Universidad ORT Uruguay

Facultad de Ingeniería

Fitit Returns

Sistema centralizado para devoluciones

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

Nicolás Andreoli - 210630

Florencia Batlle - 239392

Federico Czarnievicz - 201250

Tutor: Leonardo Scafarelli

2025

Declaración de Autoría

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

Nicolás Andreoli
14/10/2025

Florencia Batlle
14/10/2025

Federico Czarnievicz
14/10/2025

Agradecimientos

Este proyecto representa el cierre de una etapa muy significativa para nosotros. Más allá de ser el trabajo final de carrera, simboliza años de aprendizaje, colaboración y crecimiento compartido.

Queremos agradecer especialmente a nuestras familias y amigos, por acompañarnos durante todo este proceso, por su apoyo constante y por entender las horas de trabajo y las exigencias que implicó este proyecto. Su paciencia y confianza fueron fundamentales para que pudiéramos llegar hasta el final.

Agradecemos también a la Universidad ORT Uruguay y a todos los docentes que nos guiaron a lo largo de estos años, brindándonos las herramientas necesarias para desarrollarnos como profesionales. En particular, queremos destacar el acompañamiento de Leonardo Scafarelli, nuestro tutor, por su compromiso, orientación y buena disposición durante todo el desarrollo de Fitit Returns.

Extendemos nuestro agradecimiento a todas las personas que participaron en pruebas, entrevistas o validaciones, aportando ideas y observaciones que nos ayudaron a mejorar la plataforma. Su tiempo y colaboración hicieron posible que el resultado final fuera más completo y útil.

Finalmente, agradecemos al equipo de Fitit, especialmente a Daniel Bella, Dan Bzurovski y Alan Rytt, por la confianza y el apoyo brindado desde el comienzo, que nos permitió aplicar lo aprendido en un proyecto real y desafiante.

A todos los que de alguna forma fueron parte de este proceso, muchas gracias.

Abstract

Fitit Returns es un sistema web desarrollado para optimizar la gestión de devoluciones y cambios en tiendas de indumentaria. Su objetivo principal es mejorar la experiencia de postventa tanto para las tiendas como para los usuarios finales, reduciendo la carga administrativa, los tiempos de resolución y las pérdidas económicas asociadas a errores o demoras en el proceso.

Previo al desarrollo del sistema, las devoluciones se gestionaban de manera manual a través de canales informales, lo que dificulta el seguimiento, la trazabilidad y la generación de métricas. Fitit Returns surge como una solución integral que permite digitalizar y automatizar este flujo, brindando transparencia, control y eficiencia en cada etapa.

El sistema está compuesto por una aplicación web de cara al usuario, donde los consumidores pueden iniciar y monitorear solicitudes de devolución, y por un panel de gestión para tiendas, que permite revisar, aprobar o rechazar solicitudes, administrar políticas de devolución y emitir créditos de tienda. Además, incluye un panel interno de backoffice para la administración central de Fitit y la gestión de configuraciones globales.

Desde el punto de vista técnico, el sistema fue desarrollado sobre una arquitectura modular y escalable, utilizando React.js y Next.js para el *frontend*, y un *backend serverless* basado en Firebase *Cloud Functions.* A su vez, existen otros elementos importantes como Credits API, desarrollada en Node.js, y otros servicios que utilizan Google Pub/Sub, Firestore y MySQL.

El proyecto se desarrolló siguiendo un ciclo de vida incremental–iterativo, acompañado de metodologías ágiles como Scrum y Kanban. Estas permitieron mantener un ritmo de trabajo sostenido, entregar valor en cada sprint y adaptarse con agilidad a cambios de requerimientos, como la incorporación del sistema de créditos para tiendas a mitad del proceso.

A lo largo del desarrollo se definieron planes de calidad, gestión de configuración y mitigación de riesgos, además de estándares de codificación y validaciones automatizadas que aseguraron la consistencia del código y la estabilidad del sistema.

Actualmente, Fitit Returns se encuentra completamente implementado y en un entorno pre-productivo. El sistema constituye una herramienta clave para mejorar la eficiencia operativa, fortalecer la relación entre Fitit y sus clientes y sentar las bases para la expansión internacional del producto.

Palabras clave

E-commerce; Fitit; Fitit Returns; Cloud Functions; Usuario; Tienda; Cadeterías; Backoffice; Next.js; Node.js; SQL; NoSQL; Cloud Functions; Serverless; GCP; AWS; Kanban; Scrum.

Glosario

***API*:** Interfaz de Programación de Aplicaciones. Conjunto de definiciones y protocolos que permiten que diferentes sistemas o servicios se comuniquen entre sí. Facilita la integración y el intercambio de datos entre aplicaciones.

