2 Elección de herramientas

A continuación, se describen las herramientas seleccionadas por el equipo para gestionar los elementos identificados anteriormente.

Para los elementos de tipo *software,* se decidió utilizar *Git* como control de versiones, y *GitHub* como servicio de alojamiento de los repositorios. Si bien el equipo comenzó utilizando *Bitbucket*, luego de seleccionar a *Travis CI* como herramienta de integración continua, se optó por migrarse a *Github* debido a la incompatibilidad con esta.

A su vez, tanto para los elementos de tipo software como para los de librerías, se utilizó *Nexus Repository Manager* como manejador de dependencias y artefactos privados. Esta herramienta fue instalada dentro de un servidor de *Amazon Web Services* de manera de que el acceso a este fuese únicamente posible mediante credenciales de sesión privadas [115]. La elección de *Nexus* favoreció la aplicación de integración con los servidores de integración continua.

Para los elementos de infraestructura, como ya fue mencionado anteriormente, no se eligieron herramientas específicas para tal acción. Sin embargo, la utilización de *Travis CI*, *plugins* de *Maven*, *scripts* rpm y configuraciones propias pueden incluirse dentro de esta clasificación.

Para la gestión, versionado y seguimiento de los que son de tipo documento, se optó por *Google Drive* para tener almacenados todos los documentos en la nube y por la cantidad de funcionalidades que la misma ofrece. Es una herramienta sencilla de entender que permitió trabajar de forma concurrente sobre los archivos facilitando el trabajo remoto debido al COVID 2019. Sin embargo, para la redacción de la documentación del proyecto final se utilizó a *Microsoft Word* y *OneDrive*, de manera de dar soporte a una mayor compatibilidad a la hora de la exportación del archivo a pdf.

Por último, dentro del contexto de *Scrum*, la gestión de los cambios de código fue notificados mediante *Jira*, integrando a los *tickets* automáticamente con *Travis CI* y *Github* para sincronizarse con los cambios en las ramas y *pull requests*.

9.3 Metodología de trabajo

El equipo decidió utilizar la metodología *GitFlow (*véase en Anexo 13.25*)* para mantener un estándar a la hora de desarrollar. Permitió que los desarrolladores trabajen en paralelo separando las funcionalidades en distintas *branches*.

A continuación, se definen las ramas que presenta el modelo:

- Rama *master*: contiene cada una de las versiones estables del proyecto.
- Rama *develop*: *branch* que integra las funcionalidades de desarrollo para luego fusionar a la rama master para su despliegue a producción.
- Rama *feature*: rama de apoyo que surgen de la rama *develop* con el objetivo de iniciar una nueva funcionalidad. Al finalizar con el desarrollo de la funcionalidad, se crea un pull-request y al ser aprobado luego se hace integra a *develop*.
- Rama *release*: rama de apoyo que surgen de la rama *develop.* Esta contiene el código de la versión a liberar próximamente. Contiene todo el código de *develop* para el despliegue.
- Rama *hotfix*: rama de apoyo que surgen de la rama master con el objetivo de arreglar un error en producción de manera urgente. Luego de solucionar el mismo, se integra nuevamente a master.

La metodología *GitFlow* fue utilizada en el contexto de automatización con las prácticas de integración y despliegue continuo, por lo que su aplicación fue configurada mediante scripts y reglas del servidor de CI/CD de *Travis CI*. Estas serán mencionadas a continuación y son explicadas en mayor detalle en el capítulo de Automatización del proceso de ingeniería de software: un acercamiento a *DevOps*:

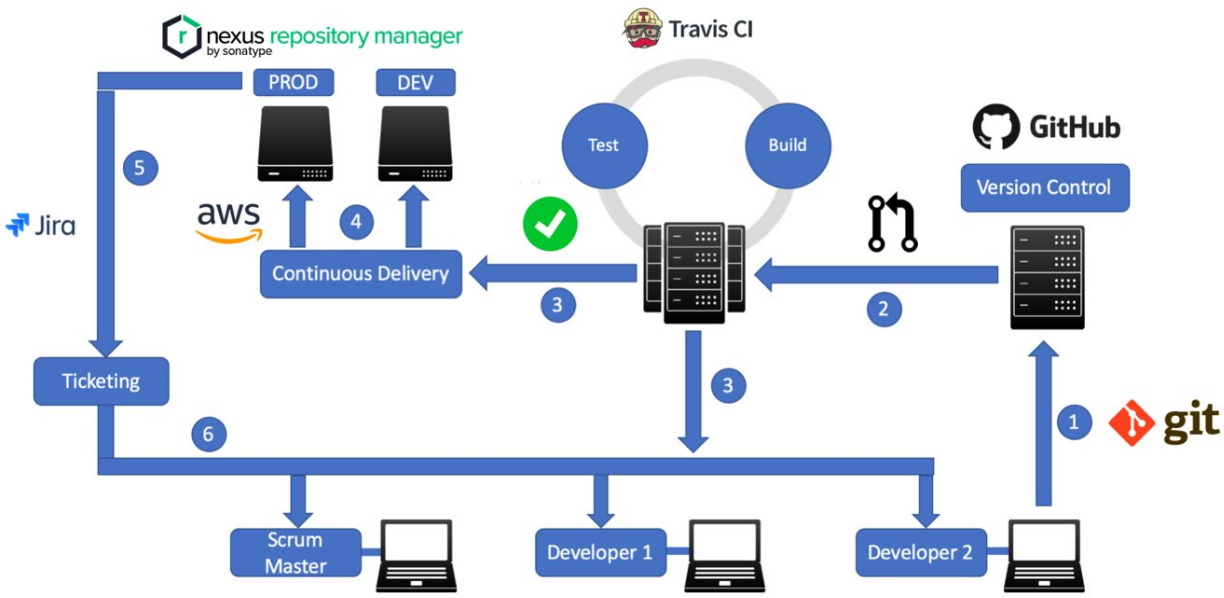
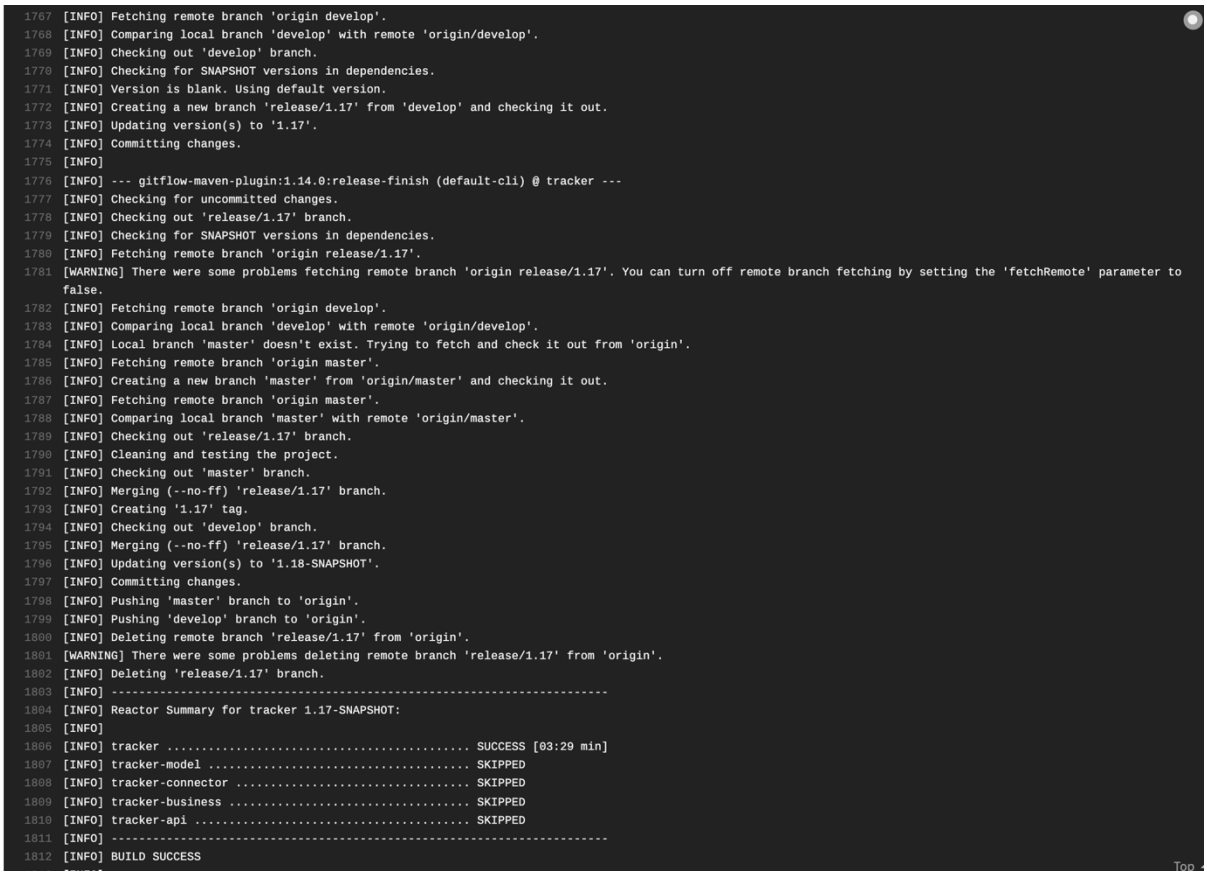


Tabla 14 - Metodología de trabajo mediante *pipelines* automáticos

Tal como se puede visualizar en la ilustración de arriba, luego de la finalización de la tarea de un desarrollador, este debía de realizar un *pull-request* de su rama *feature* hacia *develop.* Al finalizar las pruebas automáticas accionadas por el *pipeline* de integración y pasados los *checks* de calidad, este era *mergeado* a *develop.* De allí otro proceso es ejecutado por el servidor de integración continua, en donde es realizada la creación de una rama *reléase* y liberada una versión que es desplegada a *Nexus Repository* y *commiteada* como un *tag* en *GitHub*. De allí, los *merge* a *master* como *develop* son realizados y la versión se incrementa en 1.

Cabe destacar que el equipo utilizo *Maven*, que internamente maneja versiones para sus proyectos lo que benefició la organización de *snapshots* y *releases* de manera fácil (véase en Anexo 13.26). Por último, se realizaba el despliegue de la aplicación a los servidores correspondientes.

A continuación, se puede visualizar el proceso de versionado con *gitflow* realizado automáticamente mediante el *pipeline* de integración con *Travis CI.*



```
1767 [INFO] Fetching remote branch 'origin develop'.
1768 [INFO] Comparing local branch 'develop' with remote 'origin/develop'.
1769 [INFO] Checking out 'develop' branch.
1770 [INFO] Checking for SNAPSHOT versions in dependencies.
1771 [INFO] Version is blank. Using default version.
1772 [INFO] Creating a new branch 'release/1.17' from 'develop' and checking it out.
1773 [INFO] Updating version(s) to '1.17'.
1774 [INFO] Committing changes.
1775 [INFO]
1776 [INFO] --- gitflow-maven-plugin:1.14.0:release-finish (default-cli) @ tracker ---
1777 [INFO] Checking for uncommitted changes.
1778 [INFO] Checking out 'release/1.17' branch.
1779 [INFO] Checking for SNAPSHOT versions in dependencies.
1780 [INFO] Fetching remote branch 'origin release/1.17'.
1781 [WARNING] There were some problems fetching remote branch 'origin release/1.17'. You can turn off remote branch fetching by setting the 'fetchRemote' parameter to
     false.
1782 [INFO] Fetching remote branch 'origin develop'.
1783 [INFO] Comparing local branch 'develop' with remote 'origin/develop'.
1784 [INFO] Local branch 'master' doesn't exist. Trying to fetch and check it out from 'origin'.
1785 [INFO] Fetching remote branch 'origin master'.
1786 [INFO] Creating a new branch 'master' from 'origin/master' and checking it out.
1787 [INFO] Fetching remote branch 'origin master'.
1788 [INFO] Comparing local branch 'master' with remote 'origin/master'.
1789 [INFO] Checking out 'release/1.17' branch.
1790 [INFO] Cleaning and testing the project.
1791 [INFO] Checking out 'master' branch.
1792 [INFO] Merging (--no-ff) 'release/1.17' branch.
1793 [INFO] Creating '1.17' tag.
1794 [INFO] Checking out 'develop' branch.
1795 [INFO] Merging (--no-ff) 'release/1.17' branch.
1796 [INFO] Updating version(s) to '1.18-SNAPSHOT'.
1797 [INFO] Committing changes.
1798 [INFO] Pushing 'master' branch to 'origin'.
1799 [INFO] Pushing 'develop' branch to 'origin'.
1800 [INFO] Deleting remote branch 'release/1.17' from 'origin'.
1801 [WARNING] There were some problems deleting remote branch 'release/1.17' from 'origin'.
1802 [INFO] Deleting 'release/1.17' branch.
1803 [INFO] ------------------------------------------------------------------------
1804 [INFO] Reactor Summary for tracker 1.17-SNAPSHOT:
1805 [INFO]
1806 [INFO] tracker ..................................... SUCCESS [03:29 min]
1807 [INFO] tracker-model .............................. SKIPPED
1808 [INFO] tracker-connector ........................... SKIPPED
1809 [INFO] tracker-business ............................ SKIPPED
1810 [INFO] tracker-api ................................. SKIPPED
1811 [INFO] ------------------------------------------------------------------------
1812 [INFO] BUILD SUCCESS
```

Tabla 15 - *Gitflow* con *Travis CI*

Cabe destacar que luego de cada entrega los *tickets* de Jira eran sincronizados de manera de mantener actualizados los cambios de *software* en los *sprints (véase* en Anexo 13.27*).*

9.4 Resumen

La gestión de la configuración es fundamental en todos los proyectos, pero para los que requieran de realizar entregas y ser ágiles es necesario mantener una estricta organización del mismo para no afectar los flujos de sus procesos. En el caso puntual, esta estuvo muy ligada a la automatización de los procesos de ingeniería de software y a las prácticas de integración y entrega continua, que permitieron configurar y versionar a los proyectos de manera automática y rápida.

10. Gestión de la Calidad

En este capítulo se hará una descripción de las diferentes actividades realizadas, estándares utilizados, y buenas prácticas, con el fin de asegurar la calidad del proyecto. Para ello se detallan los objetivos de calidad del producto y que es lo que esto significa. Por último, se mostrarán las métricas de calidad utilizadas para evaluar el avance a lo largo del proyecto.

10.1 Objetivos de Calidad del Producto

Tomando como referencia la definición de calidad de un producto según Joseph Juran- "*la calidad del software es el conjunto de características de un producto que satisfacen las necesidades de los clientes y, en consecuencia, hacen satisfactorio el producto*" [116]- el objetivo principal fue cubrir las funcionalidades básicas del producto visibles en la sección de Listado de requerimientos.

Estos objetivos fueron verificados y validados a lo largo del ciclo de vida, de manera de poder minimizar el riesgo de desviación con los mismos. Las verificaciones se dieron continuamente, durante el sprint a la hora de revisar *pull-requests*, y al finalizar el *sprint* durante *el sprint review*, y las validaciones se dieron en reuniones pactadas con el cliente generalmente al finalizar un *milestone*.

10.2 Plan de Calidad

Se elaboró un plan de calidad en el proyecto para cumplir con los objetivos propuestos. Este documento sirvió como referencia para planificar y llevar a cabo cada una de las distintas actividades. Para cada una de estas se identificaron datos de entrada y salida. A su vez, se definieron herramientas y tecnologías que ayudaron a llevarlas a cabo, y roles que definen quien se encargó y quien participó en las mismas.

El plan de calidad se encuentra bajo el Anexo 13.28.

10.3 Aseguramiento de la Calidad

Se definió el plan de calidad con el propósito de guiar y asegurar las distintas actividades de calidad para poder cumplir con los objetivos planteados. El plan fue realizado en el inicio del proyecto y fue actualizándose a medida que fue avanzando el mismo. A continuación, se detallan las prácticas y metodologías utilizadas.

10.3.1 Aplicación de estándares

Poder definir estándares para la documentación y codificación fue clave para asegurar que todos los integrantes del equipo tuvieran un mismo criterio a la hora de realizar entregables.

Estándares de documentación

Los estándares utilizados fueron los provistos por la Universidad ORT Uruguay.

Documento	Estándar	Referencia
Normas para la presentación de trabajos finales de carrera	Documento 302: Normas específicas para la presentación de trabajos finales de carrera.	ORT
Lista de verificación de formato	Documento 303: Lista de verificación de formato	ORT
Normas para el desarrollo de trabajos finales de carrera	Documento 304: Normas para el desarrollo de trabajos finales de carrera	ORT

Títulos, resúmenes e informes de corrección	Documento 306: Guía para títulos, resúmenes e informes de corrección	ORT
Documento final	Guía para la entrega final	ORT

Tabla 16 - Estándares de documentación

Estándares de codificación

Antes de comenzar con el desarrollo del software, el equipo decidió apegarse a un artículo conteniendo estándares y buenas prácticas para la codificación [117]. Esto permitió lograr un código legible y organizado, efectivamente acelerando el proceso de desarrollo (véase Anexo 13.29).

10.3.2 Capacitación en tecnologías

Antes de entrar en la fase de desarrollo, el equipo se tuvo que capacitar en ciertas tecnologías que sirvieron para el desarrollo e infraestructura del sistema. En lo que respecta al *frontend* solo Mauricio Pisabarro tenía conocimientos previos por lo que Gonzalo Kurin realizo tutoriales *online para* introducirse en la misma. Para el *backend* Itai Miller y Gonzalo Strauss reforzaron sus conocimientos sobre el *framework* Spring [118] [119].

También se realizó un curso de integración continua con *Github* y *Travis* para la automatización y despliegue continuo mediante la plataforma online Udemy.

10.3.3 Revisiones

Con respecto al desarrollo, se hicieron revisiones cruzadas entre los integrantes del *frontend* y el *backend* con el objetivo de que ambos sub-equipos sepan sobre el código la otra parte. Esto significa que integrantes del *backend* realizaron revisiones sobre el *frontend*, e integrantes del *frontend* realizaron revisiones sobre el *backend*.

10.3.4 Retrospectivas

Al final de cada sprint se realizaron reuniones en retrospectiva al mismo, tal como lo propone el evento de Scrum [93]. En estas participaban todos los integrantes del equipo y se discutían que cosas se hicieron bien durante el sprint, que cosas se hicieron mal, y cuales se podían mejorar. Las mismas permitieron un intercambio positivo entre los integrantes y ayudó a aumentar la productividad del equipo.

10.3.5 Verificaciones

La verificación del producto implica probar que el mismo cumple con los requerimientos recabados del *product backlog*. Esto fue realizado al final de cada sprint durante el *Sprint Review*, tomando las historias de usuario e *issues* terminados, y comparando con lo implementado. Durante estas instancias también se verificó el cumplimiento de los estándares de codificación y documentación definidos anteriormente.

10.3.6 Validaciones

Se realizaron distintas validaciones a lo largo del proyecto. La validación busca probar que tan bien se ajusta lo implementado a las necesidades del negocio. En una etapa inicial se le presentó distintos prototipos al cliente con el fin de validar los requerimientos del sistema. Antes de comenzar con le fase de desarrollo, el equipo automatizó la integración y despliegue continuos con el objetivo de poder mostrar a *Conecta361* las diferentes iteraciones del producto frecuentemente, y así minimizar el riesgo de desviarse de las necesidades del negocio.

Con los mismos objetivos previamente mencionados, antes de comenzar con el último *milestone* se le presentó una versión beta del producto a Mundo Trabajo, cliente de *Conecta361*. Este supo brindar aportes relevantes al producto, especialmente en aspectos de usabilidad y presentación de la información. Luego del último *milestone*,

Mundo Trabajo transicionó a la versión final del producto, la cual usa hasta el día de escritura de esta documentación.

Ilustración 67 - Validación del producto con MundoTrabajo

10.3.7 Pruebas de Software

Con pruebas se buscó maximizar la probabilidad de encontrar fallas, lograr un efecto de documentación del comportamiento esperado del *software*, y brindar seguridad al equipo a la hora de realizar cambios. Por estas razones es que a lo largo del ciclo de vida del software se realizaron distintas pruebas funcionales, específicamente unitarias y de integración, así como pruebas de carga.

Pruebas Unitarias

Se realizaron pruebas unitarias en el *backend*, que se fueron creando durante el ciclo de vida del *software*, y se fueron ejecutando con los procesos de despliegue e integración

automáticos. La herramienta utilizada fue Junit[15], y en la siguiente ilustración se pueden observar algunos resultados de estas.

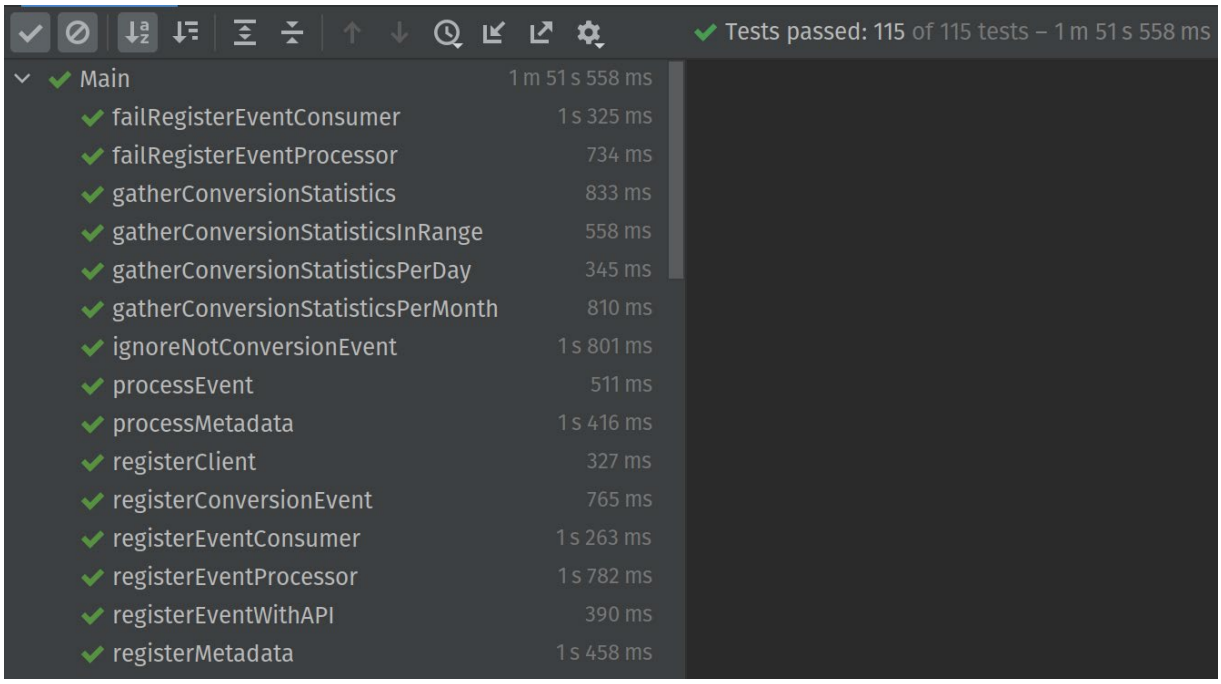


Ilustración 68 - Pruebas unitarias

Pruebas de Integración

Se realizaron pruebas de integración manuales entre el *frontend* y el *backend* al culminar cada sprint con el objetivo de verificar que las nuevas *features* funcionaran juntas correctamente. Se necesitó de ambos equipos para realizar dichas pruebas. Cabe destacar que fueron planificadas, pero no documentadas.

[15] https://junit.org/junit5/

Pruebas de Carga

En lo que refiere a la carga que debe soportar el sistema, se identificó una sola área en donde se pueden generar grandes volúmenes de datos y donde es crítico no perder información por culpa de esta. Esta área es el procesamiento de eventos de compra, o conversiones, para cada uno de los clientes de *Conecta361*, en donde se estima que podrían llegar a ocurrir hasta 1 millón de eventos por día peor caso.

Este número fue estimado tomando la cantidad de transacciones promedio por día registradas por *MercadoPago* [120]: 4.2 millones en noviembre 2020. Luego, se consideró un escenario en donde *Conecta361* aumenta drásticamente su cantidad de clientes, en donde las ventas generadas por ellos ocupan un 2.5% de las transacciones previamente mencionadas. Esto equivale a 105.000 transacciones por día. Finalmente, considerando días con carga hasta 10 veces superior a la normal como podría ser *Black Friday*, se obtienen 1 millón de transacciones en el día.

El equipo utilizó la herramienta *JMeter*[16] para simular dicho peor caso estimado de carga, y asegurar que el software pueda registrar todos los eventos sin pérdida de información.

[16] https://jmeter.apache.org/

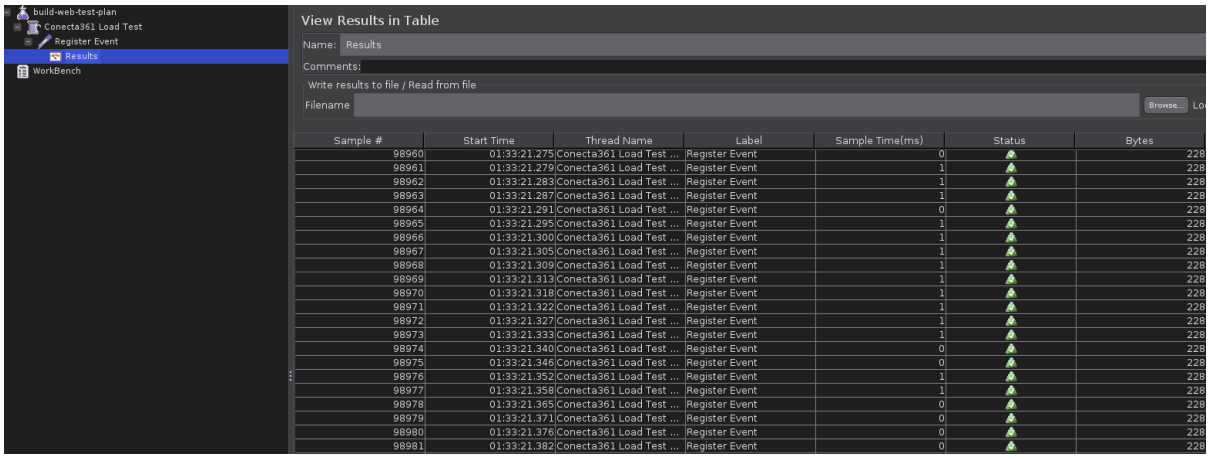


Ilustración 69 - Pruebas de carga

10.3.8 Automatización de pruebas

Como se explicó en el capítulo de Automatización del proceso de ingeniería de software: un acercamiento a DevOps, a la hora de crear una *pull-request*, se corren pruebas unitarias y herramientas de análisis de código sobre la rama de donde se origina la *pull-request.* Luego de que pasen esas pruebas y el o los revisores aprueben la *request*, se comienza un merge de la rama a develop, en donde también se corren las pruebas automatizadas y de pasar, se completa exitosamente el *merge*.

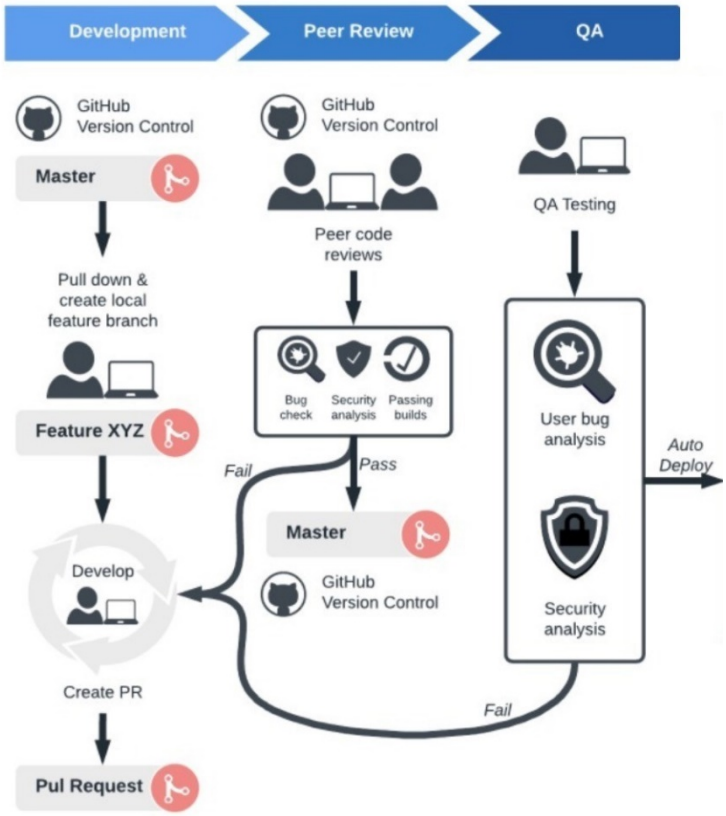


Ilustración 70 – Checks de calidad

Las herramientas de análisis de código automatizadas fueron capaces de detectar faltas en buenas prácticas e inconsistencias en el uso de estas, y analizar la deuda técnica del proyecto en cuestión. Para el *backend* se configuró SonarQube para hacer cumplir con los estilos de programación en Java propuestos por *Google* [121], y para el *frontend* se configuró es-lint para los estilos de programación en *React* propuestos por *Airbnb* [122].

10.4Gestión de incidentes

Se utilizó el tablero de *Jira* para reportar *bugs* incluyendo un título, una descripción de este, y un paso a paso de como reproducirlo. Una vez reportados los mismos eran revisados en cada *Sprint Planning*, de manera de priorizarlos y estimarlos, para subsecuentemente atacarlos durante los mismos.

Durante el sprint los *bugs* pasaron por una etapa de testing, en donde un integrante diferente al encargado de arreglarlo determinaría si la solución implementada realmente resolvía el *bug*. De verse arreglado el mismo sería integrado a la rama develop por medio del sistema de integración continua. De lo contrario la tarea asociada al *bug* era enviada nuevamente a la columna "*doing*".

10.5 Métricas de calidad

Con el fin de definir calidad y poder llevar a cabo actividades que permitieran asegurar la misma en el producto es que se definieron métricas de calidad. Estas permitieron la evaluación continua de la calidad, necesaria para poder detectar desviaciones de manera temprana, y de esta manera reducir el riesgo de crear un producto de mala calidad.

10.5.1 Calidad de código

Para lograr medir la calidad del código de la manera más cuantitativa posible se utilizaron herramientas de análisis de código estático configuradas en base a guías de calidad para el desarrollo en los lenguajes y *frameworks* utilizados como se menciona en la sección de Automatización de pruebas.

Frontend

Como se menciona previamente, para el *frontend* se utiliza *es-lint* configurado con la guía de desarrollo para *React* de *Airbnb* [122]. Este *linter* simplemente hace correcciones de estilo en forma de advertencias de manera que haya consistencia a lo largo del código, por lo que como métrica el equipo se propuso lograr que el código no contenga ninguna de ellas a la hora de finalizar el sprint. A continuación, se puede ver una lista de advertencias en el código durante el último sprint:

Ilustración 71 - *Frontend linter checks*

Backend

Para el *backend* se utilizó *SonarQube*, una herramienta más robusta, que además de hacer cumplir estilos, es capaz de detectar deuda técnica, ambas definidas en su documento de definición de métricas [123]. Resumidamente la deuda técnica es definida por *SonarQube* como el esfuerzo requerido para arreglar todas sus advertencias, y se calcula como el tiempo promedio estimado para resolver cada uno de estos por la cantidad de las advertencias en cuestión. El objetivo propuesto por el equipo fue mantener la deuda técnica por debajo de una hora.

Esta herramienta fue configurada con la guía de estilos de *Google* para *Java* [121].

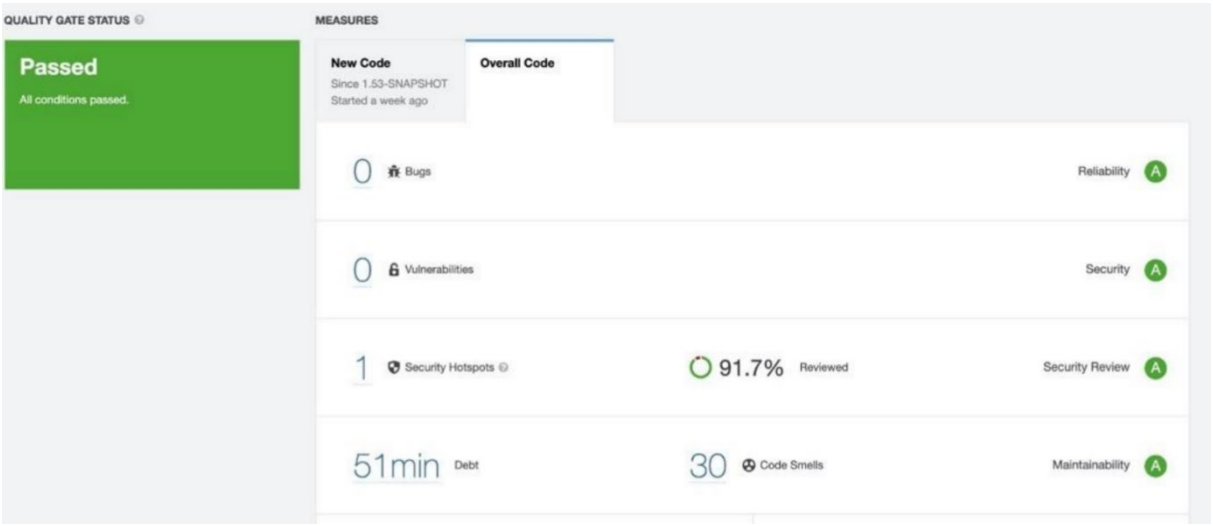


Ilustración 72 - Análisis de *SonarQube*

11. Conclusiones

En el presente capítulo se presenta el estado actual del proyecto y las conclusiones del mismo respecto a los objetivos planteados en el comienzo (ver Objetivos). A su vez, se explicarán las lecciones aprendidas y los pasos a seguir en el futuro.

11.1 Estado actual

Hoy en día el sistema de *Tracker361* se encuentra funcional y contiene todas las funcionalidades acordadas con el cliente.

La aplicación web se encuentra disponible de forma pública a través del siguiente enlace: https://app.conecta361.tk; aunque no es posible acceder a ella sin un usuario previamente registrado por el administrador de *Conecta361*.

A fecha de hoy, Mundo Trabajo es el único cliente de *Conecta361* que se encuentra utilizando la plataforma en producción. El mismo ha probado de forma paulatina algunas de las funcionalidades para comenzar a monitorear las conversiones *offline* de sus campañas publicitarias. Durante estas, se activó a *MercadoPago* como fuente de datos y a *Facebook Ads* como administrador de anuncios y procesador de conversiones.

11.2 Conclusiones del proyecto

11.2.1 Conclusiones de los objetivos académicos

Aplicar los conocimientos adquiridos durante la carrera

El equipo supo aprovechar esta oportunidad para aplicar los conocimientos adquiridos a lo largo de la carrera. Se volcaron conocimientos de distintas áreas como lo son diseño de aplicaciones, arquitectura y desarrollo de *software*. A su vez, se realizaron actividades de gestión de proyecto y calidad. A pesar de que todos los integrantes del equipo trabajan actualmente, nunca se habían enfrentado frente a la oportunidad de tener que auto gestionarse y tomar decisiones de gran relevancia. Significó para el equipo una instancia de crecimiento profesional de cara al ámbito laboral.

Aprender nuevas tecnologías y herramientas

Con el propósito de mantenerse actualizados ante las nuevas herramientas y tecnologías de la actualidad, el equipo decidió utilizar algunas de las más populares de hoy en día y que no fueron vistas a lo largo de la carrera. Se logró aprender sobre nuevos lenguajes de programación como *ReactJS*, gestionar infraestructura y servicios *cloud* con *Amazon Web Services*, utilizar *Nexus* como manejador de dependencias, *Travis CI como* herramienta de integración continua y *DataDog* como plataforma de monitoreo en la nube. Más allá de las mencionadas, el equipo aprendió muchas otras y, sobre todo, fue capaz de utilizarlas en conjunto, generando una solución moderna y vanguardista.

Producto de gran complejidad

Sin lugar a dudas el equipo cumplió el objetivo de realizar un producto sumamente desafiante y complejo. El dominio del producto es innovador, poco conocido aún en la industria y generaba gran incertidumbre por la escasez de herramientas y plataformas en el mercado. Tecnológicamente hablando, el desarrollo del producto requirió de desafíos que iban desde la creación de una infraestructura completa desde cero hasta el

despliegue a producción. También diseñando arquitecturas que cumplan con los atributos de calidad necesarios para un correcto funcionamiento, creación de API's y SDK's que permitieron que los clientes puedan conectarse más allá de la solución web, manejo de integraciones de fuente de datos y administradores de anuncios, entre otros.

11.2.2 Conclusiones de los objetivos del producto

Acortar la brecha entre las ventas offline y el marketing digital

El cumplimiento de este objetivo se basó en una única premisa: poder medir las conversiones *offline* además de las ya contempladas *online*, y así obtener el panorama completo de todas las conversiones de ambos canales (*online* y *offline*). El objetivo se logró ya que la solución permitió, por un lado, integrar las herramientas de *Facebook* y *Google Ads* como mecanismos de detección de conversiones, y por otro, obtener los datos de los compradores desde diferentes fuentes de datos. Como resultado de la combinación de ambos procesos se pudieron obtener estadísticas que muestran la verdadera rentabilidad de las campañas digitales.

Como posible mejora en el futuro vale la pena investigar en nuevos mecanismos de conversiones *offline* como flyers de descuentos, uso de códigos QR y *beacons* geográficos. Además, poder abarcar todas las plataformas utilizadas por los clientes es sumamente importante de manera de seguir aumentando la precisión del cálculo de la rentabilidad. Por esta razón, se deben poder integrarse todos los administradores de anuncios más importantes, como *TikTok Ads, YoutTube Ads* y otros que puedan aparecer con el paso del tiempo.

Agregar valor a los clientes mediante la optimización de su presupuesto publicitario

Acortar la brecha entre las ventas *offline* y el marketing digital significa poder obtener datos más precisos acerca del retorno de inversión de las campañas publicitarias. Tener en consideración estos datos dan un valor agregado al cliente para las inversiones de sus campañas futuras, optimizando aquellas que no cumplan con la rentabilidad esperada.

En menor o mayor medida, este ajuste presupuestario se traduce a un ajuste monetario y ganancia por parte del cliente.

Automatizar procesos que permitan la detección de ventas físicas producidas a través de campañas digitales

Antes de realizada la solución, los procesos de *Conecta361* se basaban en un trabajo 100% manual, y a pesar de tener consigo soluciones mediante los administradores de anuncios de *Facebook* y *Google Ads* para la detección de conversiones, la implicación humana se mantenía igual. El producto permitió la escalabilidad suficiente para que el trabajo humano no solo sea reducido, sino que eliminado casi al completo. Esto se hizo a través de la automatización de los procesos de obtención de datos y detección de conversiones realizadas mediante API's.

11.2.3 Conclusiones de los objetivos del proyecto

Autogestión del equipo

La división de roles entre los integrantes del equipo fue un punto clave para cumplir con este objetivo. El equipo conocía las fortalezas y debilidades de cada integrante permitiendo que cada uno se sienta motivado con la responsabilidad asignada. A su vez, el haber aplicado las buenas prácticas del área de ingeniería de *software*, facilitó obtener buenos resultados. El plan de gestión del proyecto y su seguimiento fue completamente realizado por el equipo, desde el plan de riesgos hasta calidad, comunicación con el cliente y seguimiento de las etapas de investigación, desarrollo y documentación.

Cumplir con el alcance acordado

El equipo logró cumplir con el alcance acordado al inicio del proyecto. Su cumplieron con casi todos los requerimientos funcionales de alta y media prioridad, así como también los principales requerimientos no funcionales. A su vez, se realizaron algunos de los

requerimientos funcionales de prioridad baja. Aquellos requerimientos que no fueron realizados no afectaron al funcionamiento *core* de la solución.

Lograr un producto que sea puesto en producción antes de la entrega final

Lograr que el producto sea productivo antes de la entrega final fue sin lugar a dudas uno de los objetivos más desafiantes y motivantes para el equipo. La decisión de construir una infraestructura en las etapas tempranas del proyecto, sumado a la automatización de las prácticas de la ingeniería de software, ayudaron a dar soporte al desarrollo y arquitectura de una solución de por sí muy compleja. Hoy en día Mundo Trabajo se encuentra utilizando el producto en producción, siendo el objetivo alcanzado.

Sin embargo, hay que destacar que el objetivo de lograr que un producto sea puesto en producción es relativo. Si bien *"Tracker361"* ya se encuentra siendo utilizado por Mundo Trabajo, este es solo un cliente y está realizando sus primeras pruebas con usuarios reales. Por tanto, para lograr cumplir realmente con el objetivo, el producto deberá ser utilizado por más clientes de manera de poder detectar los problemas en su escalabilidad. Solo el tiempo dirá si este objetivo fue cumplido de manera 100% satisfactoria.

11.3Reflexiones y lecciones aprendidas

Durante el proyecto se tomaron decisiones que no siempre resultaron ser exitosas, impactando a veces de manera negativa en el mismo. En todos los casos, el equipo tomó a estas como oportunidad de mejora y aprendizaje.

Realizar un proyecto de a cuatro integrantes durante un año entero no fue tarea sencilla. Si bien los integrantes ya habían trabajado juntos en instancias separadas, durante la carrera no era común realizar obligatorios de tal cantidad de personas. Este requirió de una buena comunicación y organización de manera de poder auto gestionarse como equipo. Haber tenido experiencia laboral ayudó a tomar decisiones que a priori podrían haber causado conflictos internos.

El equipo se comprometió con el cliente a tener un producto en producción para antes de la entrega final. Cuando los integrantes tomaron conciencia de lo que esto significaba, y que las consecuencias iban más allá de una nota en un proyecto académico, fue un golpe de realidad. Esto fue un desafío que el equipo asumió con total motivación y seriedad para cumplir con los objetivos del proyecto.

Las características del proyecto suponían un gran desafío para todos los integrantes: el dominio era desconocido, los requerimientos no estaban definidos y el cliente esperaba el apoyo de todo el equipo para la toma de decisiones. Fue muy importante realizar una fase de investigación para comprender el mundo del marketing digital y sus derivados. Esta no fue llevada a cabo fácilmente, ya que entender el dominio y el problema en cuestión costaba mucho. Como lección, el equipo se dio cuenta que las herramientas aprendidas durante la carrera no eran excluyentes para un buen desempeño profesional, y que es necesario estar contantemente aprendiendo sobre otras temáticas y áreas ajenas a la ingeniería.

Al comenzar el proyecto, el equipo no tuvo una correcta gestión del plan de riesgos, causando que ocurran problemas nunca considerados u omitidos. Como ejemplo de esto, uno de los integrantes tenía planeado la realización de una pasantía en el exterior, que, por diferentes motivos, se canceló. Sin embargo, el equipo subestimó que esto sucediese, omitiendo el riesgo en el plan. Cuando este estuvo a punto de ocurrir, el equipo entendió que se debía de dar más importancia a la gestión de riesgos y a la elaboración de planes de respuesta y contingencia, de manera de actuar rápidamente y evitar la materialización de los riesgos a futuro.

Utilizar un ciclo de vida incremental e iterativo fue una decisión acertada por el equipo, dado que las características del proyecto demandaban trabajar en intervalos de tiempo pequeños, de modo de entregar *software* frecuente al cliente y realizar cambios de

manera rápida en caso de ser necesario. La aplicación de los eventos de *Scrum* permitió validar los incrementos resultantes de cada sprint. Si bien los integrantes adquirieron experiencia estimando *user stories*, costó mucho durante los primeros sprints debido a que no se conocía la velocidad del equipo.

Otra lección a destacar es no subestimar las herramientas y tecnologías de trabajo. Se deben analizar las ventajas y desventajas de cada una de ellas. Por ejemplo: los integrantes del equipo no contaban con experiencia previa trabajando con *ReactJS* y con el correr de las semanas se dieron cuenta que la complejidad era más alta de lo que se esperaba, por lo que las tareas llevaron más tiempo de lo estipulado.

Automatizar los procesos de la ingeniería de software fue una de las decisiones más importantes y acertadas del equipo. Aplicar las prácticas de integración y entrega continua permitieron al equipo centrarse únicamente en el desarrollo de software. Además, el proyecto quedó bien perfilado para adoptar una metodología de *DevOps* si así se diese en el futuro. Fue relevante, también, tomarse el tiempo necesario para construir una buena infraestructura, de manera de minimizar el riesgo de no saber desplegar el código antes de la fecha pactada para la salida a producción.

Debido a que se trabajó sobre un proyecto real, fue fundamental asegurar la calidad del mismo en todo momento. La elaboración de un plan de calidad sirvió como referencia para planificar y llevar a cabo cada una de las actividades. Facilitó la organización debida que, para cada una de estas, se definieron roles encargados y roles participantes. El seguir estándares de codificación, ayudó a tener un código legible y fácil de mantener. A su vez, el haber definido métricas, permitió comprobar la verificación de los objetivos. Si bien monitorear estas métricas pudo resultar un proceso tedioso, fue fundamental para haber tomado mejores decisiones y aumentar la productividad del equipo a lo largo del proceso.

No obstante, cabe destacar que el plan de calidad se hizo un poco más tarde de lo esperado. Se aprendió mucho durante el proyecto, y es por esa razón que quizás hay cosas que se hicieron durante y no antes, que hubiese sido lo ideal. Por ejemplo, si bien el equipo puso mucho énfasis en la automatización de las pruebas, no todas estas se hicieron al comienzo del proyecto. De todas estas, el equipo tomó conciencia de su importancia y de su oportunidad de mejora en el futuro.

Por último, pero no menos importante, entender que no existe un proyecto perfecto y pueden surgir cambios a lo largo del mismo. Si bien hay situaciones o riesgos que se pueden predecir, muchos otros no se pueden anticipar. Como lección a esto, se debe estar preparado para los cambios o problemas que surjan y aceptarlos como tal.

11.4 Próximos pasos

Luego de la entrega académica, el equipo y el cliente tienen acordado juntarse para coordinar como continuará el vínculo entre ambas partes, aunque ambos ya han manifestado el interés de mantener el proyecto. La buena relación que se mantuvo con el equipo de Conecta361 favoreció la motivación de todos los involucrados para hacer de *Tracker361* un producto popular.

A continuación, se presentan algunas de las acciones que se pretenden tomar una vez finalizada la entrega académica del proyecto. Estas se presentan como pasos secuenciales de un *roadmap* a seguir, visualizado en la siguiente ilustración:

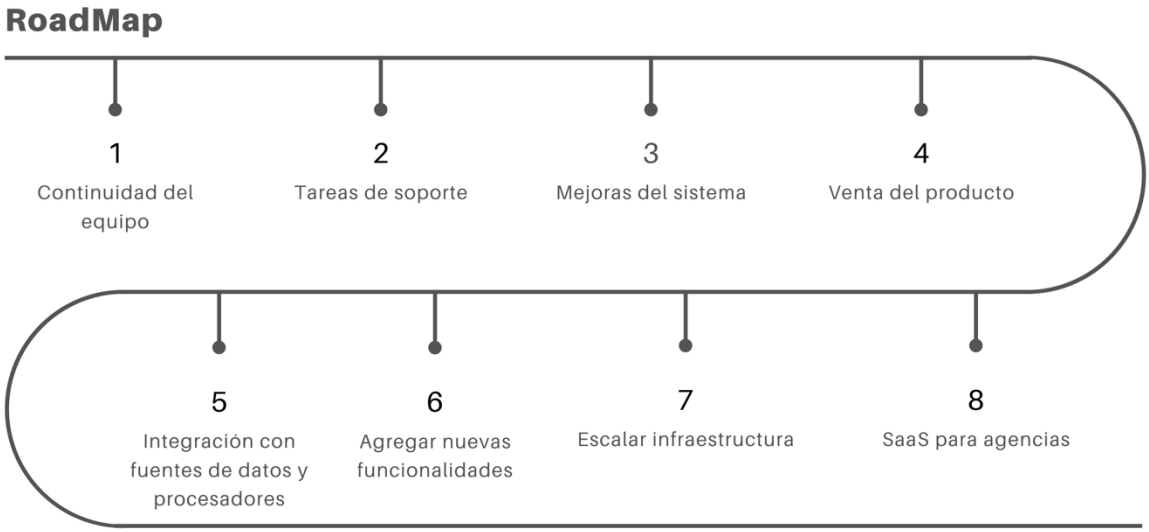


Ilustración 73 - Roadmap

1. Ultimar detalles de la continuidad del proyecto del equipo

Si bien el expreso deseo de ambas partes es la continuidad del proyecto, aún restan detalles por definir de la implementación del proceso. Los integrantes del equipo

continúan trabajando en sus empresas, por lo que dedicarle horas al mantenimiento y continuidad de la plataforma *Tracker361* supondría movimientos y ajustes en la organización privada de cada uno de estos. El equipo propondrá mantener el proyecto como un *side-project,* y en caso de ser necesario, realizar la contratación de más recursos.

2. Tareas de soporte

El equipo planea realizar tareas de soporte del sistema, evaluando el comportamiento del mismo en producción. Para esto ya tiene planeado instancias de recolección de *feedback* con *Conecta361* y Mundo Trabajo como actores activos del sistema.

- Mejoras del sistema

En base al *feedback* recolectado por Mundo Trabajo, y los errores que se encuentren en producción, se planea continuar con mejoras en el sistema y algunas tareas menores que quedaron pendientes en el *backlog.*

4. Venta del producto a clientes actuales de Conecta361

Agregar como producto del plan de ventas de *Conecta361* a *Tracker361.* Si bien hoy en día se cuenta con un único cliente, Mundo Trabajo, este se ofreció como *beta tester* y no se le cobra el servicio.

5. Integración con otras fuentes de datos y procesadores

Si bien para esta primera instancia se realizó integraciones con unas pocas fuentes de datos, se espera poder incorporar una mayor diversidad de estas para dar soporte a las herramientas y plataformas utilizadas por todos los clientes. A su vez, *Conecta361* espera poder añadir nuevos administradores de anuncios y procesadores a los ya incluidos *Facebook y Google Ads.* Dentro de los planes de esta expansión, se encuentran *TikTok Ads* y *Microsoft Ads.*

6. Agregar nuevas funcionalidades

Se buscará agregar funcionalidades que, por exceder el alcance del proyecto, no fueron incluidas en una primera instancia. Dentro de estas, se encuentran:

- Agregar nuevos mecanismos de detección de conversiones *offline*. Estos son, por ejemplo, call-*tracking, email-tracking* y *pick & collect*.
- Además de funcionar como un gestor de conversiones *offline,* permitir utilizar la plataforma como centralizador de todas las conversiones, tanto *online* como *offline.*
- Utilizar los datos y comportamientos de los compradores para crear audiencias personalizadas, y en base a estas, aplicar técnicas del *marketing* como por ejemplo el *retargeting*.

7. Escalar infraestructura

Para la realización de la entrega académica, el equipo hizo uso de los recursos provistos por los servicios *cloud* de *Amazon Web Services* como alojamiento de los componentes del sistema. Por cuestiones financieras, el equipo contrató la mínima cantidad posible de servidores para dar soporte a un correcto funcionamiento de la plataforma. Sin embargo, si el volumen de tráfico de esta aumenta en el futuro, será importante escalar de manera horizontal los servidores del sistema.

8. SaaS para otras agencias de marketing locales o internacionales

Como última instancia del *roadmap* definido, se evaluará si el producto puede ser exportado como un *SaaS* para agencias de marketing locales o internacionales, si es que así lo desea *Conecta361.* La opción ya ha sido debatida en instancias pasadas, en donde se abrió la posibilidad de realizar dicha acción, sobre todo en agencias internacionales que no sean competencia directa de *Conecta361.*

12. Referencias bibliográficas

[1] Facebook, «Información sobre el administrador de anuncios de Facebook,» [En línea]. Available: https://www.facebook.com/business/help/200000840044554?id=802745156580214. [Último acceso: 4 Febrero 2021].

[2] Cyberclick, «Campaña publicitaria,» [En línea]. Available: https://www.cyberclick.es/publicidad/campana-publicitaria. [Último acceso: 4 Febrero 2021].

[3] Cyberclick, «¿Qué son las conversiones digitales?,» [En línea]. Available: https://www.cyberclick.es/que-es/conversion-digital. [Último acceso: 4 Febrero 2021].

[4] L. R. Cid, «Sin Conversión Offline no hay Paraiso en Márketing Online,» [En línea]. Available: https://marketingwebmadrid.es/sin-conversion-offline-no-hay-paraiso-marketing-online/#Que_es_una_conversion_offline. [Último acceso: 5 Febrero 2021].

[5] G. Nardi, «¿Cuál es la diferencia entre cuenta publicitaria y administrador comercial de Facebook?,» [En línea]. Available: https://desnudandoelmarketing.com/diferencia-entre-cuenta-publicitaria-y-administrador-comercial-de-facebook. [Último acceso: 4 Febrero 2021].

[6] Antevenio, «Qué es Marketing de Atribución y Porqué es Necesario?,» [En línea]. Available: https://www.antevenio.com/blog/2017/11/que-es-marketing-de-atribucion-y-por-que-es-necesario/.

[7] 40defiebre, «¿Qué es el ROI?,» [En línea]. Available: https://www.40defiebre.com/que-es/roi. [Último acceso: 4 Febrero 2021].

[8] Wikipedia, «Web scraping,» [En línea]. Available: https://es.wikipedia.org/wiki/Web_scraping. [Último acceso: 5 Febrero 2021].

[9] inboundcycle, «Tráfico web: ¿Cómo conseguirlo?,» [En línea]. Available: https://www.inboundcycle.com/trafico-web. [Último acceso: 4 Febrero 2021].

[10] Conecta361.com, «Conecta361,» [En línea]. Available: https://www.conecta361.com.

[11] Conecta361.com, «Conecta 361 - Agencia de Marketing Digital en Uruguay,» [En línea]. Available: https://conecta361.com/. [Último acceso: 23 Diciembre 2020].

[12] U. O. Uruguay, «CIE - Oincs,» [En línea]. Available: https://cie.ort.edu.uy/43006/18/oincs.html. [Último acceso: 2018 Febrero 2021].

[13] S. Romeo, «Social Media Marketing: Qué es y cómo implementarlo,» [En línea]. Available: https://blog.fromdoppler.com/social-media-marketing-que-es-y-como-implementarlo-exitosamente/. [Último acceso: 11 Febrero 2021].

[14] «Linkedin - Marcelo Wilkorwsky,» [En línea]. Available: https://www.linkedin.com/in/marcelo-wilkorwsky/. [Último acceso: 18 Febrero 2021].

[15] C. Academy, «¿Qué es marketing? Definición, ventajas y cómo funciona,» [En línea]. Available: https://www.cyberclick.es/marketing#que-es-marketing. [Último acceso: 28 Febrero 2021].

[16] P. Drucker, «¿Qué es el Marketing?,» [En línea]. Available: https://igestion20.com/%C2%BFque-es-el-marketing/. [Último acceso: 28 Febrero 2021].

[17] Business2community.com, «Business2community - Digital Marketing & Traditional Marketing infoographic,» [En línea]. Available: https://www.business2community.com/infographics/digital-marketing-vs-traditional-marketing-infographic-02252932. [Último acceso: 28 Diciembre 2020].

[18] M. BLANCO, «Marketing tradicional: qué es y ejemplos principales,» [En línea]. Available: https://marketingblanco.com/marketing-tradicional-que-es-y-ejemplos-principales/. [Último acceso: 24 Enero 2021].

[19] R. Content, «¿Qué es la publicidad digital?,» [En línea]. Available: https://rockcontent.com/es/blog/publicidad-digital/. [Último acceso: 9 Enero 2021].

[20] Mariella. [En línea]. Available: https://www.datatrust.pe/marketing-digital/7-diferencias-marketing-digital-vs-marketing-tradicional/. [Último acceso: 15 Febrero 2021].

[21] Cyberclick.es, «Cyberclick - Tipos de Marketing,» [En línea]. Available: https://www.cyberclick.es/marketing#tipos-de-marketing.

[22] Ryte.com, «Ryte - Marketing de Redes Sociales,» [En línea]. Available: https://es.ryte.com/wiki/Marketing_de_Redes_Sociales.

[23] Conversific.com, «www.conversific.com,» [En línea]. Available: State Of Marketing Report Post Covid 2020. [Último acceso: 11 Marzo 2021].

[24] HubSpot, «HubSpot - Not another state of marketing report 2020,» [En línea]. Available: https://cdn2.hubspot.net/hubfs/53/tools/state-of-marketing/PDFs/Not%20Another%20State%20of%20Marketing%20Report%20-%20Web%20Version.pdf. [Último acceso: 11 Marzo 2021].

[25] Tendenciasdigitales.com, «Tendencias Digitales - La metodologia ACT del mercadeo en internet,» [En línea]. Available: http://tendenciasdigitales.com/la-metodologia-act-del-mercadeo-en-internet/. [Último acceso: 27 Diciembre 2020].

[26] 40defiebre, «¿Qué es el SEO y por qué lo necesito?,» [En línea]. Available: https://www.40defiebre.com/guia-seo/que-es-seo-por-que-necesito. [Último acceso: 6 Febrero 2021].

[27] 40defiebre, «¿Qué es el SEM?,» [En línea]. Available: https://www.40defiebre.com/que-es/sem. [Último acceso: 6 Febrero 2021].

[28] J. C. M. Llano, «QUÉ ES EL MARKETING DIGITAL, SU IMPORTANCIA Y PRINCIPALES ESTRATEGIAS,» [En línea]. Available: https://www.juancmejia.com/marketing-digital/que-es-el-marketing-digital-su-importancia-y-principales-estrategias/. [Último acceso: 8 Enero 2021].

[29] d. comunicación, «Qué es Facebook Ads y cómo funciona,» [En línea]. Available: https://www.difusion.org/facebook-ads-funciona/. [Último acceso: 11 Febrero 2021].

[30] «Ranking mundial de redes sociales por numero de usuarios,» [En línea]. Available: https://es.statista.com/estadisticas/600712/ranking-mundial-de-redes-sociales-por-numero-de-usuarios/. [Último acceso: 16 Marzo 2021].

[31] Facebook.com, «Tools - Ads Manager,» [En línea]. Available: https://www.facebook.com/business/tools/ads-manager. [Último acceso: 16 Marzo 2021].

[32] bluecaribu, «Google Ads: qué es y cómo funciona para una empresa,» [En línea]. Available: https://www.bluecaribu.com/las-3-unicas-razones-por-las-que-google-adwords-no-le-funciona-a-algunas-empresas. [Último acceso: 12 Febrero 2021].

[33] Oleoshop.com, «Oleshop - Que es el retail,» [En línea]. Available: https://www.oleoshop.com/blog/que-es-retail. [Último acceso: 1 Febrero 2021].

[34] Wikipedia, «Retail,» [En línea]. Available: https://es.wikipedia.org/wiki/Retail. [Último acceso: 9 Febrero 2021].

[35] M. Nicolás, «¿Qué es retail? Definición y características,» [En línea]. Available: https://www.oleoshop.com/blog/que-es-retail. [Último acceso: 9 Febrero 2021].

[36] Wikipedia, «World Wide Web,» [En línea]. Available: https://es.wikipedia.org/wiki/World_Wide_Web. [Último acceso: 10 Febrero 2021].

[37] pwc, «Retailers and the Age of Disruption,» [En línea]. Available: https://www.pwc.com/gx/en/retail-consumer/retail-consumer-publications/global-multi-channel-consumer-survey/assets/pdf/total-retail-2015.pdf. [Último acceso: 10 Febrero 2021].

[38] D. Rekuc, «Study: Why 92% of Retail Purchases Still Happen Offline,» [En línea]. Available: https://ripen.com/blog/ecommerce_survey. [Último acceso: 9 Febrero 2021].

[39] F. Ali, «US ecommerce grows 44.0% in 2020,» [En línea]. Available: https://www.digitalcommerce360.com/article/us-ecommerce-sales/. [Último acceso: 10 Febrero 2021].

[40] hubspot, «The Ultimate List of Marketing Statistics for 2020,» [En línea]. Available: https://www.hubspot.com/marketing-statistics. [Último acceso: 10 Febrero 2021].

[41] A. Orendorff, «O2O Commerce: Conquering Online-to-Offline Retail's Trillion Dollar Opportunity,» [En línea]. Available: https://www.shopify.com/enterprise/o2o-online-to-offline-commerce. [Último acceso: 10 Febrero 2021].

[42] bazaarvoice, «How online reviews influence offline sales,» [En línea]. Available: https://www.bazaarvoice.com/resources/the-robo-economy-infographic/. [Último acceso: 10 Febrero 2021].

[43] J. Wanamaker, «Offline Conversion Tracking, The Missing Metric,» [En línea]. Available: https://searchengineland.com/offline-conversion-tracking-the-missing-metric-12467. [Último acceso: 01 03 2021].

[44] adglow, «Cómo monitorizar las conversiones offline de Facebook Ads,» [En línea]. Available: https://www.adglow.com/es-blog/c%C3%B3mo-monitorizar-las-conversiones-offline-de-facebook-ads. [Último acceso: 12 Febrero 2021].

[45] M. Wilkorwsky, «IQLATINO,» [En línea]. Available: https://iqlatino.org/author/marcelo-wilkorwsky/. [Último acceso: 5 Noviembre 2020].

[46] Agilealliance.org, «Agilealliance - What is Agile?,» [En línea]. Available: ttps://www.agilealliance.org/agile101/.

[47] M. Fowler, «Martin Fowler - Writing The Agile Manifesto,» [En línea]. Available: https://martinfowler.com/articles/agileStory.html. [Último acceso: 27 Octubre 2020].

[48] Kanbanize, «Qué es Kanban: Definición, Características y Ventajas,» [En línea]. Available: https://kanbanize.com/es/recursos-de-kanban/primeros-pasos/que-es-kanban. [Último acceso: 28 02 2021].

[49] Scrum.org, «What is Scrum?,» [En línea]. Available: https://www.scrum.org/resources/what-is-scrum. [Último acceso: 28 Febrero 2021].

[50] Wikipedia.com, «Wikipedia - Manifiesto Ágil,» [En línea]. Available: https://es.wikipedia.org/wiki/Manifiesto_%C3%A1gil. [Último acceso: 12 Marzo 2021].

[51] Wikipedia, «Ingeniería inversa,» [En línea]. Available: https://es.wikipedia.org/wiki/Ingenier%C3%ADa_inversa. [Último acceso: 13 Febrero 2021].

[52] T. Tsumaki y T. Tamai , «A Framework for Matching Requirements Engineering Techniques to Project Characteristics and Situation Changes,» *The University of Tokyo,* vol. 1, nº Requirements Engineering Techniques, p. 15, 2005.

[53] V. B. Training, «Valera Business Training - Piramide del aprendizaje de William Glasser,» [En línea]. Available: https://valerabusinesstraining.com/piramide-del-aprendizaje-de-william-glasser/. [Último acceso: 4 Marzo 2021].

[54] a. academy, «Definition of Spike,» [En línea]. Available: https://www.agile-academy.com/en/agile-dictionary/spike/. [Último acceso: 13 Febrero 2021].

[55] M. REHKOPF, «Epics, historias, temas e iniciativas,» [En línea]. Available: https://www.atlassian.com/es/agile/project-management/epics-stories-themes. [Último acceso: 13 Febrero 2021].

[56] S. Casanova, «Cómo definir criterios de aceptación,» [En línea]. Available: https://samuelcasanova.com/2017/11/criterios-de-aceptacion/. [Último acceso: 14 Febrero 2021].

[57] Atlassian.com, «Atlassian - User stories,» [En línea]. Available: https://www.atlassian.com/agile/project-management/user-stories.

[58] Santi, «Wireframes: Que son y como crearlos,» [En línea]. Available: https://webdesdecero.com/wireframes-que-son-y-como-crearlos/. [Último acceso: 12 Febrero 2021].

[59] L. Bass, P. Clements y R. Kazman, «Interoperability,» de *Software Architecture in Practice* , 3er ed., Boston, Addison-Wesley, 2012, p. 121.

[60] L. Bass, P. Clements y R. Kazman, «Modifiability,» de *Software Architecture In Practice* , 3er ed., Boston, Addison-Wesley, 2012, p. 117.

[61] L. Bass, P. Clements y R. Kazman, «Availability,» de *Software Architecture in Practice* , 3er ed., Boston, Addison-Wesley, 2012, p. 97.

[62] L. Bass, P. Clements y R. Kazman, «Security,» de *Software Architecure In Practice* , 3er ed., Boston, Addison-Wesley, 2012, p. 148.

[63] C. P. K. R. Bass Len, «Testeability,» de *Software Architecture in Practice*, 3er ed., Boston, Addison-Wesley, 2012, p. 182.

[64] L. Bass, P. Clements y R. Kazman, «Performance,» de *Software Architecture in Practice* , 3er ed., Boston, Addison-Wesley, 2012, p. 152.

[65] L. Bass, P. Clements y R. Kazman, «Usability,» de *Software Architecture in Practice*, 3er ed., Boston, Addison-Wesley, 2012, p. 198.

[66] L. Bass, P. Clements y R. Kazman, «Architecture and Requirements,» de *Software Architecture In Practice*, 3er ed., Boston, Addison-Wesley, 2012, p. 196.

[67] GuidanceShare.com, «Chapter 7 - Quality Attributes,» [En línea]. Available: http://www.guidanceshare.com/wiki/Application_Architecture_Guide_-_Chapter_7_-_Quality_Attributes#Interoperability. [Último acceso: 26 Enero 2021].

[68] Redhat.com, «Redhat - What is a rest api,» [En línea]. Available: https://www.redhat.com/es/topics/api/what-is-a-rest-api. [Último acceso: 18 02 2021].

[69] bbvaapimarket.com, «api rest que es y cuales son sus ventajas en el desarrollo de proyectos,» [En línea]. Available: https://www.bbvaapimarket.com/es/mundo-api/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos/. [Último acceso: 24 01 2021].

[70] L. Armentano, «El Baul Del Programador - Buenas practicas para el diseno de una api restful programatica,» [En línea]. Available:

https://elbauldelprogramador.com/buenas-practicas-para-el-diseno-de-una-api-restful-pragmatica/. [Último acceso: 16 11 2021].

[71] 1. F. App, «The Twelve factor app,» [En línea]. Available: https://12factor.net/es/config. [Último acceso: 28 Febrero 2021].

[72] grapheverywhere.com, «Graph Everywhere - NoSQL vs SQL,» [En línea]. Available: https://www.grapheverywhere.com/nosql-vs-sql/. [Último acceso: 02 Marzo 2021].

[73] Amazon, «Amazon DynamoDB,» [En línea]. Available: https://aws.amazon.com/es/dynamodb/. [Último acceso: 28 Febrero 2021].

[74] Datadog, «About Datadog,» [En línea]. Available: https://www.datadoghq.com/about/leadership/. [Último acceso: 28 Febrero 2021].

[75] spring, «spring,» 8 Febrero 2021. [En línea]. Available: https://spring.io/.

[76] L. Bass, P. Clements y R. Kazman, «Testeability,» de *Software Architecture in Practice*, 3er ed., Boston, Addison-Wesley, 2012, p. 187.

[77] Neoteric, «Single-page application vs. multiple-page application,» [En línea]. Available: https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58. [Último acceso: 5 Marzo 2021].

[78] Amazon.com, «Implementing priority queue with Amazon DynamoDB,» [En línea]. Available: https://aws.amazon.com/blogs/database/implementing-priority-queueing-with-amazon-dynamodb/.

[79] Google, «Material Design,» [En línea]. Available: https://material.io/.

[80] Google, «Material UI,» [En línea]. Available: https://material.io/.

[81] Toptal, «Angular vs React for Web Development,» Toptal, [En línea]. Available: https://www.toptal.com/front-end/angular-vs-react-for-web-development.

[82] M. Pisabarro, «What's with Typescript,» Light-it, [En línea]. Available: https://lightit.io/blog/whats-with-typescript/.

[83] Simform, «Compute Pricing Comparison,» [En línea]. Available: https://www.simform.com/compute-pricing-comparison-aws-azure-googlecloud/.

[84] Amazon, «AWS RDS,» [En línea]. Available: https://docs.aws.amazon.com/es_es/AmazonRDS/latest/UserGuide/Welcome.html. [Último acceso: 26 Febrero 2021].

[85] hostingpedia, «MySQL,» [En línea]. Available: https://hostingpedia.net/mysql.html. [Último acceso: 27 Febrero 2021].

[86] Amazon, «¿Qué es Amazon S3?,» [En línea]. Available: https://docs.aws.amazon.com/es_es/AmazonS3/latest/dev/Welcome.html. [Último acceso: 28 Febrero 2021].

[87] Amazon, «Amazon EC2,» [En línea]. Available: https://aws.amazon.com/es/ec2/?ec2-whats-new.sort-by=item.additionalFields.postDateTime&ec2-whats-new.sort-order=desc. [Último acceso: 28 Febrero 2021].

[88] J. Garzas, «¿Y para qué sirve Nexus o Artifactory?,» [En línea]. Available: https://www.javiergarzas.com/2014/08/nexus-artifactory-10-min.html. [Último acceso: 27 Febrero 2021].

[89] Amazon, «AWS Certificate Manager,» [En línea]. Available: https://aws.amazon.com/certificate-manager/. [Último acceso: 28 Febrero 2021].

[90] RelentlessHosting, «Why Domain Names Are Important,» [En línea]. Available: https://www.relentlesshosting.com.au/why-domain-names-are-important/. [Último acceso: 27 Febrero 2021].

[91] Atlassian, «Jira,» [En línea]. Available: https://www.atlassian.com/software/jira. [Último acceso: 28 Febrero 2021].

[92] K. S. &. J. Sutherland, «La Guía Scrum,» [En línea]. Available: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Spanish-European.pdf. [Último acceso: 10 Febrero 2021].

[93] S. Guides, «The 2020 Scrum Guide,» [En línea]. Available: https://scrumguides.org/scrum-guide.html. [Último acceso: 24 Febrero 2021].

[94] M. Goat, «Planning Poker,» [En línea]. Available: https://www.mountaingoatsoftware.com/agile/planning-poker#:~:text=Planning%20Poker%C2%AE%20is%20a,or%20any%20other%20estimating%20unit.. [Último acceso: 25 Febrero 2021].

[95] M. Cohn, Agile Estimating and Planning, Prentice Hall, 2005.

[96] P. M. Institute, Pennsylvania, 2004.

[97] D. N. Forsgren , D. D. Smith, J. Humble y J. Frazelle, «State of DevOps 2019,» [En línea]. Available: https://services.google.com/fh/files/misc/state-of-devops-2019.pdf.

[98] Aws.amazon.com, «Amazon - What is DevOps,» [En línea]. Available: https://aws.amazon.com/es/devops/what-is-devops/. [Último acceso: 10 Enero 2021].

[99] «Tribalyte - Blog Introduccion DevOps,» [En línea]. Available: https://tech.tribalyte.eu/blog-introduccion-devops#. [Último acceso: 10 Enero 2021].

[100] Y. W. Wasna y P. Debois, «Agile Infraestracture Agile 2008,» [En línea]. Available: http://www.jedi.be/presentations/agile-infrastructure-agile-2008.pdf.

[101] «A tu Servicio - Integración contnua CI,» [En línea]. Available: http://www.atuservicio.net/integracion-continua-ci/.

[102] JakobTheDev, «Medium - What is DevOps,» [En línea]. Available: https://medium.com/taptuit/what-is-devops-fb3d044ef659.

[103] «Azure - What is DevOps Culture,» [En línea]. Available: https://azure.microsoft.com/es-es/overview/what-is-devops/#culture.

[104] Amazon.com, «AWS - Continuous Integration,» [En línea]. Available: https://aws.amazon.com/es/devops/continuous-integration/. [Último acceso: 14 Marzo 2021].

[105] Wikipedia.com, «Wikipedia - SonarQube,» [En línea]. Available: https://es.wikipedia.org/wiki/SonarQube.

[106] Wikipedia.com, «Wikipedia - JUnit,» [En línea]. Available: https://es.wikipedia.org/wiki/JUnit.

[107] Wikipedia, «Revisión por pares,» [En línea]. Available: https://es.wikipedia.org/wiki/Revisi%C3%B3n_por_pares. [Último acceso: 26 Febrero 2021].

[108] Microsoft, «Prácticas de DevOps,» [En línea]. Available: https://azure.microsoft.com/es-es/overview/what-is-devops/#practices. [Último acceso: 26 Febrero 2021].

[109] Atlassian, «Qué es Git,» [En línea]. Available: https://www.atlassian.com/es/git/tutorials/what-is-git. [Último acceso: 10 Febrero 2021].

[110] ECURED, «Gestión de configuración,» [En línea]. Available: https://www.ecured.cu/Gesti%C3%B3n_de_configuraci%C3%B3n. [Último acceso: 25 Febrero 2021].

[111] Trendmicro.com, «TrendMicro - ¿Qué es la infraestructura como código?,» [En línea]. Available: https://www.trendmicro.com/es_es/what-is/cloud-security/infrastructure-as-code.html.

[112] L. F. G. Perez, «Infraestructura Como Código,» 29 5 2019. [En línea]. Available: https://codigofacilito.com/articulos/infraestructura-como-codigo.

[113] «Ilimit - Infraestructura como código,» [En línea]. Available: https://www.ilimit.com/blog/infraestructura-como-codigo-terraform-ansible/.

[114] Hpe.com, «Hpe - ¿QUÉ ES LA INFRAESTRUCTURA COMO CÓDIGO?,» [En línea]. Available: https://www.hpe.com/lamerica/es/what-is/infrastructure-as-code.html.

[115] sonatype, «nexus repository oss,» [En línea]. Available: https://www.sonatype.com/nexus/repository-oss?topnav=true. [Último acceso: 4 Marzo 2021].

[116] J. M. Juran, Quality-control handbook, New York: McGraw-Hill, 1988.

[117] M. Pisabarro, «How to Run an Effective Code Audit,» 14 12 2020. [En línea]. Available: https://lightit.io/blog/how-to-run-an-effective-code-audit-includes-code-report-template/.

[118] React, «Tutorial: Intro to React,» [En línea]. Available: https://reactjs.org/tutorial/tutorial.html. [Último acceso: 10 Febrero 2021].

[119] Baeldung, «Spring Core Annotations,» [En línea]. Available: https://www.baeldung.com/spring-core-annotations. [Último acceso: 5 Febrero 2021].

[120] M. Libre, «Noticias Mercado Libre,» 4 Noviembre 2020. [En línea]. Available: https://ideas.mercadolibre.com/ar/noticias/historia-de-mercado-libre/.

[121] Google, «Google Java Style Guide,» [En línea]. Available: https://google.github.io/styleguide/javaguide.html.

[122] Aribnb, «Airbnb React/JSX Style Guide,» [En línea]. Available: https://airbnb.io/javascript/react/.

[123] SonarQube, «SonarQube Metric Definitions,» SonarQube, [En línea]. Available: https://docs.sonarqube.org/latest/user-guide/metric-definitions/.

[124] Facebook.com, «Facebook Ads - Prácticas recomendadas para los datos de eventos offline,» [En línea]. Available:

https://www.facebook.com/business/help/1798506233494677?id=565900110447546. [Último acceso: 12 Marzo 2021].

[125] Facebook.com, «Facebook - Subir datos de eventos offline,» [En línea]. Available: https://www.facebook.com/business/help/155437961572700?id=565900110447546.

[126] «Jira Software,» Jira Software, [En línea]. Available: https://www.atlassian.com/es/software/jira/guides/getting-started/overview. [Último acceso: 13 March 2021].

[127] Capterra, «¿Qué es LeadsBridge?,» [En línea]. Available: https://www.capterra.es/software/152361/leadsbridge. [Último acceso: 2 Marzo 2021].

[128] Segment, «What is Segment?,» [En línea]. Available: https://segment.com/docs/guides/. [Último acceso: 2 Marzo 2021].

[129] Inconcertcc.com, «Innconcert,» [En línea]. Available: https://www.inconcertcc.com/es/.

[130] Apiumhub.com, «Ventajas de Jenkins,» [En línea]. Available: https://apiumhub.com/es/tech-blog-barcelona/ventajas-de-jenkins/.

[131] Atlassian, «Gitflow Workflow,» [En línea]. Available: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow. [Último acceso: 28 Febrero 2021].

[132] Udemy, «Integración Continua con GitHub Education y Travis,» [En línea]. Available: https://www.udemy.com/course/integracion-continua-con-github-education-y-travis/. [Último acceso: 6 Febrero 2021].

[133] A. R. Mesa, «Qué es un Sprint de Scrum,» [En línea]. Available: https://openwebinars.net/blog/que-es-un-sprint-scrum/. [Último acceso: 26 Febrero 2021].

[134] A. JMeter, «Apache JMeter,» [En línea]. Available: https://jmeter.apache.org/. [Último acceso: 24 Febrero 2021].

[135] Appypie.com, «Appypie - Online vs Offline Advertising,» [En línea]. Available: https://www.appypie.com/online-vs-offline-advertising .

[136] SHERMAN, «Digital Marketing vs Traditional Marketing: Which Produces Better ROI?,» [En línea]. Available: https://www.lyfemarketing.com/blog/digital-marketing-vs-traditional-marketing/. [Último acceso: 10 Febrero 2021].

[137] «Facebook - Solución de problemas con la subida del archivo de eventos offline,» [En línea]. Available: https://www.facebook.com/business/help/184554988773121. [Último acceso: 9 Marzo 2021].

[138] Wikipedia, «TapClicks,» [En línea]. Available: https://en.wikipedia.org/wiki/TapClicks. [Último acceso: 2 Marzo 2021].

[139] LyfeMarketing.com, «LiyfeMarketing - Digital Marketing vs Traditional Marketing,» [En línea]. Available: https://www.lyfemarketing.com/blog/digital-marketing-vs-traditional-marketing/. [Último acceso: 23 Diciembre 2020].

[140] «Netoattack - Marketer,» [En línea]. Available: https://neoattack.com/neowiki/marketer/. [Último acceso: 1 Marzo 2021].

[141] Atlassian.com, «Epics Stories Themes,» [En línea]. Available: https://www.atlassian.com/es/agile/project-management/epics-stories-themes. [Último acceso: 14 Marzo 2021].

[142] Udelar.edu.uy, «Udelar - CI/CD,» [En línea]. Available: https://git.posgrados.udelar.edu.uy/help/ci/README.md. [Último acceso: 12 Marzo 2021].

[143] M. Cohn, «Agile Needs to Be Both Iterative and Incremental,» [En línea]. Available: https://www.mountaingoatsoftware.com/blog/agile-needs-to-be-both-iterative-and-incremental. [Último acceso: 27 Febrero 2021].

[144] «Atlassian - Gitflow Workflow,» [En línea]. Available: https://www.atlassian.com/es/git/tutorials/comparing-workflows/gitflow-workflow. [Último acceso: 14 Marzo 2021].

13. Anexos

13.1 Archivos de ventas de clientes de Conecta361

La obtención de los archivos de ventas de los clientes por parte de *Conecta361* se hizo de diversas formas y por diferentes medios. Los archivos fueron enviados por mail, *WhatsApp* y otros. A su vez, los formatos dependen del tipo de programa que se maneje internamente en la empresa, que puede variar desde *CRM's* locales a *cloud,* sistemas de facturación electrónica, puntos de venta online y cualquier otro tipo de almacenamiento personalizado.

Solicitud de archivos por email

Como ejemplo de ilustración de la dificultad que conllevaba recolectar los diferentes archivos de los clientes, se puede observar en la siguiente imagen la solicitud de archivo de *Conecta361* hacina Mundo Trabajo.

Ilustración 74 - Solicitud de archivo de *Facebook* vía *email*

Formatos de archivos recibidos

Por otro lado, en las siguientes ilustraciones se pueden visualizar diferentes plataformas y formatos de archivo que los clientes manejaban, generando dificultad a la hora de normalizar estos.

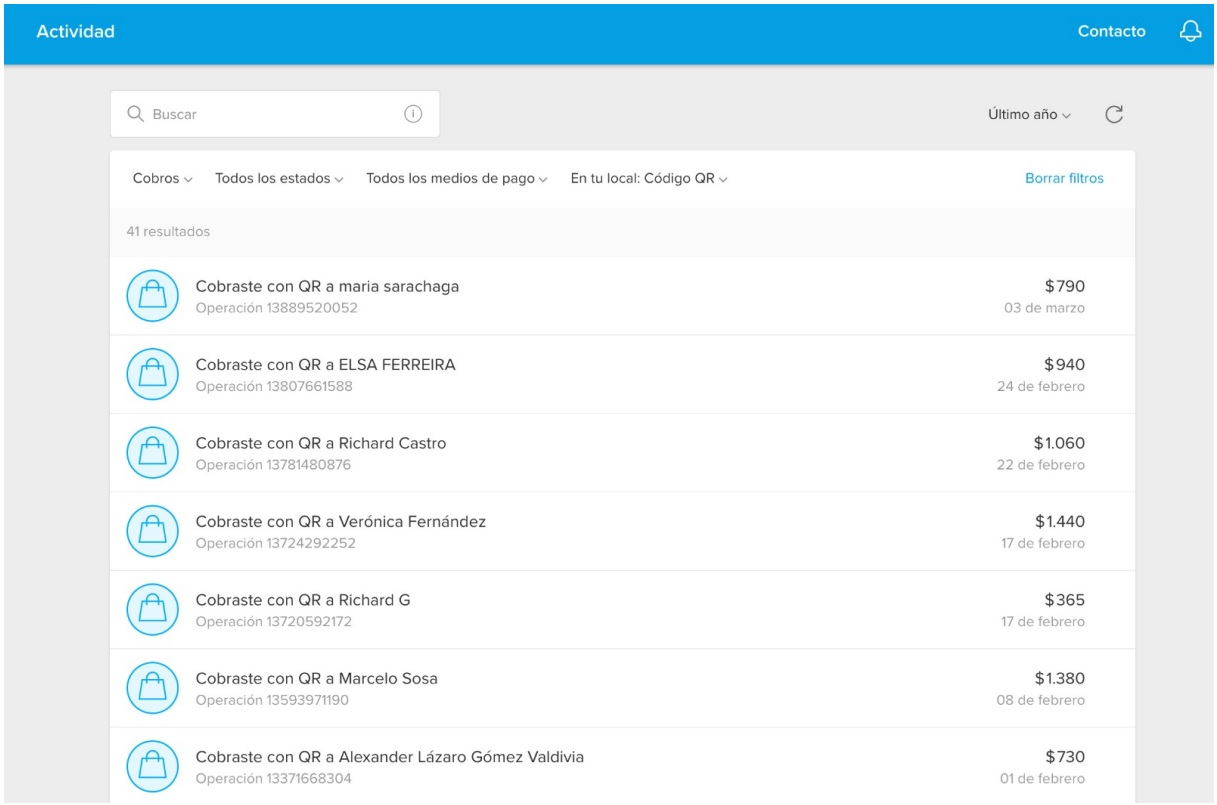


Ilustración 75 - Reporte de ventas de *MercadoPago*

A	B	C	D	E	F	G	H	I	J
EXTERNAL_REFERENC	SOURCE_ID	USER_ID	PAYMENT_METHOD	PAYMENT_METHOD	SITE	TRANSACTION_TYPE	TRANSACTION_AMOUNT	TRANSACTION_CURRENCY	TRANSACTION_DATE
4423185808	14014551812	284478807	credit_card	oca	MLU	SETTLEMENT	556.00	UYU	2021-03-11T22:10:51.000-04:00
4422951639	14013079728	284478807	credit_card	visa	MLU	SETTLEMENT	4190.00	UYU	2021-03-11T20:20:56.000-04:00
4422897332	14012720098	284478807	available_money	available_money	MLU	SETTLEMENT	1070.00	UYU	2021-03-11T19:57:38.000-04:00
4422891865	14012662343	284478807	credit_card	visa	MLU	SETTLEMENT	956.00	UYU	2021-03-11T19:55:33.000-04:00
4422682997	14011055382	284478807	credit_card	oca	MLU	SETTLEMENT	356.00	UYU	2021-03-11T18:24:52.000-04:00
4422649836	14010770455	284478807	credit_card	visa	MLU	SETTLEMENT	2686.00	UYU	2021-03-11T18:09:47.000-04:00
4422620273	14010516822	284478807	credit_card	visa	MLU	SETTLEMENT	806.00	UYU	2021-03-11T17:55:25.000-04:00
4422613920	14010495730	284478807	credit_card	visa	MLU	SETTLEMENT	1089.00	UYU	2021-03-11T17:53:25.000-04:00
4421833382	14005567813	284478807	ticket	redpagos	MLU	SETTLEMENT	1156.00	UYU	2021-03-11T12:31:08.000-04:00
WC-54041	14009633400	284478807	credit_card	oca	MLU	SETTLEMENT	42500.00	UYU	2021-03-11T17:01:47.000-04:00

Ilustración 76 - Archivo de ventas en *Excel*

13.2 Formato y especificación de campos de *Facebook Ads*

El formato y especificación de los campos principales para la subida de múltiples eventos por *Facebok Ads* es el siguiente:

Tipo de clave, única y múltiple	Nombre de clave
Direcciones de correo electrónico	email
Números de teléfono	phone
Sexo	gen
Fecha de nacimiento. Formato AAAA	doby
Fecha de nacimiento. Formato MM	dobm
Fecha de nacimiento. Formato DD	dobd
Apellido	ln
Nombre	fn
Inicial del nombre	fi
Ciudad	ct
Estados de los EE. UU.	st
Códigos postales	zip
País	country

Tabla 17 - Especifiación y formato de campos de *Facebook Ads* [124]

13.3 Flujo de subida de archivo manual en *Facebook Ads*

El flujo de subida de archivo de eventos a través del administrador de anuncios de *Facebook Ads* parte del cliente se detalla a continuación:

En primer lugar, *Conecta361* debe dirigirse al administrador de eventos y seleccionar la cuenta publicitaria a utilizar. Cabe recordar que cada cliente tiene su propia cuenta publicitaria, por lo que subir un archivo en una cuenta equivocada puede producir imprecisiones en el resultado de las conversiones y arrojar estadísticas incorrectas a sus clientes.

Una vez dentro de la cuenta publicitaria seleccionada, el administrador solicitará subir los eventos *offline.* En la siguiente pantalla se puede visualizar los pasos para la importación del archivo [125].

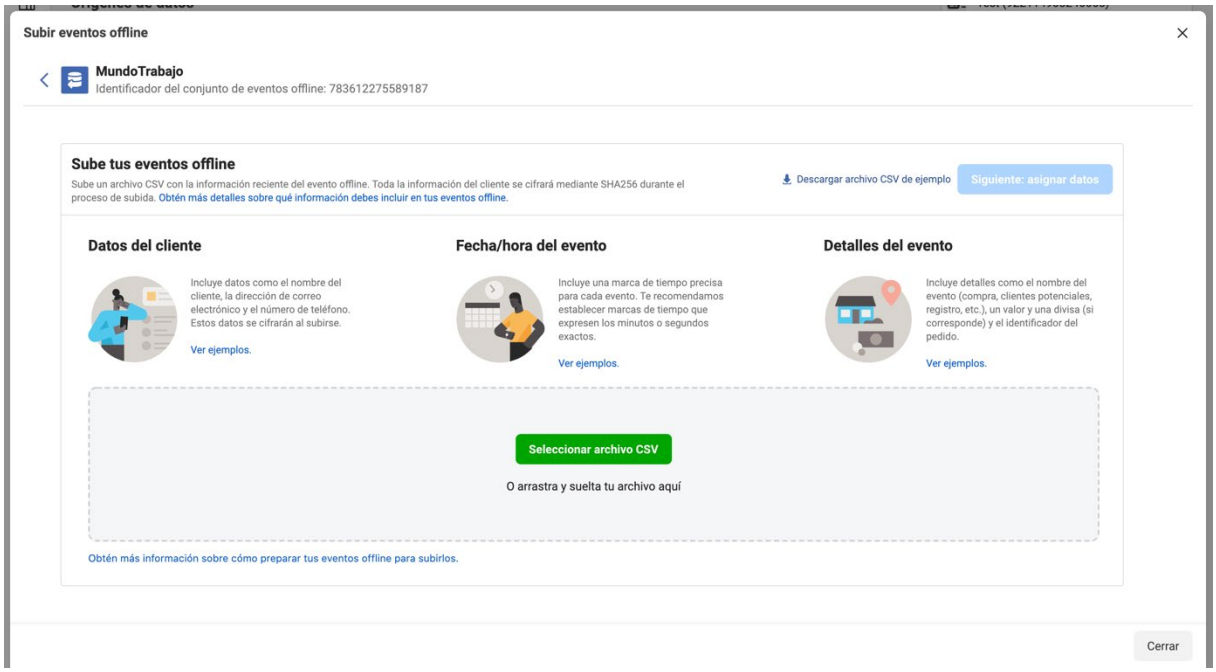


Ilustración 77 - Pantalla de subida de archivo de *Facebook Ads*

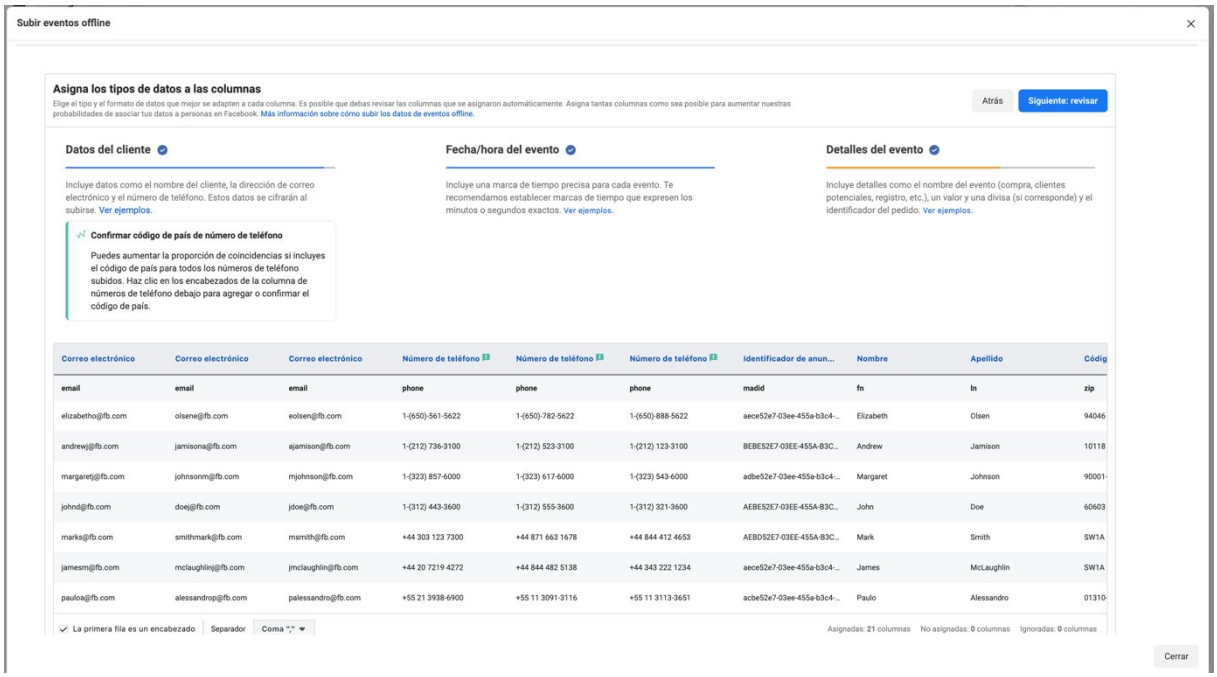


Ilustración 78 - Pantalla de asignación de campos de *Facebook Ads*

El archivo debe ser CSV y contener el formato especificado en el Anexo 13.2. Una vez arrastrado al recuadro solicitado, aparecerá la siguiente ilustración:

En esta, *Facebook* analiza una muestra del archivo para identificar el tipo de datos de cada columna. Se debe asegurar que todos los campos estén bien puestos, y verificar que la asignación de eventos y tipo de datos sea la correcta.

Al continuar al siguiente paso, se deberán resolver los errores o advertencias que aparecen cuando faltan datos, están mal asignados o tienen un formato incorrecto. A continuación, se visualizan la cantidad de registros listos para subir, la proporción aproximada de coincidencias y cualquier advertencia adicional.

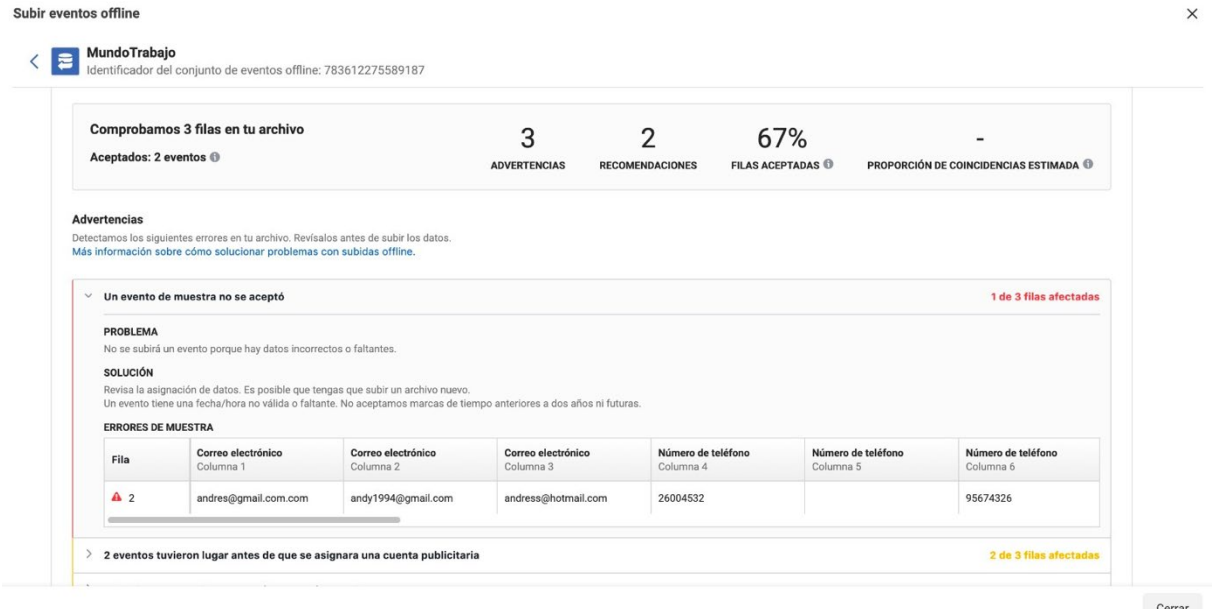


Ilustración 79 - Pantalla de revisión de *Facebook Ads*

Por último, si todos los errores son arreglados, se podrá proceder al inicio de subida de archivo. Este puede demorar algunos minutos dependiendo del tamaño y registros que tenga el archivo.

Una vez procesador los datos, se podrán visualizar la cantidad final de registros subidos o emitidos, y demorará hasta 15 minutos en aparecer en el administrador de eventos y en los informes publicitarios de la plataforma.

Complejidades del flujo

Tal como se puede ver, este es un flujo que, si bien no consta de muchos pasos, puede ser tedioso sobre todo si se hace para varios clientes de forma diaria. Las validaciones de *Facebook* son pocos flexibles, generando que eventos mal formateados o especificados puedan ser omitidos en su procesamiento y no ser tenidos en cuenta al arrojar estadísticas.

13.4 *Wireframes, mockups* y prototipos iniciales

13.4.1 Wireframes

Los primeros *wireframes* fueron realizados junto con el cliente mediante lápiz y papel.

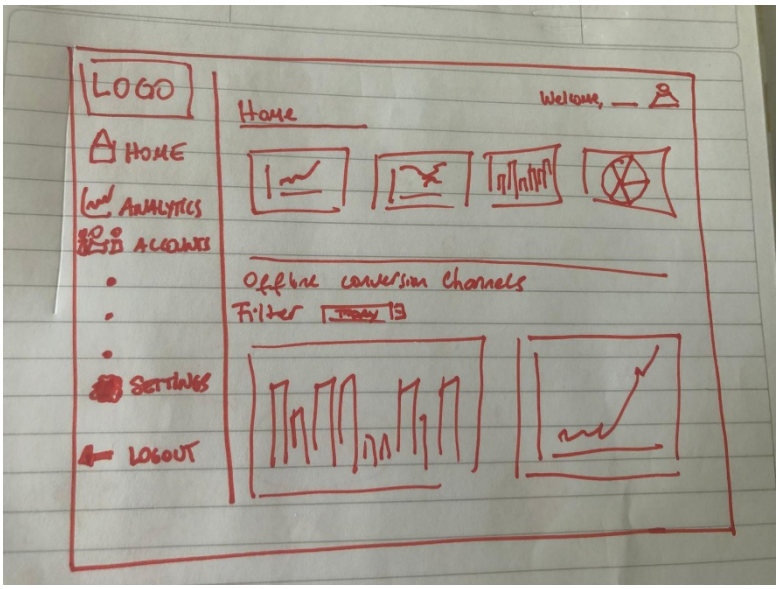


Ilustración 80 - *Wireframe* a papel de Home

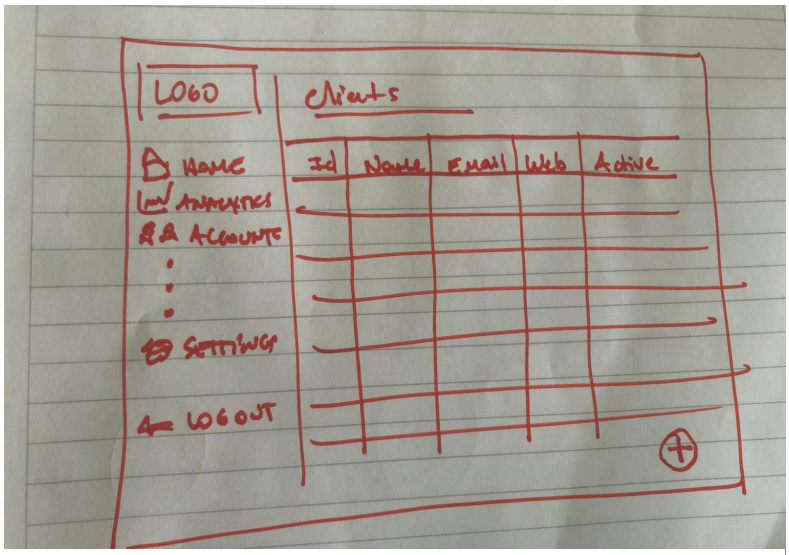


Ilustración 81 – *Wireframe* a papel de Clientes

13.4.2 M*ockups* y Prototipos iniciales

Luego de conceptualizar las pantallas con el cliente, el equipo procedió a realizar *mockups* y prototipos de la plataforma utilizando Axure RP, herramienta de diseño de *mockups* que permite simular los flujos del sistema mediante acciones ejecutadas tales como clicks o inputs.

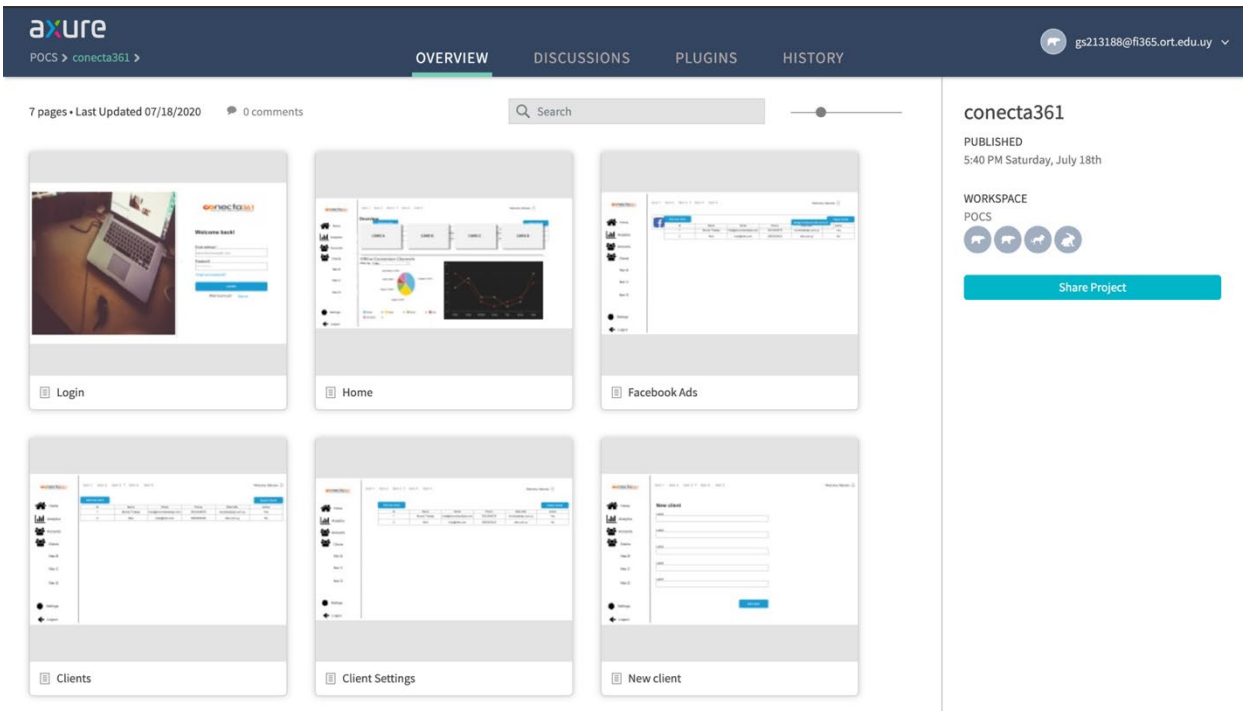


Ilustración 82 - Pantalla de administración de *Axure RP*

A continuación, se muestran algunos de los prototipos realizados:

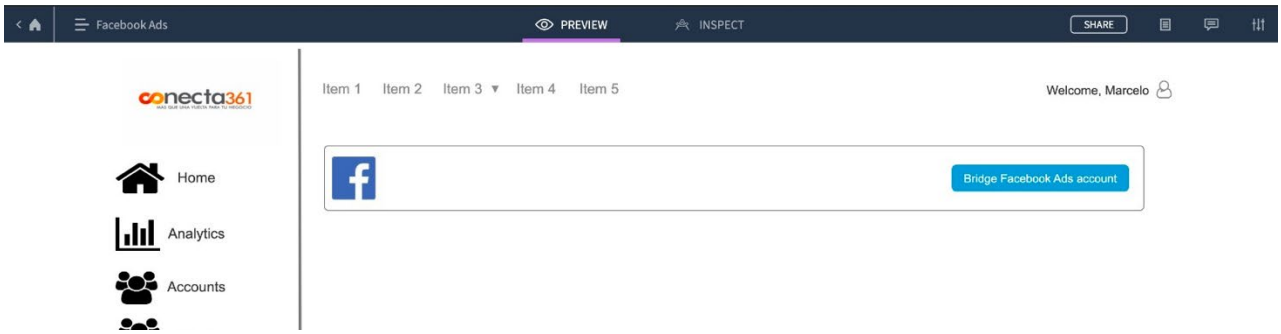


Ilustración 83 - Prototipo de asociación de *Facebook Ads*

Ilustración 84 - Prototipo de pantalla de clientes

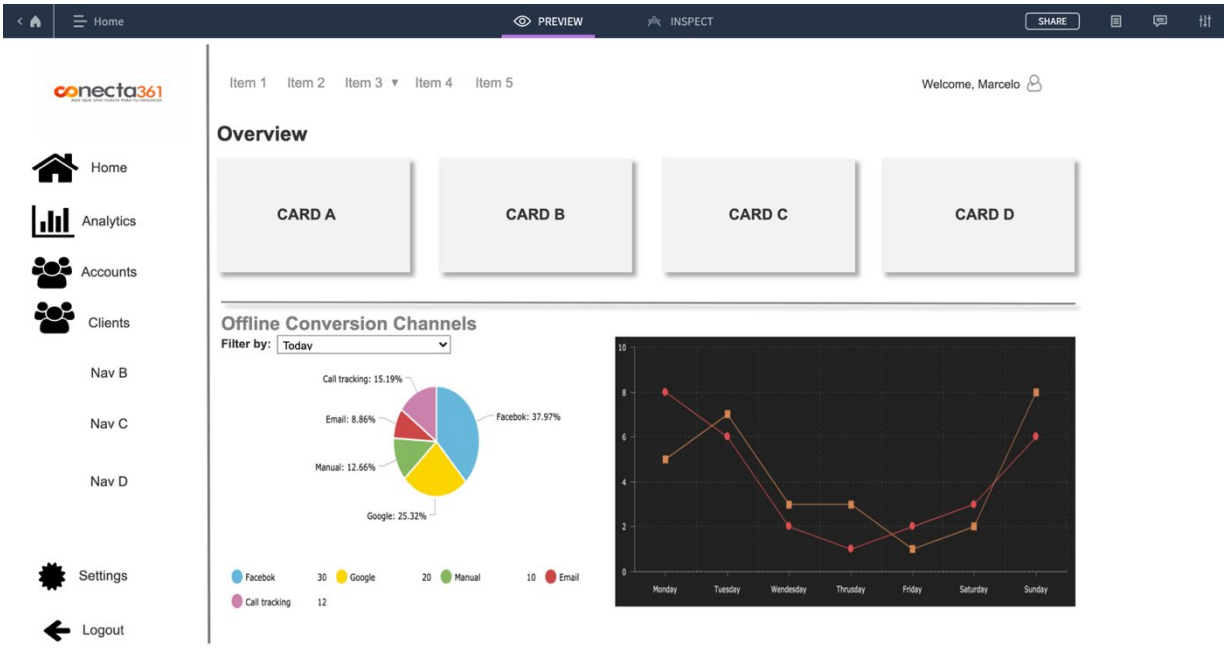


Ilustración 85 - Prototipo de pantalla de inicio

13.5 Resumen de herramientas utilizadas

Nombre	Descripción	Propósito
	Herramienta de diseño utilizada para la prototipación	Prototipación
	Herramienta utilizada para la gestión de trabajo de equipos ágiles, centrada en "planificar, supervisar y lanzar software de primer nivel" [126]	Gestión y seguimiento de las tareas relacionadas a requerimientos y defectos del sistema
	Herramienta que permite el seguimiento de horas dentro de un proyecto	Seguimiento de horas
	Herramienta de video llamadas	Comunicación interna en el equipo
	Herramienta de video llamadas	Comunicación con el cliente
	Herramienta de almacenamiento compartido de Google	Documentación
	Editor de texto que permite la colaboración de equipo simultánea	Documentación

	Proveedor de *Infrastructure as a Service* (IaaS) de tipo Cloud Computing	Infraestructura
	Herramienta de almacenamiento de código mediante repositorios y permite el manejo de versionado.	Almacenamiento de código fuente
	Herramienta de integración continua, permite la integración con GitHub	Integración y despliegue continuo
	Herramienta de monitoreo y manejo de *logs* de sistemas, funciona como *IaaS* (*Infrastructure as a Service*)	Monitoreo del sistema
	Repositorio privado que permite manejar dependencias dentro de sistemas	Repositorio de dependencias
	Herramienta de diagramación *online*	Creación de diagramas
	Herramienta de diagramación.	Creación de diagramas
	Cliente HTTP que permite consultar y testear el manejo de APIs	Pruebas manuales de integración de la API. Especificación de la API

sonarqube	Plataforma utilizada para evaluar código y obtener métricas del mismo	Medición de calidad de código
miro	Herramienta para equipos distribuidos, permite crear y modificar diagramas en conjunto	Herramienta visual colaborativa
APACHE JMeter™	Herramienta para la realización de pruebas de carga de sistemas	Pruebas de carga
	Editor de texto	Creación de diagramas

Tabla 18 - Resumen de herramientas utilizadas

13.6Justificación de herramientas utilizadas

Axure

El equipo decidió utilizar *Axure* para realizar algunos de los prototipos del sistema por la libertad que brinda la herramienta, y la gran cantidad de interacción que permite con los prototipos. Además, dos de los integrantes del equipo ya habían trabajado con la herramienta y les pareció adecuada su utilización.

Jira

El equipo optó por la utilización de *Jira* por sobre otras herramientas de la misma índole, principalmente por la gran cantidad de métricas y posibilidad de análisis que ofrece *Jira.* Además, el equipo se sentía cómodo con la herramienta debido a que la habían utilizado todos en reiteradas ocasiones.

Clockify

Se optó utilizar *Clockify* porque el equipo ya lo había utilizado, así como también porque permite la integración con Jira.

Microsoft Teams

Se utilizó *Microsoft Teams* para la comunicación con el tutor, por el motivo de que ya era la herramienta utilizada para todos los aspectos relacionados a la universidad, y era de mayor comodidad tanto para el tutor como para los integrantes del equipo.

Skype

Skype fue utilizado para tener video llamadas con el cliente, principalmente al inicio del proyecto, debido a la pandemia del COVID-19, la cual no permitió que el equipo se junte con el cliente de manera presencial.

Google Drive

Se utilizó *Google Drive* para el manejo y almacenamiento de prácticamente todos los documentos realizados en el proyecto. Esto se decidió porque todos los integrantes utilizaban asiduamente *Google Drive*, además de la vasta cantidad de herramientas que ofrece y el no tener costo para el almacenamiento.

Microsoft Word

Microsoft Word se utilizó para la realización de la documentación final. Se optó por esta herramienta porque es el editor de texto de preferencia utilizado por el equipo hace mucho tiempo, además de que permite la colaboración de los integrantes en simultáneo.

AWS

Se optó por la utilización de *AWS* para realizar todo el hosting de la infraestructura del sistema por varios motivos. Principalmente fue debido a la libertad y amplia gama de servicios que ofrece, así como también la motivación del equipo de conocer mejor la herramienta para obtener experiencia en la misma. Una justificación más detallada se puede ver en el Anexo 13.11.

GitHub

Se optó por utilizar *GitHub* como cliente de *git*, para el versionado del código. Esto permitió mantener el repositorio de manera remota, manejar el historial de cambios, realizar *code reviews*, y muchas otras funcionalidades que fueron de gran utilidad para el proyecto. Se optó por esta herramienta por motivos de gustos personales del equipo.

TravisCI

Se optó por la elección de *TravisCI* para el manejo de la integración continúa debido a que brindaba un paquete de estudiantes gratuito, además de que permitía la integración directa con *GitHub*. La utilización de este fue de gran ayuda para el equipo en cada momento que se hacía una nueva *pull-request*, ya que se corrían automáticamente las

pruebas de integración, verificando que los cambios ingresados no afecten al resto del sistema.

DataDog

Se optó por la utilización de *DataDog* por sobre otras herramientas de monitoreo, principalmente por el motivo de que venía incluida en el paquete de *GitHub Education*, así como también por la motivación del equipo de aprender a utilizar la herramienta.

Nexus

Se optó por utilizar *Nexus* para el manejo del repositorio de paquetes privados dentro del sistema debido a que ofrece una gran ventaja para el control de versiones de los componentes, lo cual permitió aplicar las buenas prácticas de integración y despliegue continuo.

Draw.io

Draw.io es una herramienta para la creación de diagramas ampliamente utilizada en el mundo. El equipo optó por utilizarla debido a que ya la conocía y había utilizado, y también por la facilidad y rapidez que brinda para realizar diagramas y algunos prototipos simples.

Astah

Se optó por la utilización de *Astah* para los diagramas de la documentación debido a la gran cantidad de funcionalidades que presenta. Se puede diagramar cualquier aspecto del sistema, permitiendo guardar aspectos técnicos de este, la cual se puede replicar en distintos diagramas sin tener que realizar trabajo duplicado.

Postman

Se utilizó *Postman* para acompañar el desarrollo de la API del *backend*, debido a que el mismo permite con gran facilidad probar los distintos *endpoints* del sistema, permitiendo también tener una especie de documentación de la estructura de la API del sistema.

SonarQube

Se utilizó *SonarQube* para obtener métricas detalladas de distintos aspectos del sistema. Se optó por la elección del este debido a la gran cantidad de métricas que brinda, además de que permite la integración directa con *GitHub*.

Miro

Se utilizó *Miro* para realizar algunos diagramas colaborativos de manera remota. Se optó por la utilización del mismo debido a que algunos integrantes del equipo lo habían utilizado en algunas materias de la carrera, y les pareció una herramienta interesante para realizar diagramas en conjunto, en las etapas iniciales del proyecto, al momento de proponer ideas.

JMeter

Se optó por utilizar *JMeter* para las pruebas de carga, debido a que los integrantes del equipo lo habían utilizado recientemente en otra materia de la carrera, y se sintieron cómodos con la herramienta.

Pages

Pages es un editor de texto que viene incluido en los sistemas de Apple, por lo que el equipo lo utilizó para realizar algunos diagramas.

13.7 Ingeniería Inversa

Se realizó ingeniería inversa sobre otros sistemas de funcionalidades similares con el objetivo de tener una visión más amplia del negocio, así como el surgimiento de nuevas ideas que agreguen valor a la plataforma. A continuación, se brinda un resumen de las principales detallando su valor en la aportación a la solución.

Leadsbridge

Ilustración 86 - *Leadsbridge*

Leadsbridge es una plataforma de generación de clientes potenciales integral que ofrece un conjunto de herramientas para conectar todas sus fuentes de generación de clientes potenciales con más de 240 plataformas [127].

A través de los blogs publicados en su página web, el equipo aprendió mucho de las conversiones *offline* y cómo funcionan. Si bien esta no se dedica exclusivamente a la detección de conversiones offline, si tiene integraciones con *Facebook* y *Google* y ayudó al equipo a entender sobre estas.

Segment

Segment es una plataforma *Customer Data Platform* (CDP por sus siglas en inglés), que brinda un servicio que simplifica la recopilación y el uso de datos de los usuarios de sus sitios web. *Segment* puede recopilar, transformar, enviar, y archivar los datos de sus clientes [128].

Segment fue otra de las plataformas que trabajan con conceptos y soluciones similares que el equipo buscaba. Se realizó una inspección sobre su plataforma y se extrajeron conceptos que luego fueron utilizados en *"Tracker361",* como lo son las fuentes de datos y las conexiones entre estas y los procesadores.

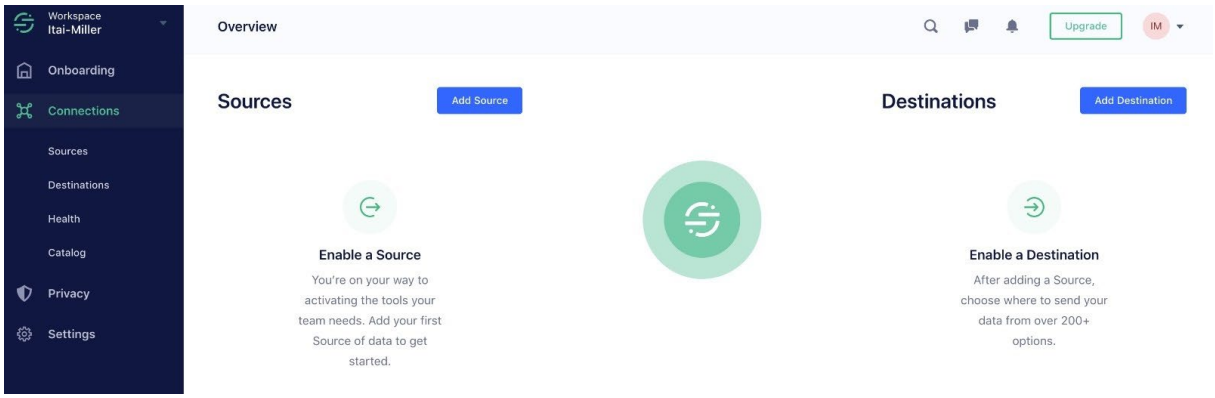


Ilustración 87 - Vista de Segment

Innconcert

Innconcert es una "plataforma de software integrada que ayuda a rentabilizar tus acciones de marketing y obtener el máximo rendimiento de tu inversión en medios" [129] Esta plataforma permitió al equipo obtener información acerca de cómo mostrar estadísticas de *marketing* digital a los usuarios. El equipó aplicó el modelo para utilizarlo en su propia solución, por ejemplo, en la vista de inicio que se compara a continuación:

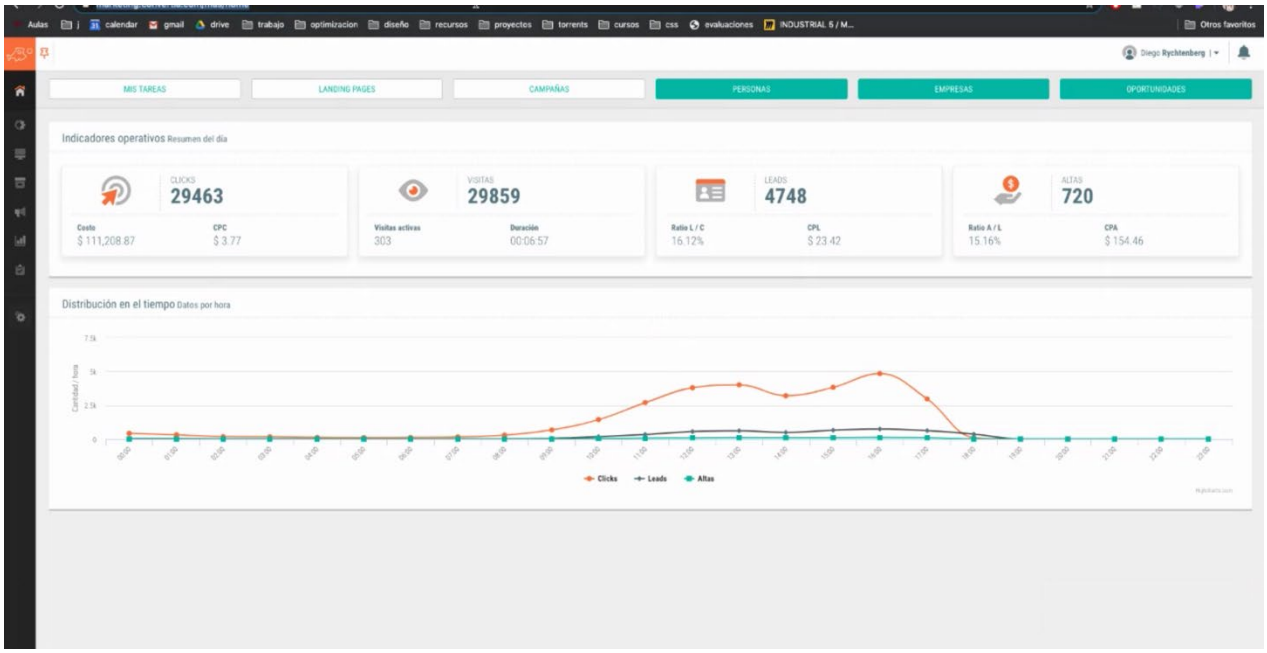


Ilustración 88 - Vista de inicio de *Innconcert*

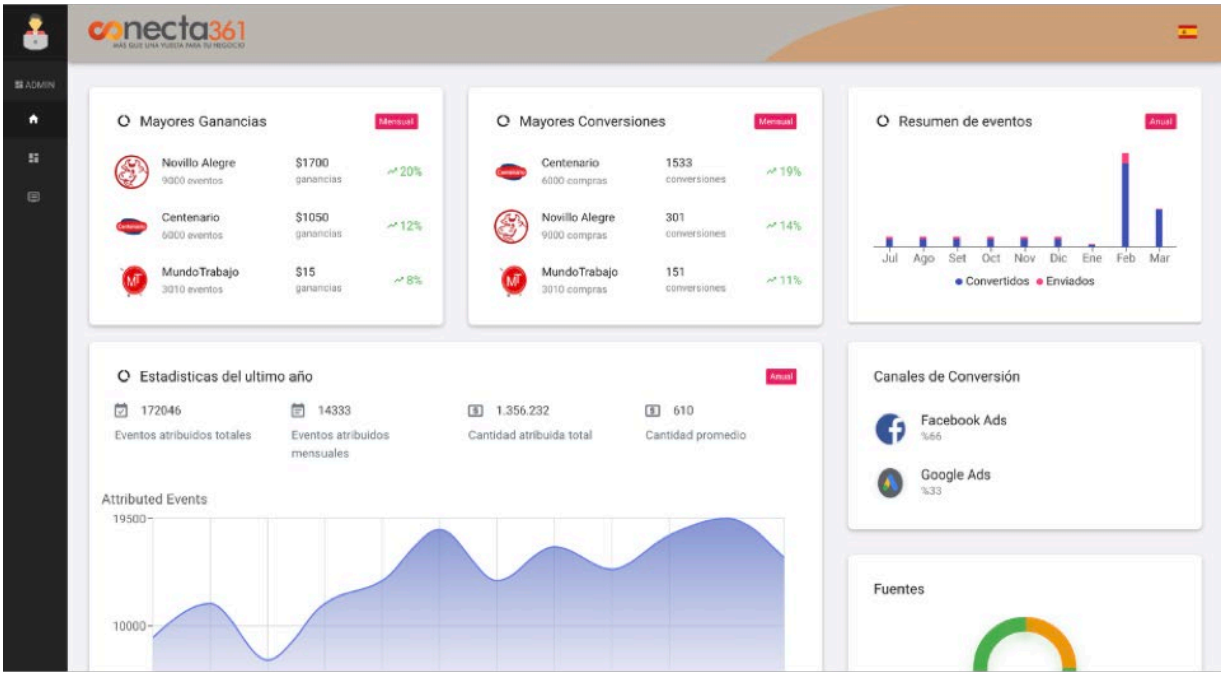


Ilustración 89 - Vista de *Tracker361*

Tal como se puede observar, ambas vistas son muy parecidas. Estas utilizan *cards* para mostrar diferentes estadísticas y luego en la parte inferior de su pantalla muestran una gráfica de rendimiento.

13.8Evidencia de primeras investigaciones

Búsquedas de diferentes mecanismos de detección de conversión *offline*

- Cupones con código de descuento.
- Códigos QR
- El boca a boca. “Me dijeron que si vengo a nombre de x me aplican un descuento “ – Influencers
- GCLID
- Facebook
- Formularios
- Conversiones post llamadas
- Conversiones mailchimp
- Mercadolibre
- Google AdWord GPS

Diagramas conceptuales

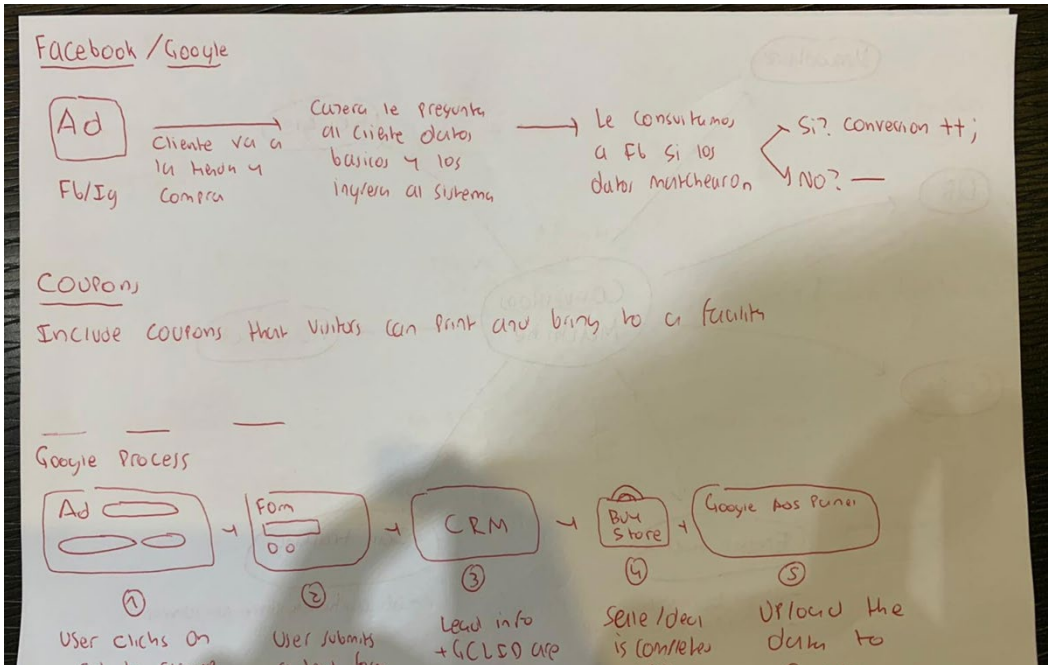


Ilustración 90 - Diagrama conceptual de *administradores de anuncios*

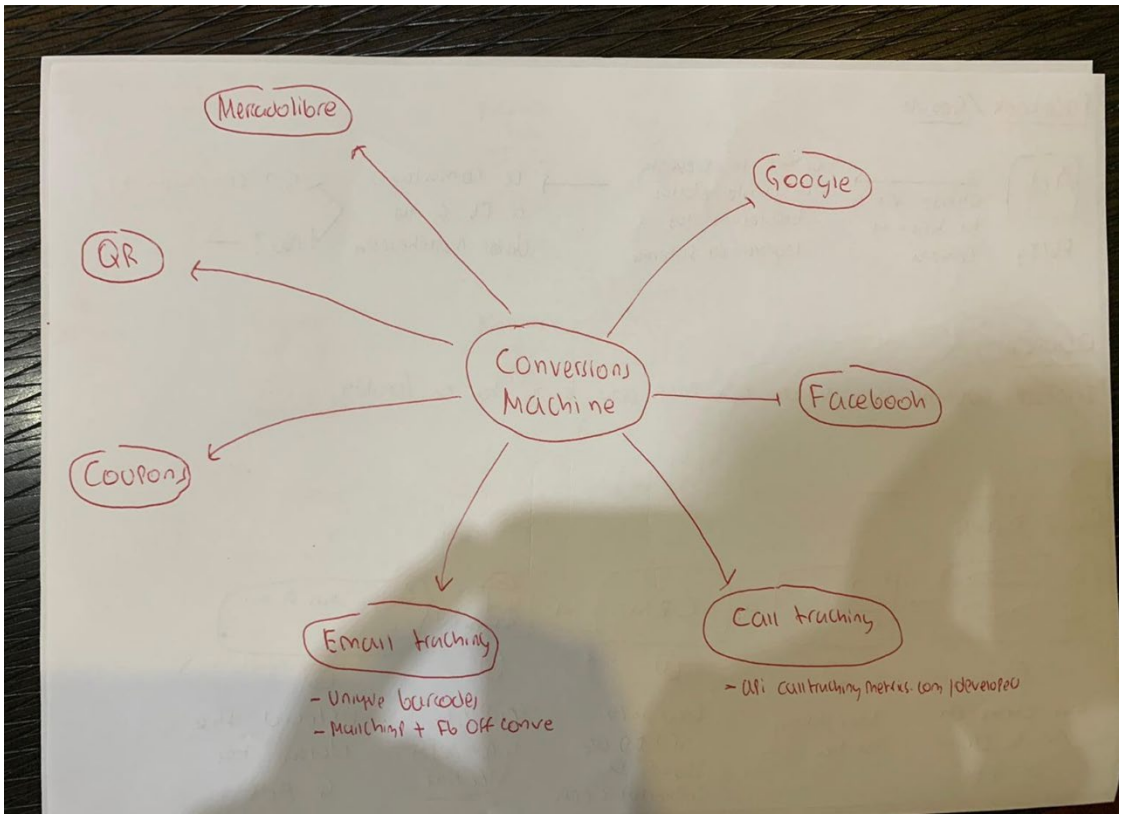


Ilustración 91 - Mindmap de mécanismos de detección de conversiones *offline*

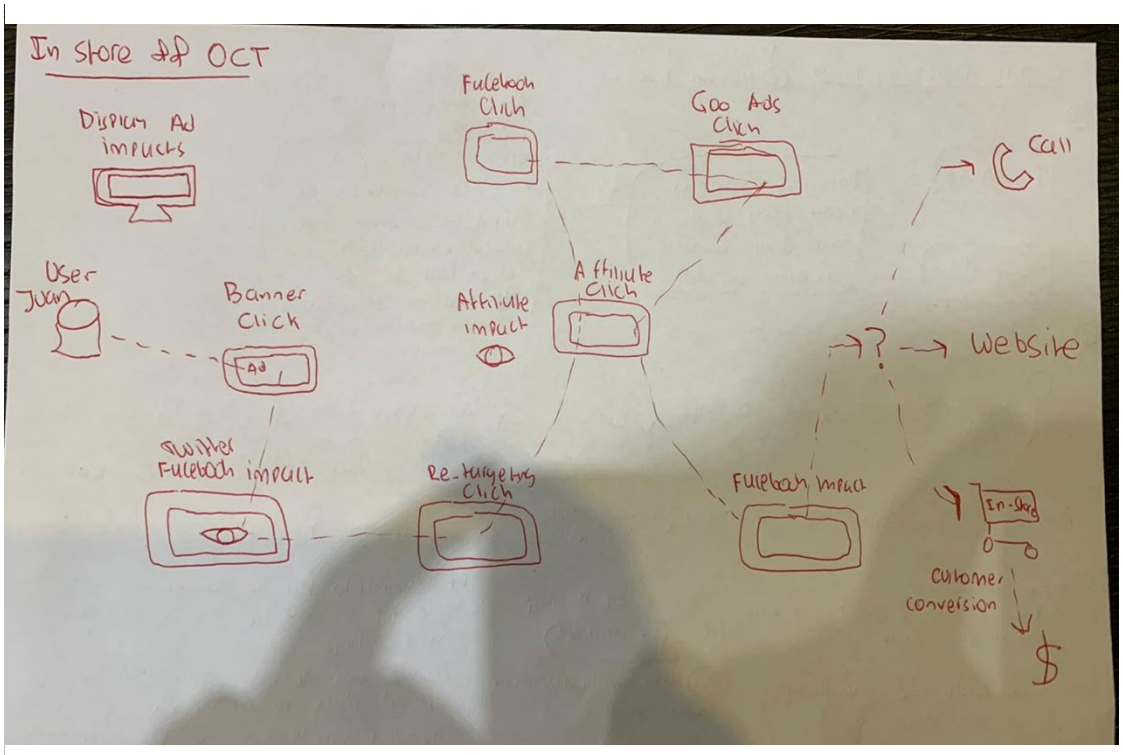


Ilustración 92 - Diagrama conceptual de In Store y OCT

13.9 Listado de principales temas de investigación

- ¿Qué es, cómo funciona y para qué sirve el marketing digital? ¿Qué estrategias existen?
- ¿Qué tipos de conversiones existen en el marketing digital?
- ¿Qué herramientas se utilizan en la actualidad para el marketing digital?
- Conversiones *offline*. ¿Qué son? ¿Cómo funcionan? ¿Qué mecanismos actuales existen?
- Plataforma de anuncios de *Facebook*
- Plataforma de anuncios de *Google*
- API de conversiones offline de *Facebook*
- API de conversiones offline de *Google*
- Comportamiento y contexto de las ventas *retail*
- Conversiones *offline* mediante *Zapier*
- API de *MercadoPago*
- Conversiones *offline* mediante e*mailing*
- Integración con *Saico* (CRM)
- Repositorios privados con *Nexus*
- Integración continua con *BitBucket*
- Integración continua con *GitHub*
- Integración continua con *TravisCI*

13.10 Evidencias de flujos de investigación de APIs y sistema

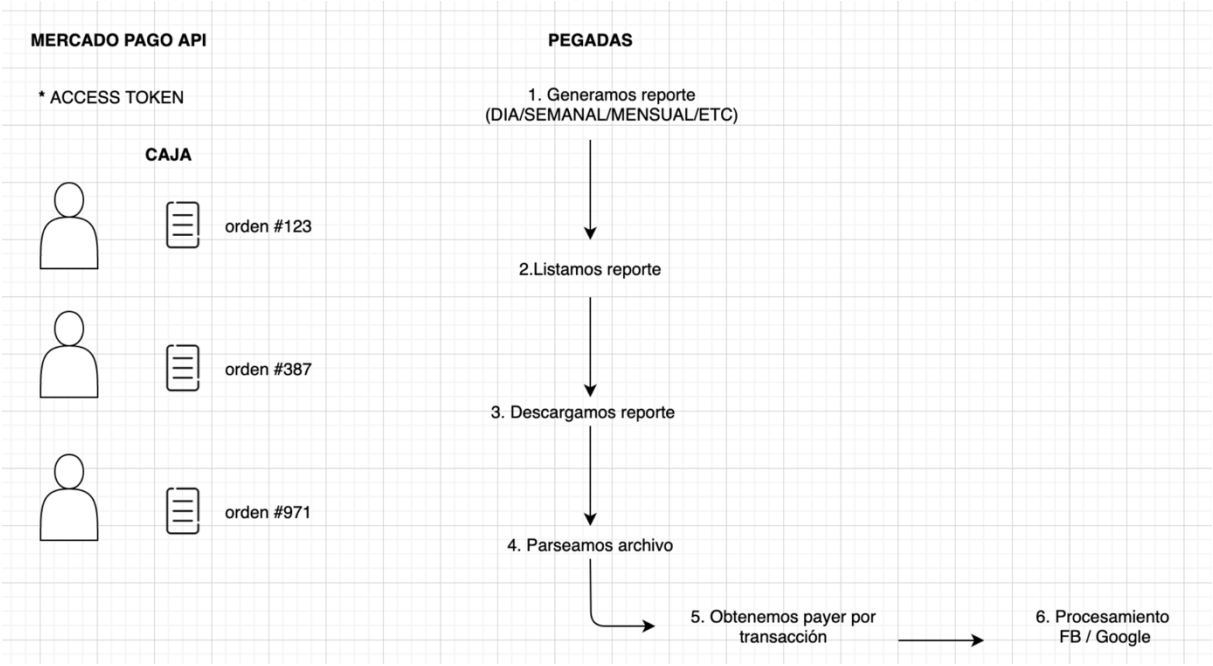


Ilustración 94 - Diagrama de investigación de MercadoPago

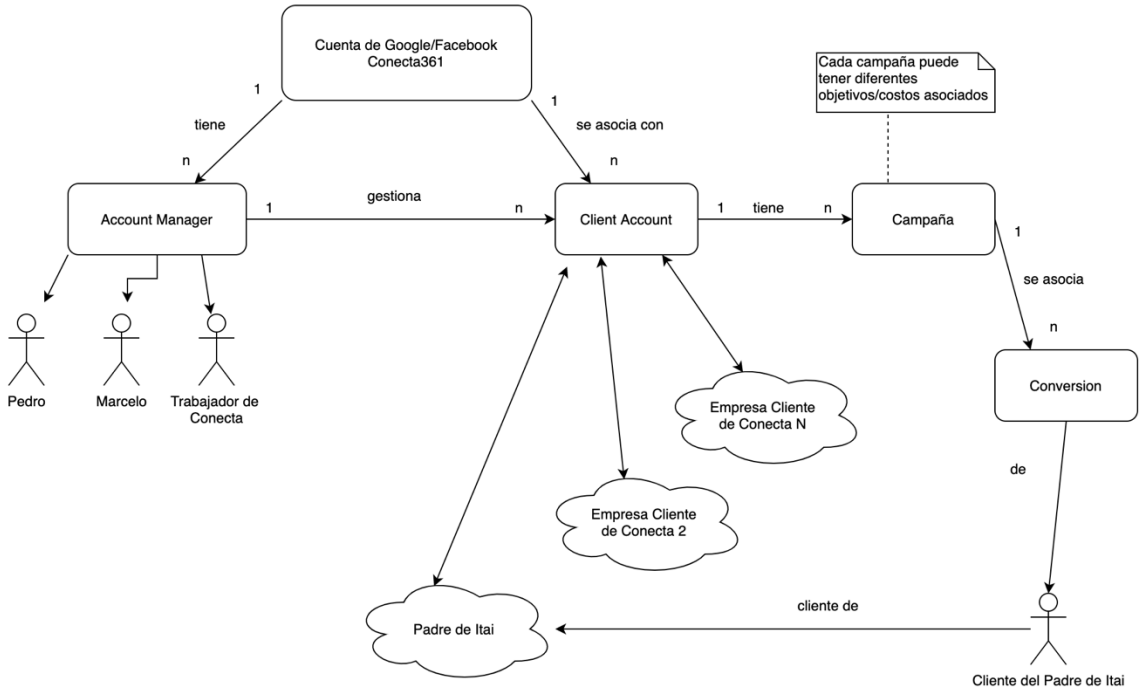


Ilustración 93 - Diagrama de relaciones de la solución

13.11 Product Backlog

A continuación, se muestra la lista de requerimientos funcionales en formato de historias de usuario.

Historias de usuario	Prioridad
Como cualquier usuario debo poder hacer *login* al sistema	Alta
Como cualquier usuario debo poder hacer *logout* al sistema	Alta
Como *Super Admin* debo poder configurar los proveedores de datos de mis clientes	Alta
Como *Admin* debo poder importar un archivo csv con los datos de mis compradores	Alta
Como *Admin* debo poder ver las estadísticas de mis campañas	Alta
Como *Super Admin* debo poder crear las cuentas publicitarias de mis clientes	Alta
Como *Admin* debo poder crear las cuentas de más usuarios administradores	Alta
Como *Admin* o *Client* debo poder activar mi cuenta desde el enlace recibido por correo	Alta
Como *Super Admin* debo recibir una notificación por correo cuando un cliente active su cuenta	Media
Como *Super Admin* debo poder actualizar los datos de un cliente	Alta

Como cualquier usuario debo poder actualizar mi contraseña	Media
Como *Super Admin* debo poder ver un listado de todos mis clientes	Alta
Como *Admin* debo poder ver un listado de todos mis administradores	Media
Como cualquier usuario debo poder ver mensajes de error en el sistema	Alta
Como *Admin* debo poder ver un listado de todos mis compradores	Alta
Como *Admin* debo poder editar cualquier dato de mis compradores	Alta
Como *Super Admin* debo poder filtrar el listado de mis clientes por Nombre, País o Estado	Alta
Como *Admin* debo poder filtrar el listado de mis compradores por Nombre, Apellido, Sexo o País	Alta
Como *Super Admin* debo poder ver el estado de mis Clientes en el listado	Media
Como *Super Admin* debo poder ver el detalle de un Cliente	Alta
Como *Super Admin* debo poder configurar los datos de mi cuenta de *Facebook*	Alta
Como *Super Admin* debo poder configurar las cuentas de anuncios de mis clientes	Alta

Como *Super Admin* debo poder ver las estadísticas de campañas de mis clientes	Alta
Como *Super Admin* debo poder ver un listado de los eventos convertidos de todos los clientes	Alta
Como *Super Admin* debo poder filtrar el listado de eventos convertidos.	Alta
Como *Admin* debo poder ver un listado de mis eventos convertidos	Alta
Como *Admin* debo poder filtrar el listado de eventos convertidos	Alta
Como *Super Admin* debo poder ver las estadísticas generales del sistema	Alta
Como cualquier usuario debo poder cambiar el lenguaje del sistema	Media
Como *Admin* debo poder ver el detalle de un comprador	Alta
El sistema debe proveer una API para los clientes de *Conecta361* presentando los datos y estadísticas de las campañas	Alta
Implementar una *SDK* para los clientes de *Conecta361* para que integren sus propias soluciones directamente al sistema.	Alta
El sistema debe poder generar y obtener datos desde y hacia la API de *Facebook Ads*	Alta
El sistema debe poder generar y obtener datos desde y hacia la API de *Google Ads*	Baja

El sistema debe obtener datos de los compradores mediante la API de *MercadoPago*	Alta
El sistema debe obtener datos de los compradores mediante la API de *MailChimp*	Media
El sistema debe poder importar archivos de datos exportados de *Zoho*	Alta
El sistema debe poder importar archivos de datos exportados de *MailChimp*	Alta
El sistema debe poder importar archivos de datos exportados en un formato propio	Alta
El sistema debe presentar una interfaz para la implementación de otros proveedores de anuncios (ej: *Microsoft Ads*)	Alta
El sistema debe normalizar los datos obtenidos de los distintos proveedores para generar los eventos	Alta
Se debe proveer un *endpoint* HTTP que indique el estado de salud del sistema	Media
El sistema debe poder realizar *retargeting* de campañas	Baja

Tabla 19 - Product Backlog

13.12 API *Endpoints*

A continuación, se especifican algunos de los principales *endpoints* de la API REST provista por el *backend*.

- URL base: https://api.conecta361.tk

Recurso	Descripción	Verbo HTTP	Endpoint
Clients	Obtener a los compradores de un cliente	GET	/api/v1/clients/{id}/payers
	Obtener los datasets de un cliente	GET	/api/v1/clients/{id}/datasets
	Crear cuenta publicitaria en Facebook para un cliente	POST	/api/v1/clients/{id}/facebook/account
	Obtener datos de la cuenta publicitaria de un cliente	GET	/api/v1/clients/{id}/facebook/account
	Editar la cuenta publicitaria del cliente	PUT	/api/v1/clients/{id}/facebook/account

	Crear dataset para la cuenta publicitaria de un cliente	POST	/api/v1/clients/{id}/facebook/dataset
	Asociar una fuente de datos a un cliente	POST	/api/v1/clients/{id}/providers
	Editar la asociación de una fuente de datos a un cliente	PUT	/api/v1/clients/{id}/providers
	Obtener las fuentes de datos asociadas a un cliente	GET	/api/v1/clients/{id}/providers
	Eliminar la asociación entre la fuente de datos y el cliente	DELETE	/api/v1/clients/{id}/providers
Auth	Login	POST	/api/v1/auth/login
	Logout	DELETE	/api/v1/auth/logout

Events	Obtener todos los eventos de un cliente	GET	/api/v1/events/clients/{id}
Payers	Importar archivo de compradores	POST	/api/v1/payers/import
	Crear un comprador	POST	/api/v1/payers
	Editar un comprador	PUT	/api/v1/payers/{id}/edit
	Eliminar un comprador	DELETE	/api/v1/payers/{id}/delete
	Obtener un comprador	GET	/api/v1/payers/{id}
Providers	Crear una fuente de datos	POST	/api/v1/providers
	Editar una fuente de datos	PUT	/api/v1/providers/{id}/edit
	Eliminar una fuente de datos	DELETE	/api/v1/providers/{id}/delete
Processors	Crear un procesador	POST	/api/v1/processors

	Obtener todos los procesadores	GET	/api/v1/processors
Stats	Obtener las estadísticas de Facebook	GET	/api/v1/stats/{clientId}
Users	Crear usuario colaborador del negocio	POST	/api/v1/users
	Editar un usuario colaborador	PUT	/api/v1/users/{id}/edit
	Editar permisos de usuario colaborador	PUT	/api/v1/users/{id}/role
	Eliminar un usuario colaborador	DELETE	/api/v1/users/{id}/delete
	Obtener un usuario colaborador	GET	/api/v1/users/{id}
	Cambiar la contraseña	PUT	/api/v1/users/reset/password

Admin	Configurar la cuenta al administrador	POST	/api/v1/admin/settings
	Obtener la cuenta del administrador	GET	/api/v1/admin/settings
	Obtener la metadata asociada al administrador	GET	/api/v1/admin/metadata

Tabla 20 - API endpoints

13.13 Análisis de hosting para el sistema

Se tuvieron en cuenta distintas posibilidades para la infraestructura del sistema, siempre dentro del rango de los proveedores de *cloud computing* conocidos. Se descartó la opción de realizar un sistema de infraestructura propia, por falta de recursos económicos, así como por la facilidad que brindan los servicios de *cloud computing*. Las dos principales opciones que se tuvieron en cuenta fueron Amazon Web Services (AWS) y Heroku. A continuación, se detalla cada una de ellas y se presenta el análisis realizado.

AWS

Es un proveedor de *Infrastructure as a Service* (IaaS). El mismo brinda la posibilidad de manejar prácticamente toda la infraestructura de un sistema desde lo que denominan la Consola de AWS. Esta ofrece todo tipo de servicios, desde instancias para poder de cómputo, bases de datos, sistemas de seguridad, sistemas de mensajería, etc. Es uno de los proveedores de infraestructura más utilizados en el mundo, por lo cual cuentan con un excelente servicio de soporte y una amplia comunidad.

Además de lo ya mencionado, algunos de los integrantes del equipo ya contaban con experiencia previa trabajando con AWS, lo cual fue un factor importante al momento de la decisión.

Heroku

Heroku brinda un servicio de *Platform as a Ser*vice (PaaS). El mismo permite manejar y coordinar distintas aplicaciones de una manera simplificada. Busca eliminar muchas de las complejidades que normalmente se encuentran al momento de manejar la infraestructura de un sistema, gestionándolas ellos mismos en su servicio. Esto brinda la posibilidad de centrarse más en el sistema, y menos en la infraestructura en sí, pero al mismo tiempo limita de distintas maneras las posibilidades que se pueden variar dentro de la infraestructura de un sistema.

El equipo no contaba con nada de experiencia previa en Heroku al momento de iniciar el proyecto.

Comparación:

Las ponderaciones se corresponden con los siguientes niveles:

Alto – 5 puntos

Medio – 3 puntos

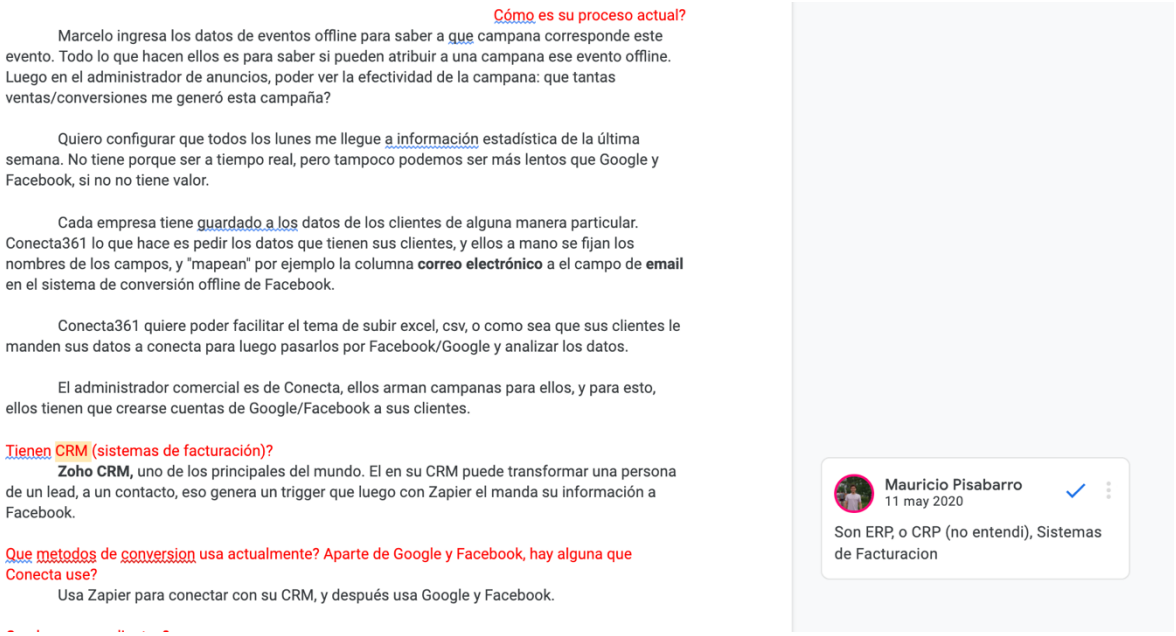
Bajo – 1 punto

	AWS	Heroku
Libertad de manejo de infraestructura	5	3
Comunidad y soporte	5	5
Experiencia del equipo	3	0
Facilidad de aprendizaje	3	5
Total	16	13

Tabla 21 - AWS vs Heroku

Al analizar ambas plataformas, y teniendo en cuenta las preferencias del equipo, se optó por utilizar AWS para hostear el sistema. Por más que el equipo tenía claro que Heroku es una herramienta más fácil de utilizar, el equipo fue motivado por el aprendizaje de la herramienta de AWS, la cual brinda mucha más libertad a la hora de armar la infraestructura, y optó por sacrificar un poco de facilidad para aprender más acerca del mundo de la infraestructura y obtener los conocimientos para el futuro.

13.14 Evidencia de *feedback* recolectado en *Sprint Reviews*



Cómo es su proceso actual?

Marcelo ingresa los datos de eventos offline para saber a que campana corresponde este evento. Todo lo que hacen ellos es para saber si pueden atribuir a una campana ese evento offline. Luego en el administrador de anuncios, poder ver la efectividad de la campana: que tantas ventas/conversiones me generó esta campaña?

Quiero configurar que todos los lunes me llegue a información estadística de la última semana. No tiene porque ser a tiempo real, pero tampoco podemos ser más lentos que Google y Facebook, si no no tiene valor.

Cada empresa tiene guardado a los datos de los clientes de alguna manera particular. Conecta361 lo que hace es pedir los datos que tienen sus clientes, y ellos a mano se fijan los nombres de los campos, y "mapean" por ejemplo la columna **correo electrónico** a el campo de **email** en el sistema de conversión offline de Facebook.

Conecta361 quiere poder facilitar el tema de subir excel, csv, o como sea que sus clientes le manden sus datos a conecta para luego pasarlos por Facebook/Google y analizar los datos.

El administrador comercial es de Conecta, ellos arman campanas para ellos, y para esto, ellos tienen que crearse cuentas de Google/Facebook a sus clientes.

Tienen CRM (sistemas de facturación)?

Zoho CRM, uno de los principales del mundo. El en su CRM puede transformar una persona de un lead, a un contacto, eso genera un trigger que luego con Zapier el manda su información a Facebook.

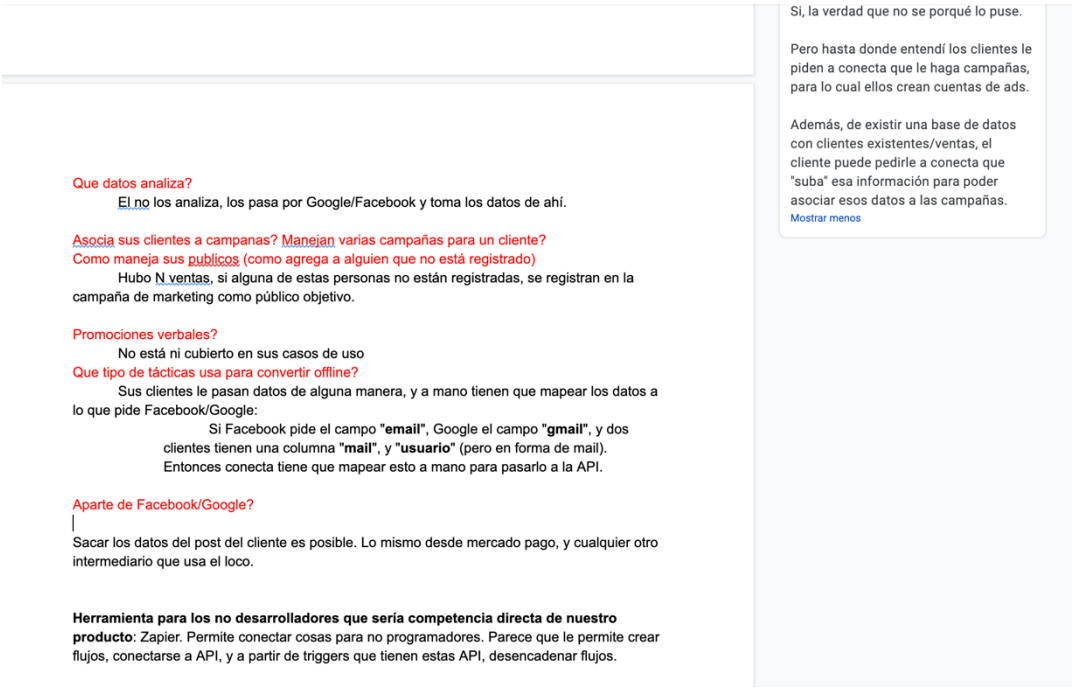
Que metodos de conversion usa actualmente? Aparte de Google y Facebook, hay alguna que Conecta use?

Usa Zapier para conectar con su CRM, y después usa Google y Facebook.

Mauricio Pisabarro
11 may 2020

Son ERP, o CRP (no entendi), Sistemas de Facturacion

Ilustración 95 - Evidencia de *feedback recolectado*



Que datos analiza?

El no los analiza, los pasa por Google/Facebook y toma los datos de ahí.

Asocia sus clientes a campanas? Manejan varias campañas para un cliente?
Como maneja sus publicos (como agrega a alguien que no está registrado)

Hubo N ventas, si alguna de estas personas no están registradas, se registran en la campaña de marketing como público objetivo.

Promociones verbales?

No está ni cubierto en sus casos de uso

Que tipo de tácticas usa para convertir offline?

Sus clientes le pasan datos de alguna manera, y a mano tienen que mapear los datos a lo que pide Facebook/Google:

Si Facebook pide el campo **"email"**, Google el campo **"gmail"**, y dos clientes tienen una columna **"mail"**, y **"usuario"** (pero en forma de mail). Entonces conecta tiene que mapear esto a mano para pasarlo a la API.

Aparte de Facebook/Google?

Sacar los datos del post del cliente es posible. Lo mismo desde mercado pago, y cualquier otro intermediario que usa el loco.

Herramienta para los no desarrolladores que sería competencia directa de nuestro producto: Zapier. Permite conectar cosas para no programadores. Parece que le permite crear flujos, conectarse a API, y a partir de triggers que tienen estas API, desencadenar flujos.

Si, la verdad que no se porqué lo puse.

Pero hasta donde entendí los clientes le piden a conecta que le haga campañas, para lo cual ellos crean cuentas de ads.

Además, de existir una base de datos con clientes existentes/ventas, el cliente puede pedirle a conecta que "suba" esa información para poder asociar esos datos a las campañas.

Mostrar menos

Ilustración 96 - Anotación sobre *feedback*

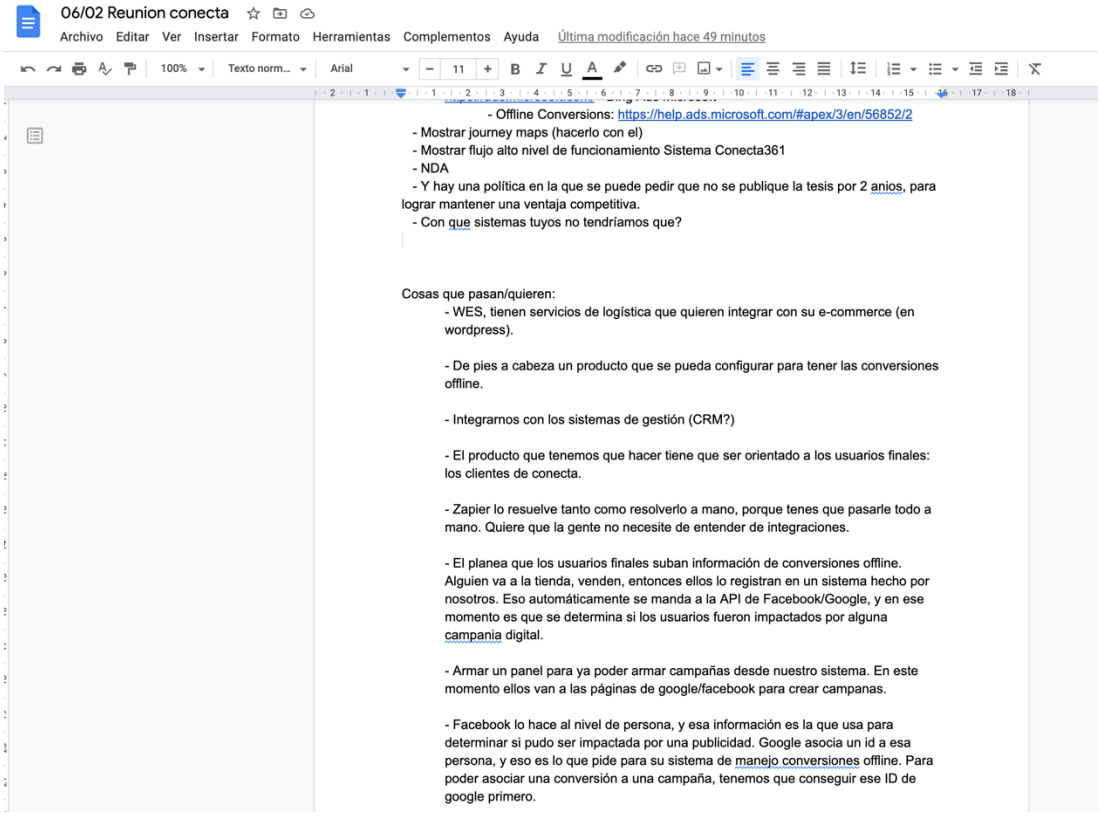


Ilustración 97 - Evidencia de comentarios sobre reuniones

13.15 Hitos principales del proyecto

[12/09/2020] Hito 1 – CI/CD

Para lograr que el producto esté puesto en producción al finalizar el proyecto, el equipo se propuso como 1er hito que la infraestructura tecnológica para el desarrollo e integración continua del producto sea lograda en su totalidad, con el fin de reducir los riesgos correspondientes a la correcta puesta en producción de este. Dentro del hito se destacan las siguientes tareas:

- Infraestructura *cloud*. Estado: Cumplido.
- Entrega continua. Estado: Cumplido.
- Integración y *deployment* continuo. Estado: Cumplido.
- Manejo de versionado y dependencias. Estado: Cumplido.
- Integración con *DataDog* para el monitoreo, *logging* de salud y *performance* de la aplicación, así como métricas de UI. Estado: Cumplido.

Porcentaje de cumplimiento del hito: 100%

[11/10/2020] Hito 2 – Panel administrativo funcional.

Dado que todos los flujos comienzan con el sistema cargado de clientes, usuarios, y compradores, se decidió que el primer hito relacionado a funcionalidad brinde la capacidad de administrar estas entidades. Dentro del hito se destacan las siguientes tareas:

- CRUD de clientes a nivel de *back* y *front*. Estado: Cumplido.
- CRUD de usuarios a nivel de *back* y *front*. Estado: Cumplido.
- CRUD de compradores a nivel de *back* y *front*. Estado: Cumplido.
- Manejo de roles. Estado: Cumplido.

Porcentaje de cumplimiento del hito: 100%

- [29/11/2020] Hito 3 – Procesamiento de conversiones offline.

En este hito es donde se encuentra las funcionalidades *CORE* del sistema. Abarca la recolección de los datos de los usuarios de las distintas fuentes de datos y su posterior normalización y envío a los procesadores para la realización de una posible conversión. Dentro del hito se destacan las siguientes tareas:

- Manejo de eventos. Estado: Cumplido.
- Normalización de los datos obtenidos desde las distintas fuentes (ej: Mercado Pago). Estado: Cumplido.
- Envío de datos normalizados a los diferentes procesadores. Estado: Cumplido.
- Manejo de datos procesados. Estado: Cumplido.

Porcentaje de cumplimiento del hito: 100%

- [27/12/2020] Hito 4 – Integración completa de las conversiones con datos procesados

El trabajo del último hito consiste en el registro de eventos de conversiones offline, y la muestra de estadísticas sobre cómo y que campañas afectaron a las conversiones de un cliente dado. Dentro del hito se destacan las siguientes tareas:

- Presentación de los datos obtenidos de los procesadores. Estado: Cumplido.
- Integración completa de las conversiones con el panel. Estado: Cumplido.
- Muestra de estadísticas de las conversiones offline. Estado: Cumplido.
- Estadísticas de las campañas publicitarias. Estado: Cumplido
- SDK. Estado: Cumplido
- *Retargeting* de campañas. Estado: No Cumplido.

Porcentaje de cumplimiento del hito: 90%

13.16 Informes de revisiones formales de Universidad ORT Uruguay

13.16.1 Informe 1 de revisión ORTsf

Revisor: Amalia Álvarez

Fecha: 29 de junio de 2020

Fortalezas:

- Relación clara con el cliente
- Dominio del problema digno de un proyecto de tesis
- Tecnologías claramente definidas
- Repositorios definidos
- Ambiente de desarrollo creados

Sugerencias:

- Definir ciclo de vida que sea más acorde a la incertidumbre que el equipo maneja.
- Crear un plan de requerimientos que permita determinar cuándo un requerimiento está listo para pasar a una etapa de desarrollo.
- Definir cómo se van a especificar los requerimientos.
- Crear un plan de control de riesgos
- Revisar planilla de riesgos más seguido. Definir el ciclo.
- Definir un roadmap.
- Definir un rol de SQA.

13.16.2 Informe 2 de revisión ORTsf

Revisor: Marcelo Cagnani

Fecha: 29 de setiembre de 2020

Fortalezas:

- Uso de entrega continua.
- Uso de pruebas unitarias.
- Comunicación clara con el cliente.
- Métodos de trabajo definidos.
- Proyecto al día con plan de reléase propuesto.

Sugerencias:

- Justificar elección de tecnologías
- Redefinir objetivos con S.M.A.R.T *goals*
- Coordinar con el cliente para llevarle un MVP del sistema
- Definir proceso de puesta en producción.
- Hacer *mocks* de la presentación de revisión.

13.16.3 Informe 3 de revisión ORTsf

Revisor: Pablo Hernandez

Fecha: 22 de diciembre de 2020

Fortalezas:

- Queda claro el problema a la hora de presentarlo.
- Se mantiene un cronograma, y se está al día.
- Se *loguea* el esfuerzo en los *sprints*.
- Método de trabajo claramente definidos.
- Buena comunicación con el cliente.
- Proceso claro de ingeniería de requerimientos.

Sugerencias:

- Finalizar *tasks* restantes en el *backlog.*
- Automatización de *testing end to end.*
- *Testing* de carga.
- Pruebas de usabilidad con usuarios reales.
- Definir un plan de transferencia del producto hacia el cliente.
- Documentación.

13.17 Plan de riesgos

ID	Nombre	Descripción	Categoría	Estrategia de respuesta	Plan de respuesta	Plan de contingencia
R1	Poco conocimiento del negocio	Poco conocimiento de los miembros del equipo sobre el mundo del Marketing Digital	Producto	Mitigar	Investigación y reuniones con cliente	Aumentar cantidad de horas a la investigación
R2	Etapa de investigación retrase el proyecto	No alterar el tiempo de desarrollo por tener que investigar más del tiempo estimado	Producto	Mitigar	Plantearnos *deadlines* que ayuden a cumplir con la fecha propuesta	Aumentar horas de investigación para llegar a la fecha estipulada
R3	Satisfacción del cliente	Cliente no satisfecho con el producto	Producto	Mitigar	Aceptar el *feedback* que plantea el cliente y realizar las modificaciones correspondientes	Realización de demos para *feedback* temprano

R4	Alta deuda técnica	Tener una deuda técnica mayor a 2hs	Producto	Mitigar	Atacar los errores generados en cada sprint, lo antes posible	Dedicar un sprint entero a la resolución de errores para reducir la deuda técnica
R5	Validación del producto	No validar el producto con clientes reales	Producto	Evitar	Conseguir por lo menos un cliente real de Conecta361 para validar el producto	Conseguir un cliente por fuera de Conecta361 para validar el producto
R6	Abandono del tutor	Que el tutor nos abandone	Equipo	Aceptar	Remplazo del tutor	Aceptar
R7	Abandono de un miembro del equipo (temporal/definitorio)	Se pierde a un integrante del equipo de forma temporal o definitiva	Equipo	Aceptar	Avisar al equipo con tiempo que fecha estará ausente para organizarnos cuando la persona no este. En caso de una salida definitiva, volver a	Líder de Gestión del Proyecto debe volver a organizar al equipo para continuar

					planificar el alcance del proyecto	
R8	Poco compromiso del equipo	La falta de compromiso de alguno de los integrantes del equipo puede llevar a alterar la planificación del proyecto	Equipo	Evitar	Acuerdo de compromiso entre integrantes según intereses y objetivos personales.	Reunión entre el equipo para buscar una solución y motivar al integrante para que aumente su compromiso
R9	Conflicto de roles	Responsabilidades mal definidas	Equipo	Evitar	Chequear en cada sprint que cada uno este cumpliendo con la responsabilidad asignada	Reuniones entre el equipo para resolver los conflictos. En caso de no solucionar los mismos, ver de invertir los roles
R10	Mala relación con el cliente	Una mala relación con el cliente puede llevar a perderlo y que se te caiga el proyecto	Cliente	Mitigar	Entender las necesidades del cliente para intentar satisfacerlas	Comunicación frecuente para estar alineados en los objetivos

R11	Finalización crédito plan estudiantil AWS/Travis	No poder cubrir los costos si se acaban los servicios gratuitos de ORT	Financiación	Aceptar	Hablar con ORT para que mantenga el crédito en los servicios utilizados	Pagar el restante entre todos los miembros del equipo
R12	API de Google y Facebook deprecadas	Cambios frecuentes en las APIs puede alterar los tiempos estipulados del proyecto	Tecnología	Aceptar	Fijarse continuamente en la documentación para detectar de forma temprana las actualizaciones	Aceptar
R13	Falta de experiencia en React	Solo uno de los cuatro integrantes tiene experiencia previa trabajando con React	Tecnología	Mitigar	Leer documentación y ver tutoriales para capacitarse	Pair Programming y documentación de React
R14	Mala estimación de User Stories	Estimar mal las tareas puede provocar que el proyecto se atrase por ende alterar la planificación del proyecto	Estimación	Mitigar	Realizar investigaciones antes de comenzar a estimar.	Volver a estimar las User Stories a realizar. Utilizar período de bugs para

					Dejar margen al estimar.	finalizar con las funcionalidades
R15	Mala estimación de la velocidad del equipo	Estimar mal la velocidad del equipo puede provocar que el Sprint se atrase por ende alterar el release plan	Estimación	Mitigar	Ir aumentando la velocidad a medida que se posible	Incrementar la cantidad de horas trabajadas por semana. Volver a construir el release plan
R16	Mal relevamiento de requerimientos	Relevar mal los requerimientos puede llevar a desentendimientos con el cliente	Relevamiento de requerimientos	Mitigar	Validación previa con el cliente mediante prototipos	Volver a relevar los requerimientos con el Cliente
R17	Mala gestión del proyecto	Puede llevar a cometer errores al organizar las tareas. Repercute en los objetivos del proyecto	Planificación	Mitigar	Planificación en etapa temprana y evaluación de herramientas y buenas practicas para la gestión	Si el equipo no puede gestionarse, se pedirá ayuda al tutor para una mejor organización

R18	Mala elección de las tecnologías	Tecnologías utilizadas no van acorde a las necesidades del proyecto.	Tecnología	Evitar	Comparación y trade-offs del uso de las diferentes tecnologías ofertadas en el mercado.	Cambio a tecnologías que ofrezcan desarrollo rápido.
R19	Modificación de requerimientos	El cliente no esta de acuerdo con el requerimiento implementado	Comunicación	Evitar	Validar con el cliente mediante prototipos para minimizar un mal entendimiento	Si el requerimiento es justificado realizar el cambio. Registrar el retrabajo y aprender del error
R20	COVID 2019	Pandemia afecta a la comunicación del equipo y reuniones presenciales con el cliente	Externo	Aceptar	Aceptar	Adaptarse a la nueva normalidad y usar canales digitales para la comunicación entre el equipo y cliente

R21	Perdida de los archivos	Se perdieron archivos que forman parte del proyecto	Control	Evitar	Realizar backups	Restauración de los backups
R22	Conecta361 no "preste" a sus clientes	No conseguir clientes que puedan probar el correcto funcionamiento de la plataforma	Recursos	Evitar	Solicitud al cliente de forma previa.	Utilizar otros medios y canales de usuarios para validar el producto.
R23	Poca prueba y evidencia	El software no va a estar bien probado	Producto	Evitar	Se van a realizar pruebas de distinto tipo a lo largo de todo el sistema	Se realizarán como mínimo pruebas de integración

Tabla 22 - Plan de riesgos

13.18 Seguimiento de principales riesgos del proyecto

En esta sección, se presentan los principales riesgos identificados en el proyecto. De cada uno se detalla una descripción, un plan de respuesta y un plan de contingencia. Además, se especifica el impacto y probabilidad de cada uno permitiendo visualizar como vario su magnitud a lo largo del proyecto.

Poco conocimiento del negocio

Descripción: Al ser un proyecto centrado en el dominio del Marketing Digital, uno de los principales riesgos que se tuvieron en cuenta fue el desconocimiento del negocio por parte del equipo. En ningún momento de la carrera se le enseñó a ninguno de los integrantes nada acerca del marketing digital, un mundo amplio y con mucho para aprender.

Plan de Respuesta: El equipo se comprometió a realizar una extensa investigación acerca del dominio, de manera de aprender la mayor cantidad de detalles técnicos que puedan ser de utilidad durante el desarrollo del proyecto. Además, se pensó tener una constante comunicación con el cliente, el cual siempre se ofreció a dar una mano en lo que el equipo necesite.

Plan de Contingencia: De no ser suficiente la cantidad de investigación planificada al comienzo del proyecto, el equipo está dispuesto a dedicarle aún más tiempo a la misma, con tal de evitar que el desconocimiento del negocio sea un aspecto que perjudique el correcto desarrollo del proyecto.

Seguimiento del riesgo:

Mes	Impacto	Probabilidad	Magnitud
Marzo	5	0.9	4.8

Abril	5	0.8	4
Mayo	5	0.7	3.5
Junio	5	0.7	3.5
Julio	4	0.7	2.8
Agosto	4	0.6	2.4
Septiembre	4	0.5	2
Octubre	3	0.5	1.5
Noviembre	2	0.5	1
Diciembre	2	0.4	0.8
Enero	1	0	0

Tabla 23 - Poco conocimiento del negocio

Poco compromiso del equipo

Descripción: Un riesgo que se tomó en cuenta como principal, fue el del compromiso del equipo. A pesar de que todo el equipo estaba motivado al inicio del proyecto, siempre existe la posibilidad de que alguien pierda la motivación, o que por algún motivo no pueda dedicarle el tiempo necesario al proyecto, lo cual podría impactar fuertemente en el mismo.

Plan de Respuesta: Para evitar esto, el equipo realizó un acuerdo de compromiso entre todos los integrantes, teniendo en cuenta los objetivos e intereses personales de cada uno de ellos para con el proyecto.

Plan de Contingencia: En caso de que alguno de los integrantes pierda la motivación en algún momento, se buscaría realizar una reunión entre todos los integrantes del equipo, intentando encontrar una razón para la pérdida del mismo, y buscando soluciones para motivarlo.

Seguimiento del riesgo:

Mes	Impacto	Probabilidad	Magnitud
Marzo	2	1	2
Abril	2	1	2
Mayo	3	0.8	2.4
Junio	3	0.8	3
Julio	3	0.7	2.1
Agosto	4	0.4	1.6
Septiembre	4	0.2	0.8
Octubre	4	0	0
Noviembre	5	0	0
Diciembre	5	0	0
Enero	5	0	0

Tabla 24 - Poco compromiso del equipo

API de Google y Facebook cambiantes

Descripción: Al trabajar con las APIs de Google Ads y Facebook, un riesgo de gran impacto para el proyecto sería algún cambio grande en dichas APIs, lo cual podría significar que el equipo tenga que aumentar el esfuerzo para acomodarse al cambio.

Plan de Respuesta: El plan de respuesta frente a este riesgo es estar en constante monitoreo de las documentaciones de las APIs, cosa de que el equipo se entere lo antes posible de un futuro cambio, y acomode el sistema para soportarlo antes de que sea un problema.

Plan de Contingencia: El plan de contingencia en caso de un cambio repentino en alguna API es aceptar, y acomodar el sistema para soportar el cambio.

Seguimiento del riesgo:

Mes	Impacto	Probabilidad	Magnitud
Marzo	0	0.6	0
Abril	0	0.6	0
Mayo	0	0.6	0
Junio	5	0.6	3
Julio	5	0.6	3
Agosto	5	0.6	3

Septiembre	5	0.6	3
Octubre	5	0.6	3
Noviembre	4	0.6	2.5
Diciembre	4	0.6	2.5
Enero	2	0.5	1

Tabla 25 - APIs de Google y Facebook cambiantes

Mala gestión del proyecto

Descripción: Como el equipo nunca se había auto gestionado en un proyecto tan grande antes, otro de los riesgos principales que se tomaron en cuenta fue que la mala gestión del mismo repercuta negativamente en los objetivos del equipo. Ya sea por malas decisiones, o por malas estimaciones en las tareas.

Plan de Respuesta: Como plan de respuesta para este riesgo el equipo buscó siempre planificar con tiempo las tareas a realizar, y le dedicó tiempo a la elección de las herramientas de gestión utilizadas, cosa de poder organizarse adecuadamente siempre.

Plan de Contingencia: En caso de que el equipo considere que no se está logrando gestionar correctamente, se le va a solicitar ayuda al tutor para la gestión del proyecto.

Seguimiento del riesgo:

Mes	Impacto	Probabilidad	Magnitud
Marzo	3	0.8	2.4
Abril	3	0.8	2.4

Mayo	4	0.5	2
Junio	4	0.4	1.6
Julio	4	0.4	1.6
Agosto	4	0.3	1.2
Septiembre	4	0.3	1.2
Octubre	5	0.2	1
Noviembre	5	0.2	1
Diciembre	5	0.1	0.5
Enero	5	0	0

Tabla 26 - Mala gestión del proyecto

Mala estimación de User Stories

Descripción: Se tomó como otro riesgo principal la mala estimación de las tareas, por el motivo de que era la primera vez que el equipo tenía que encargarse de la estimación de las tareas de un proyecto tan grande, además teniendo en cuenta que no tenía el dominio para nada claro al inicio del proyecto.

Plan de Respuesta: Como plan de respuesta se decidió realizar una extensa investigación del dominio para empaparse en conocimiento técnico y lograr así una mejor estimación de las tareas. Además, el equipo siempre se dejó un margen de tiempo al estimar las tareas, por lo menos hasta acostumbrarse a las estimaciones y lograr estimar correctamente.

Plan de Contingencia: En caso de fallar con las estimaciones, se planteó reestimar las mismas, para intentar equiparar las diferencias. En el peor de los casos, se dedicaría el tiempo designado al arreglo de bugs al final del proyecto, para terminar con las funcionalidades que no se lograron estimar correctamente.

Seguimiento del riesgo:

Mes	Impacto	Probabilidad	Magnitud
Marzo	0	0	0
Abril	0	0	0
Mayo	0	0	0
Junio	0	0	0
Julio	0	0	0
Agosto	3	0.6	1.8
Septiembre	3	0.5	1.5
Octubre	3	0.2	0.6
Noviembre	3	0.2	0.6
Diciembre	3	0.2	0.6
Enero	3	0.1	0.3

Tabla 27 - Mala estimación de *user stories*

Incompatibilidad entre la arquitectura y la integración continua

Descripción: Otro riesgo de gran importancia que se tuvo en cuenta, fue el de la incompatibilidad entre la arquitectura y la integración continua, debido a que a lo largo del Proyecto el equipo se centró en que todas las funcionalidades puedan ser integradas automáticamente y sin retrasos en ese aspecto.

Plan de Respuesta: El equipo optó por planificar la arquitectura del sistema siempre pensando en la posibilidad de que el mismo tenga compatibilidad con los requisitos de la herramienta de integración continua.

Plan de Contingencia: En caso de no lograr formular una arquitectura que sea logre ser 100% automatizada en momento de desplegar nuevas funcionalidades, el equipo deberá priorizar los aspectos del sistema que, si o si necesiten contar con integración continua, y se centrará en esos.

Seguimiento del riesgo:

Mes	Impacto	Probabilidad	Magnitud
Marzo	0	0	0
Abril	0	0	0
Mayo	0	0	0
Junio	5	0.6	3
Julio	5	0.6	3
Agosto	5	0.4	2
Septiembre	5	0.2	1

Octubre	5	0.1	0.5
Noviembre	5	0	0
Diciembre	5	0	0
Enero	5	0	0

Tabla 28 - Incompatibilidad entre la arquitectura y la integración continua

13.19 Análisis y justificación de elección de herramienta de CI/CD

Bitbucket Pipelines

La herramienta proporcionada por *Atlassian* fue probada por el equipo como primera opción. Sin embargo, a pesar de haber realizado algunas pruebas de concepto satisfactorias, luego de finalizado el periodo de prueba esta solicitaba de una subscripción mensual por una cantidad de dinero no disponible para el presupuesto del proyecto.

Jenkins

Jenkins es una herramienta muy utilizada en la comunidad para la aplicación de la entrega y despliegue continuo. Es una herramienta *open source* con gran soporte en foros, fácilmente configurable y gratuita [130]. Sin embargo, el equipo no vio con buenos ojos utilizar una herramienta que necesariamente debía ser instalada como plataforma independiente en algún servidor. Si bien varios de los integrantes tenían experiencia utilizando la herramienta, muchos coincidían en que el uso de esta podía llegar a ser más compleja de lo necesario para el proyecto en cuestión. Por tal razón se descartó a la herramienta.

Circle CI

CircleCI es un proveedor en la nube, a diferencia de Jenkins, y no es necesario configurar un servidor para ejecutar los pasos de los *pipelines.* Esta fue una opción a considerar por el equipo dado que ofrecía todos los servicios buscados y no requería de gran trabajo.

Travis CI

Travis CI fue la elegida por el equipo. La principal decisión se debió a que esta estaba incluida en el paquete educativo por parte de *Github* y era una herramienta que incluso era gratuita para organizaciones y repositorios privados. Su facilidad de uso, buena interfaz e inclusión de un cli fueron importantes para su elección.

Cabe destacar que el equipo quedó muy conforme con la utilización de Travis CI como herramienta de integración y despliegue continuo. Esto no quiere decir que las otras herramientas no hubiesen servido o no eran aptas para la dimensión del proyecto, pero el equipo tuvo que elegir una y su decisión fue acertada.

13.20 Elementos de configuración de template de desarrollo genérico.

Los elementos genéricos para el desarrollo de un proyecto del sistema fueron los siguientes;

- *pom.xml:* Es el archivo de configuración de *maven*, y además de la importancia de este a nivel de dependencias, dentro de su configuración se implementaron *plugins* para la ejecución de *gitflow* y liberación de versiones al momento de los merge hacia la rama master.
- travis.yml: Es el archivo de configuración necesario para ejecutar el *pipeline* de Travis CI. En este se detalla el sistema operativo de la máquina virtual para la ejecución, el lenguaje de ejecución, los pasos de pruebas, pre-despliegue y despliegue y condicionales para su accionar, entre otros. Este archivo es obligatorio para disparar cualquier proceso requerido con Travis.
- build.sh, clean.sh, deploy.sh, run.sh: Estos son *scripts* que son ejecutados dentro del archivo de configuración de travis.yml. En estos se encuentran las acciones a realizar para la compilación, prueba, despliegue y ejecución de la aplicación.
- rpm: Los *scripts* dentro de la carpeta rpm son aquellos necesarios para la generación de un paquete de Linux de forma de realizar la transferencia hacia el servidor destino en Amazon.
- *Checkstyle* y *Spotbugs*: Son archivos de reglas de estilo de Código y bugs. En caso de que alguno de estos falle el código no compilará y el *build* no pasará.

13.21 Evidencia de elementos de software

Github

En Github se creó una organización privada por la universidad llamada ORTConecta361. Dentro de estos fueron creados los diferentes repositorios con sus respectivos códigos fuente a lo largo del proyecto. Estos son:

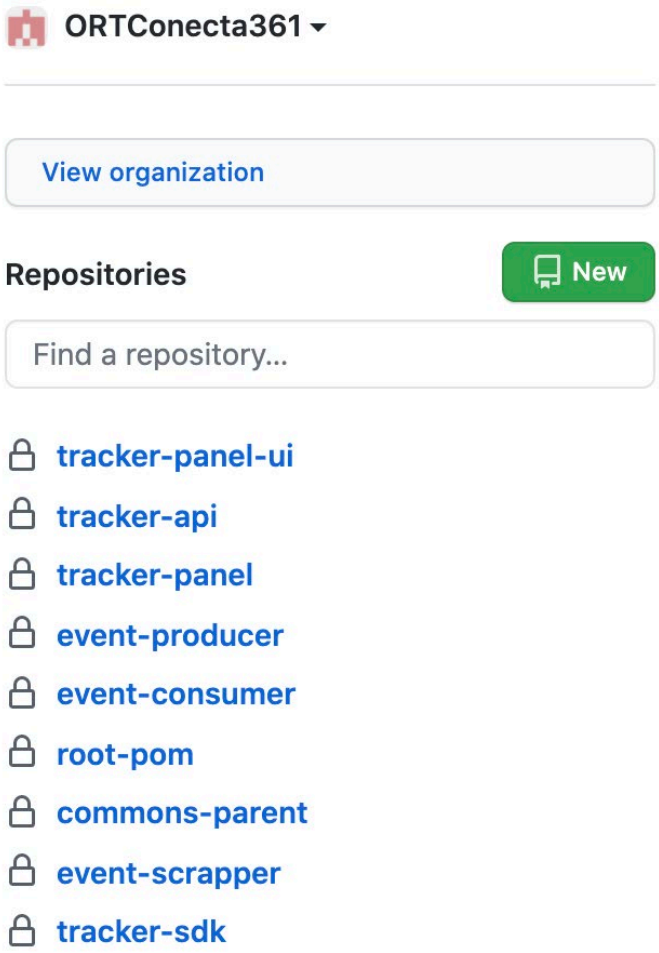


Ilustración 98 - Evidencia de elementos de software

13.22 Evidencia de librerías

Root-pom

```
<!-- Dependencies versions -->
<spring-boot.version>2.3.0.RELEASE</spring-boot.version>
<mysql-connector.version>8.0.20</mysql-connector.version>
<commons-lang3.version>3.10</commons-lang3.version>
<commons-text.version>1.8</commons-text.version>
<commons-io.version>2.6</commons-io.version>
<commons-validator.version>1.6</commons-validator.version>
<commons-net.version>3.6</commons-net.version>
<poi.version>4.1.2</poi.version>
<aws-java-sdk.version>1.11.786</aws-java-sdk.version>
<aws-secretsmanager-jdbc.version>1.0.5</aws-secretsmanager-jdbc.version>
<gson.version>2.8.6</gson.version>
<mapstruct.version>1.3.1.Final</mapstruct.version>
<okhttp.version>4.6.0</okhttp.version>
<guava.version>29.0-jre</guava.version>
<joda-time.version>2.10.6</joda-time.version>
<java-string-similarity.version>1.2.1</java-string-similarity.version>
<freemarker.version>2.3.30</freemarker.version>
<newrelic-agent.version>5.12.0</newrelic-agent.version>
<redisson.version>3.12.5</redisson.version>
<logstash-logback.version>6.3</logstash-logback.version>
<sendgrid.version>4.4.8</sendgrid.version>
<bouncycastle.version>1.65</bouncycastle.version>
<httpclient.version>4.5.12</httpclient.version>
<slf4j.version>1.7.30</slf4j.version>
<sonar-maven-plugin.version>3.7.0.1746</sonar-maven-plugin.version>
<shedlock.version>4.9.2</shedlock.version>
<distributed-lock-redis.version>1.3.0</distributed-lock-redis.version>
<jedis.version>3.3.0</jedis.version>
<lombok.version>1.18.12</lombok.version>
<commons-csv.version>1.8</commons-csv.version>
<commons-beanutils.version>1.9.4</commons-beanutils.version>
<cache2k.version>1.2.4.Final</cache2k.version>
<jsoup.version>1.13.1</jsoup.version>
<jwt.version>0.9.1</jwt.version>
<libphonenumber.version>8.12.3</libphonenumber.version>
<spring-mobile-device.version>1.1.5.RELEASE</spring-mobile-device.version>
<hazelcast.version>4.0.1</hazelcast.version>
<hazelcast-aws.version>3.2</hazelcast-aws.version>
<lovelace.version>Lovelace-SR16</lovelace.version>
<dynamodb-sdk.version>1.11.64</dynamodb-sdk.version>
<spring-dynamodb.version>5.1.0</spring-dynamodb.version>
```

```
<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-java-sdk-sqs</artifactId>
    <version>${aws-java-sdk.version}</version>
</dependency>
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>${gson.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-lang3</artifactId>
    <version>${commons-lang3.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-text</artifactId>
    <version>${commons-text.version}</version>
</dependency>
<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>${commons-io.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi</artifactId>
    <version>${poi.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi-ooxml</artifactId>
    <version>${poi.version}</version>
</dependency>
<dependency>
    <groupId>commons-validator</groupId>
    <artifactId>commons-validator</artifactId>
    <version>${commons-validator.version}</version>
</dependency>
<dependency>
    <groupId>commons-net</groupId>
    <artifactId>commons-net</artifactId>
    <version>${commons-net.version}</version>
</dependency>
<dependency>
    <groupId>org.mapstruct</groupId>
    <artifactId>mapstruct</artifactId>
    <version>${mapstruct.version}</version>
</dependency>
<dependency>
    <groupId>com.squareup.okhttp3</groupId>
    <artifactId>okhttp</artifactId>
    <version>${okhttp.version}</version>
</dependency>
<dependency>
    <groupId>com.google.guava</groupId>
    <artifactId>guava</artifactId>
    <version>${guava.version}</version>
</dependency>
<dependency>
    <groupId>joda-time</groupId>
    <artifactId>joda-time</artifactId>
    <version>${joda-time.version}</version>
</dependency>
```

Ilustración 99 - Evidencia de librerias

13.23 Evidencia de elementos de infraestructura

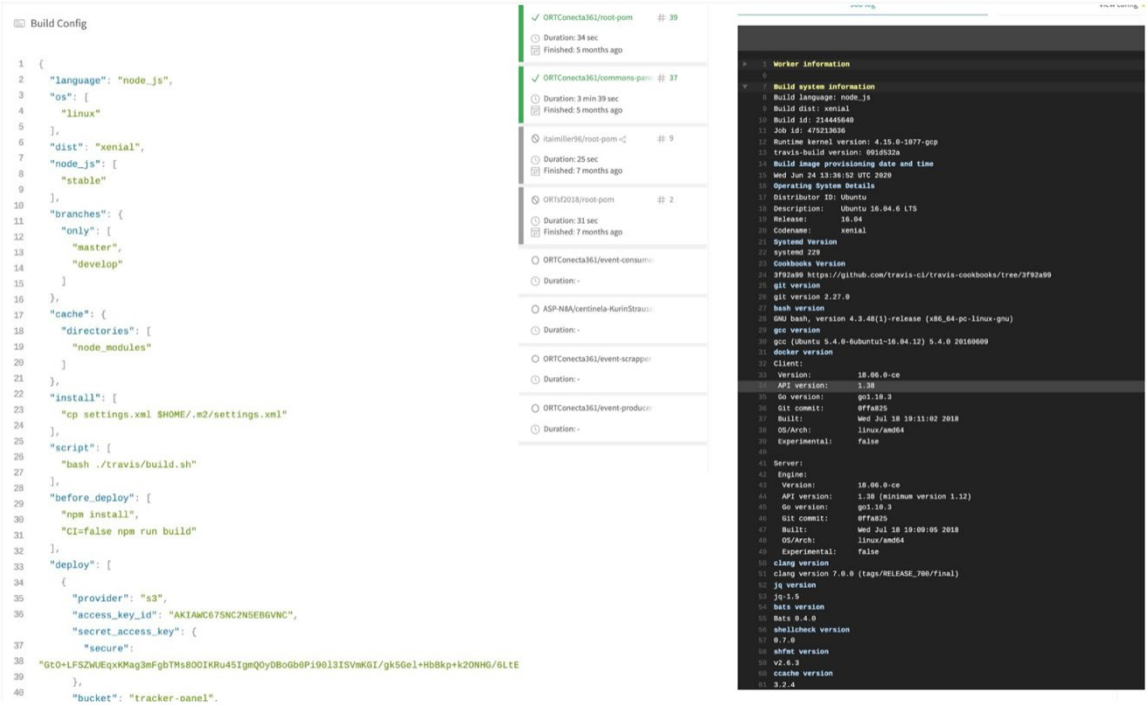


Ilustración 100 - Travis CI IaC

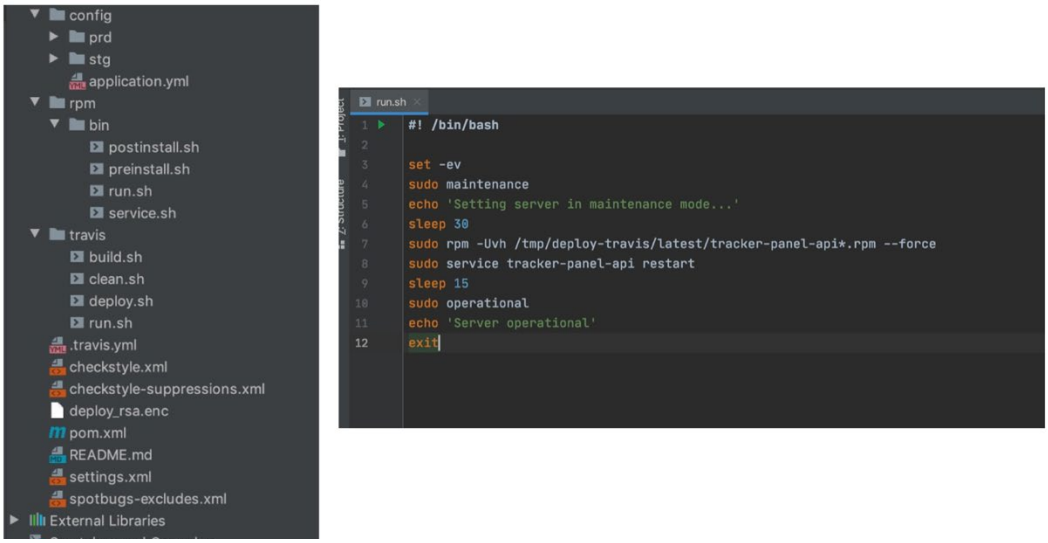


Ilustración 101 - IaC *scripts*

13.24 Evidencia de elementos de documentación

Google Drive

Se creó una carpeta compartida en *Google Drive* para almacenar todos los elementos de configuración que hacen referencia a la documentación. La estructura que posee es la siguiente:

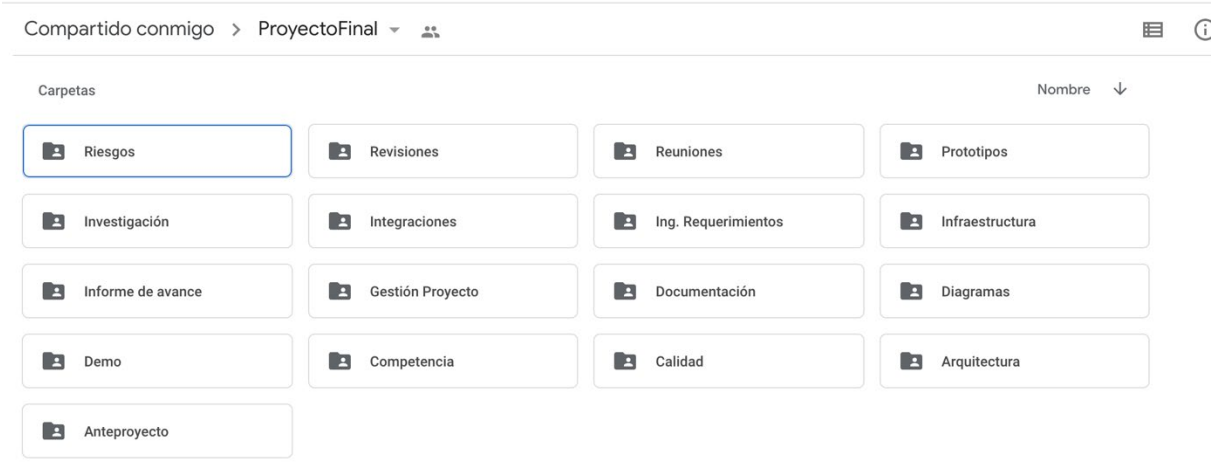


Ilustración 102 - Documentación en Google Drive

OneDrive

Para la documentación de utilizó Microsoft Word y OneDrive como repositorio en línea.

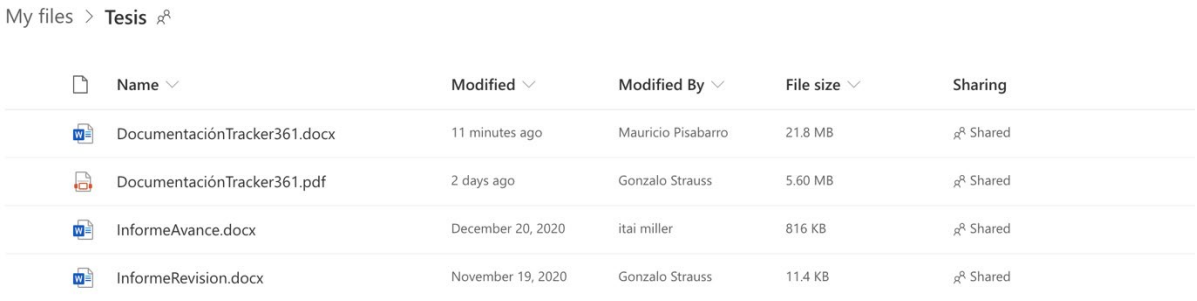


Ilustración 103 - Documentación en OneDrive

13.25 Gitflow

Gitflow es la metodología utilizada a lo largo del desarrollo de software para la gestión de cambios.

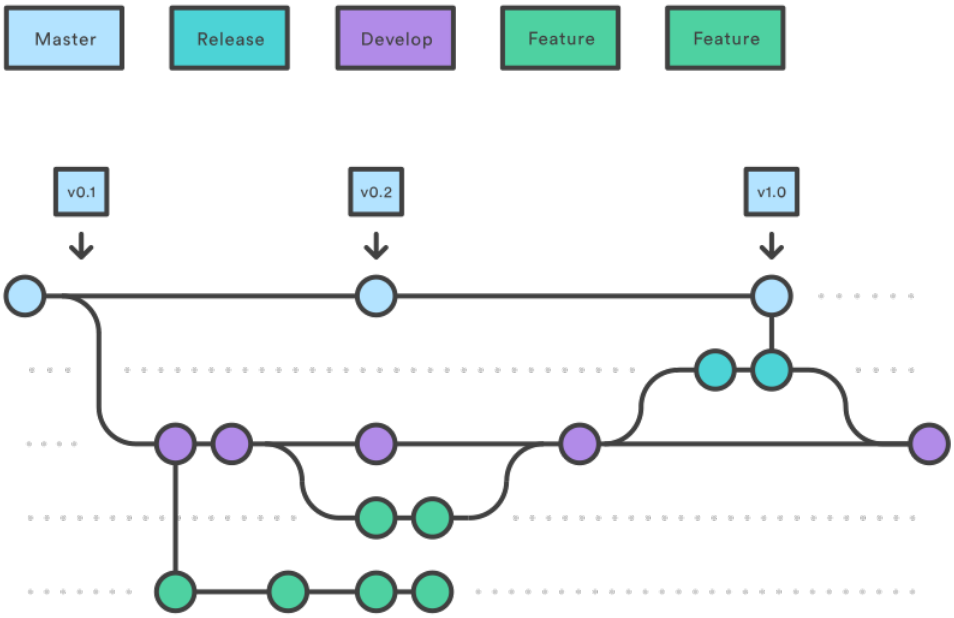


Ilustración 104 – Gitflow [144]

13.26 Evidencia de versionado

Maven

Con la utilización de *Maven*, los distintos componentes del proyecto definen su versionado dentro de sus respectivos pom.xml. En este definen dos tipos distintos, la versión local, que incluye la palabra *snapshot* y la versión productiva, que es la versión actualmente funcionando (es una versión anterior a la local).

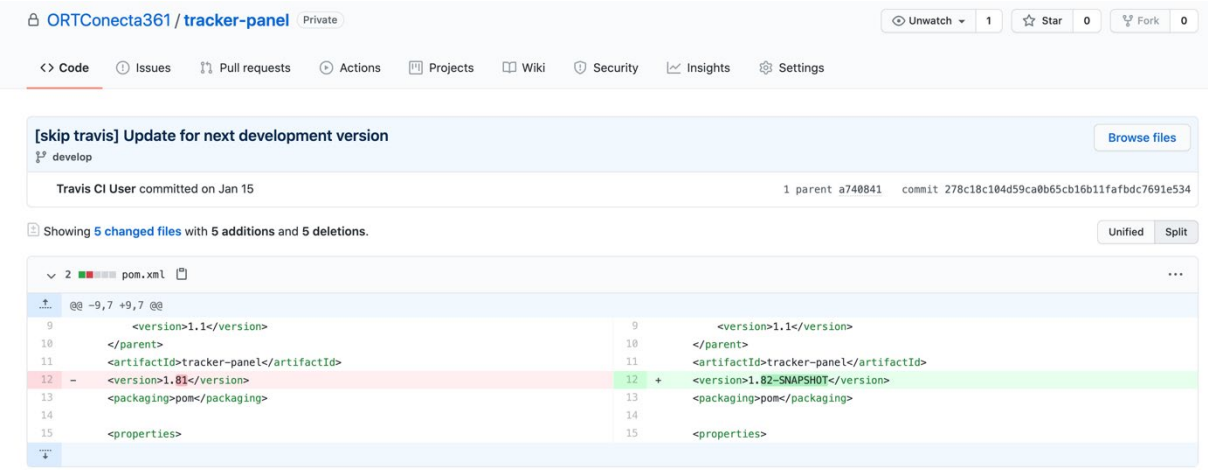


Ilustración 105 - Versionado de Maven

Github

Travis CI utiliza *Gitflow* para la integración de los releases y a través de un *plugin* de maven estos son generados con un tag definido como: <versión>.

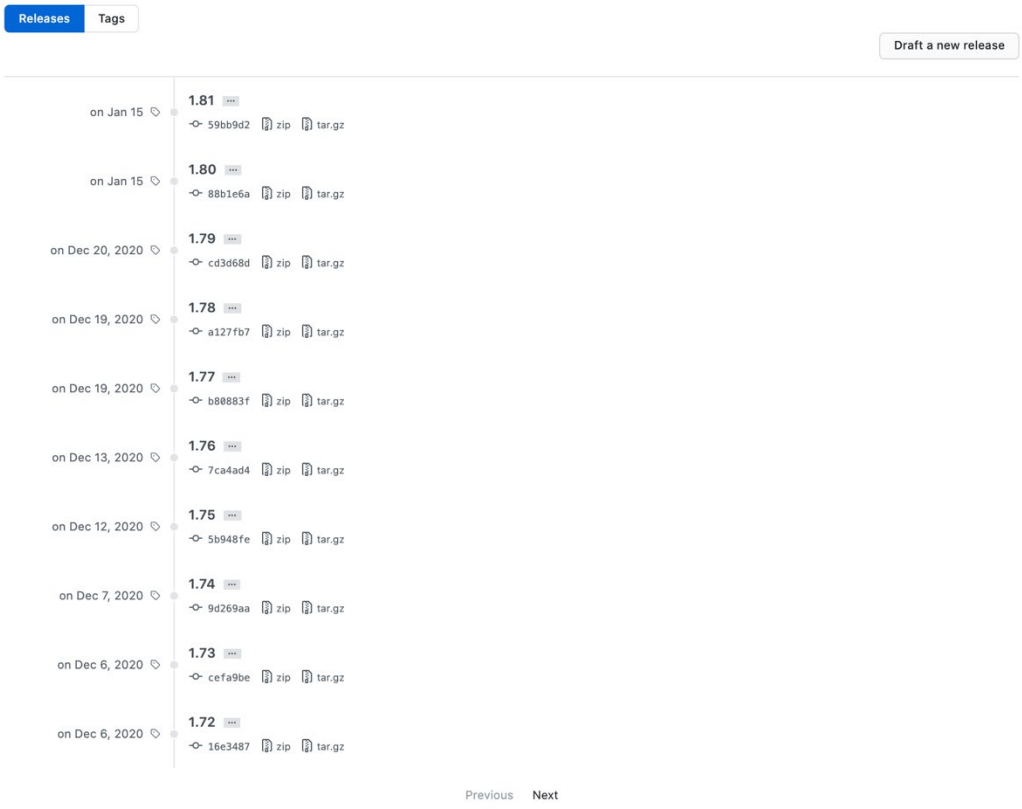


Ilustración 106 - Tags de *Github*

Nexus Repository

Evidencia de versionado de los diferentes componentes del sistema a través de Nexus:

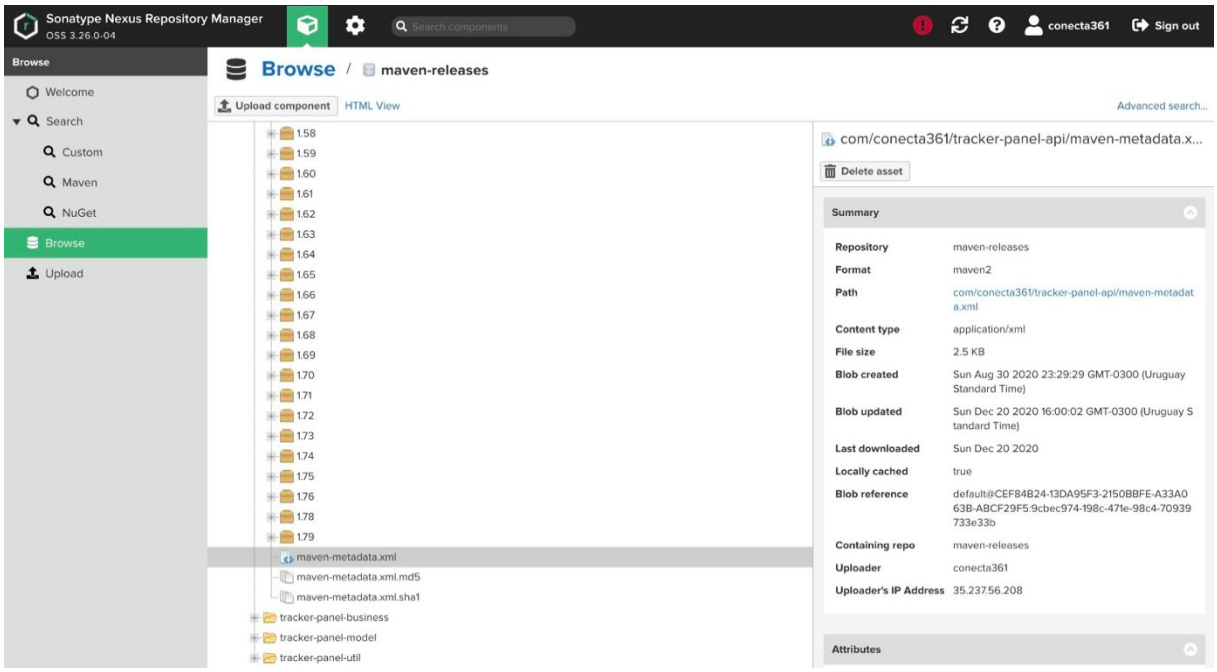


Ilustración 107 - *Releases* y *spanpshots* de *Nexus Repository*

13.27 Sincronización de Github con Jira y CI

Github y Jira

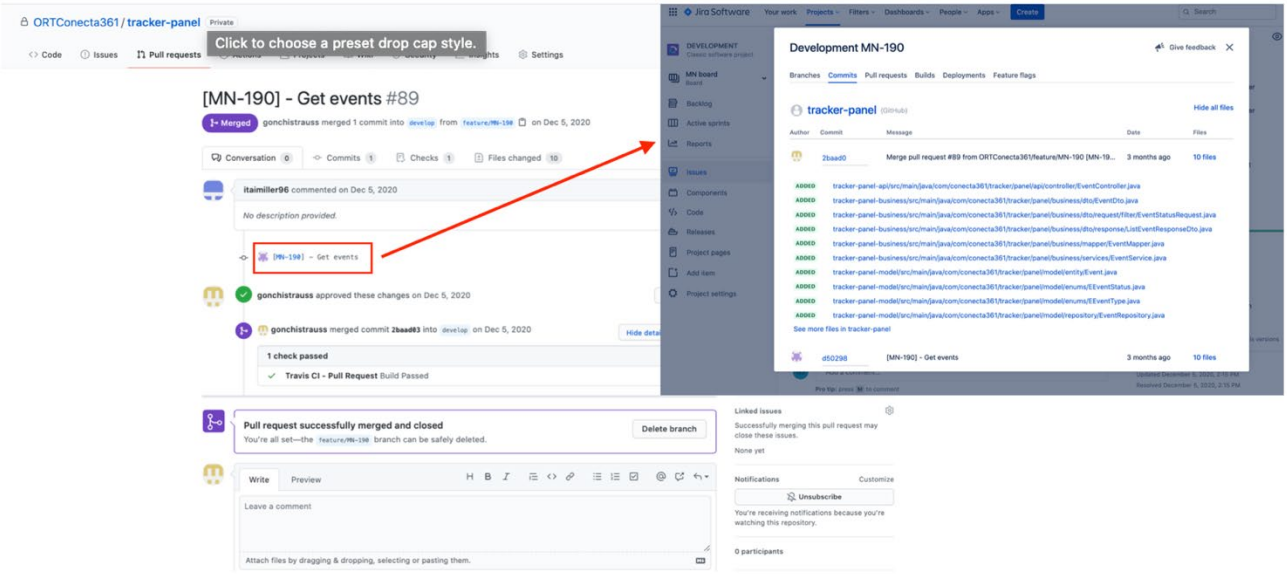


Ilustración 108 - Sincronización *Github* y *Jira*

Github y *Travis CI*

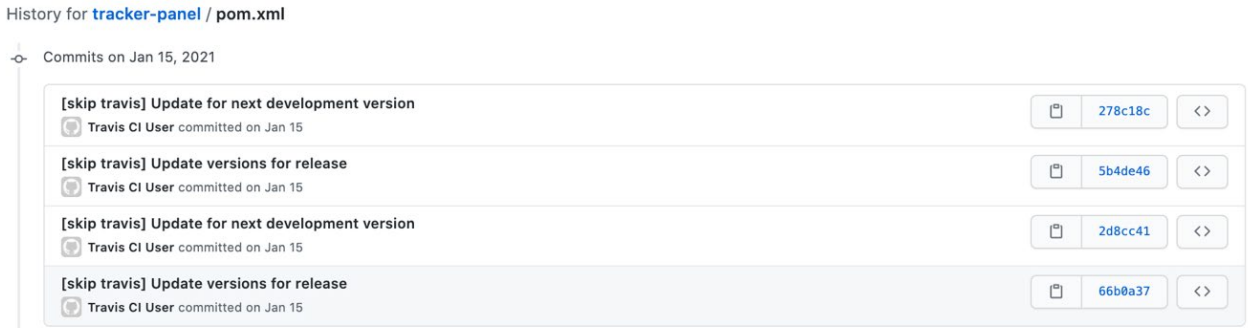


Ilustración 109 - Integración Github y Travis CI

13.28 Plan de calidad

Fase	Actividad	Entradas	Salidas	Herramientas y Tecnología	Roles encargado	Roles participantes
Investigación	*Estudiar el problema*	*Visualización de potencial problema y confirmación de su existencia*	*Problema del Proyecto*	*Lectura de articulos y documento y reuniones con product owner, zoom y Microsoft Teams*	*Equipo*	*Equipo + Product Owner*
	Buscar soluciones	*Problema del proyecto*	*Soluciones candidatas*	*Ingeniería inversa, lluvia de ideas, wireftames, bocetos, prototipos, artículos, videos de la temática y tutoriales, Zoom y Microsoft Teams*	*Equipo*	*Equipo*
	Estudiar las herramientas tecnológicas	*Problema definido*	*Análisis comparativo de teconologias, documentación con eleccion de tecnologías y justificación*	*Internet, experiencia del equipo*	*Equipo*	*Equipo*
	Capacitación de nuevas tecnologías	*Tecnologías definidas*	*Aprendizaje de las tecnologías*	*Onboarding tecnológico, tutoriales y guías de los principales foros relacionados*	*Equipo*	*Equipo*
Ingenieria de Requerimientos	*Identifcar requerimientos*	*Solución del proyecto*	*Definicion de requerimientos en forma de user stories*	*Kanban, Jira, relevamiento con el product owner*	*Equipo*	*Equipo*
	Priorizar requerimientos	*Lista de requerimientos*	*Lista de requerimientos priorizada*	*Trade-offs de cada requerimiento*		*Equipo*
	Estimar requerimientos	*Lista de requerimientos priorizados*	*Lista de requerimientos priorizada y estimada*	*Planning poker y Jira*		*Equipo*

	Definir el alcance inicial	*Lista de requerimientos priorizada y estimada*	*Product Backlog*	*Jira*	*IR*	*Equipo*
	Validación alcance	*Product Backlog*	*Requerimientos validados*	*Tutor, prototipos en Axure y papel*	*IR*	*IR*
	Especificación de requerimientos	*Requerimientos validados*	*Especificación de requerimientos*	*Jira*		
	Validar especificación	*Especificación de requerimientos*	*Especificación de requerimientos validados*	*Reuniones de equipo + Product owner*		
Analisis y validación de solución propuesta	*Investigación del problema a simular*	*Problema del proyecto*	*Problema del proyecto*	*Extractos, API de Facebook, API de MercadoPago, Postman*	*Equipo*	*Equipo*
Diseño y Arquitectura	*Análisis de requerimientos no funcionales*	*Requerimientos no funcionales*	*Atributos de calidad y tácticas de arquitectura*	*Artículos, y reunion con tutor (experto en arquitectura)*	*Equipo*	*Equipo*
	Análisis de posibles soluciones de arquitectura y diseño	*Requerimientos no funcionales, atributos de calidad, tácticas de arquitectura, product backlog*	*Diagramas de arquitectura y diseño*	*Diagramas UML*	*Arquitecto*	*Arquitecto, Equipo*
	Diseño de la arquitectura	*Diagramas*	*Documento de arquitectura*	*Consulta con tutor*		
	Revisión, validación y verificación de la arquitectura	*Documento de arquitectura*	*Arquitectura validada, verificada y revisada*	*Revisión con tutor*		
	Diseño de la solución	*Requerimientos funcionales*	*Documento de diseño*	*Diagramas UML*		

Construcción	*División de tareas*	*Arquitectura, Diseño, Tecnologías, Gestión*	*Roles con responsabilidades definidas*	*Reunión de equipo*	*Equipo*	*Equipo*
	Plan de Prueba	*Especificación de requerimientos y Product Backlog*	*Pruebas a ejecutar*	*Código fuente, Travis*	*SQA*	*SQA*
	Desarrollo de Tareas	*Diseño, Arquitectura, Product Backlog*	*Código fuente*	***Java:** IDE: Intelij J Framework: Spring, Pruebas: JUnit, Linters: Spotbugs, CheckStyle. **React**: IDE: Visual Studio Code, Checkstyle: eslint*	*Equipo*	*Equipo*
	Revisión de código	*Código fuente*	*Código fuente corregido*	*Código fuente, estándares de código*	*SQA*	*Equipo*
	Pruebas unitarias	*Casos de Prueba, Criterios de aceptación, requerimientos*	*Pruebas implementadas.y ejecutadas correctamente*	*JUnit*	*Equipo*	*Equipo*
	Pruebas de integración	*Pruebas a ejecutar*	*Pruebas ejecutadas correctamente y lista de incidentes*	*Postman, JUnit, Jira*	*Equipo*	*Equipo*
Pruebas	*Pruebas de Usabilidad*	*Especificación de requerimientos*	*Resultados de las pruebas*	*Heurísticas de Nielsen, pruebas con usuarios*	*SQA*	*Equipo*
	Pruebas de validación	*Wireframes, Mockups, Prototipos*	*Feedback, sugerencias de cambios*	*Lápiz y papel, AxureRP*		

Ilustración 110 - Plan de calidad

13.29 Estándares de codificación

A continuación, se detallan los estándares de los lenguajes de programación utilizados en el proyecto.

Lenguaje	Estándar utilizado
Java Spring	https://spring.io/
ReactJS	https://reactjs.org/
HTML	https://www.w3schools.com/html/html5_syntax.asp
CSS	https://www.w3schools.com/css/default.asp
API REST	https://restfulapi.net/resource-naming/

Ilustración 111 - Estándares de codificación