Arquitectura: Estructura y organización de los componentes de un sistema, así como las relaciones entre ellos. En *software*, define cómo se dividen las responsabilidades, cómo se comunican los módulos y qué tecnologías se emplean.

***Backend*:** Parte del sistema que gestiona la lógica del negocio, las bases de datos y las operaciones que no son visibles para el usuario final. Suele desarrollarse con lenguajes como Node.js, Java o Python, y se comunica con el frontend mediante APIs.

***Benchmark*:** Proceso de comparar el rendimiento, calidad o eficiencia de un sistema, producto o proceso frente a estándares o competidores del mercado, con el fin de identificar fortalezas y áreas de mejora.

Devolución: Proceso mediante el cual un cliente retorna un producto adquirido, generalmente por insatisfacción, error de talla o defecto. En el contexto de e-commerce, implica coordinación logística y gestión de reembolsos o cambios.

***E-commerce*:** Comercio electrónico. Actividad de comprar y vender productos o servicios a través de internet, mediante plataformas digitales o tiendas en línea.

***Frontend*:** Parte visible e interactiva de una aplicación con la que el usuario final interactúa. Se desarrolla utilizando tecnologías como HTML, CSS y JavaScript, y su objetivo es ofrecer una experiencia de usuario fluida y atractiva.

***HTTPS*:** Acrónimo de HyperText Transfer Protocol Secure. Protocolo de comunicación segura en internet que utiliza cifrado SSL/TLS para proteger los datos transmitidos entre el cliente y el servidor.

Indumentaria: Conjunto de prendas de vestir o ropa. En el contexto de e-commerce, hace referencia al sector o categoría de productos relacionados con moda, vestimenta y accesorios.

***MVP*:** Acrónimo de Minimum Viable Product. Versión inicial de un producto que incluye las funcionalidades mínimas necesarias para validar su propuesta de valor con usuarios reales, obteniendo retroalimentación temprana para futuras iteraciones.

***Release*:** Versión publicada o liberada de un producto o *software* que está lista para ser distribuida o utilizada por los usuarios. Puede referirse a una entrega parcial, una actualización o una versión final.

***Responsive*:** Característica del diseño web que permite que una aplicación o sitio se adapte automáticamente a diferentes tamaños de pantalla o dispositivos (móviles, tablets, computadoras), garantizando una experiencia de usuario consistente.

***SDK*:** Acrónimo de Software Development Kit. Conjunto de herramientas, librerías y documentación que facilita a los desarrolladores la creación de aplicaciones o la integración con un servicio o plataforma específica.

***Stakeholders*:** Personas o grupos que tienen interés o están involucrados en un proyecto, producto o empresa. Incluyen usuarios, clientes, inversores, empleados y cualquier parte afectada por las decisiones o resultados.

***Startup*:** Empresa emergente con un modelo de negocio innovador y un alto potencial de crecimiento, que suele encontrarse en una fase inicial de desarrollo y busca validar su propuesta en el mercado.

***TDD*:** Acrónimo de Test Driven Development. Metodología de desarrollo en la que se escriben primero las pruebas automáticas antes del código de la funcionalidad. Promueve un diseño más claro, código más mantenible y menor cantidad de errores.

***UML*:** Acrónimo de Unified Modeling Language. Lenguaje estandarizado para visualizar, especificar, construir y documentar los elementos de un sistema de *software* mediante diagramas estructurados.

Índice

Declaración de Autoría 2
Agradecimientos 3
Abstract 4
Palabras clave 5
Glosario 6
Índice 8
1. Introducción 14
1.1. Entorno conceptual 14
1.2. Descripción del cliente 15
1.3. Objetivos generales 16
1.3.1. Objetivos académicos 16
1.3.2. Objetivos del producto 17
1.3.3. Objetivos del proyecto 17
1.4. Integrantes del equipo y motivación para realizar este proyecto 17
2. Planteamiento del problema 19
2.1. Contexto del problema 19
2.1.1. Desde el punto de vista de las tiendas 19
2.1.2. Desde el punto de vista del usuario final 19
2.1.3. El desafío para Fitit 20
2.2. Actores principales y necesidades 20
3. Descripción de la solución 21
3.1. Descripción funcional del producto 21
3.1.1. Vista del cliente 21
3.1.2. Vista de la tienda 23
3.1.3. Vista del usuario final 28
4. Ciclo de vida 34
4.1. Características del proyecto 34
4.2. Roles definidos en el equipo 35
4.3. Fases del proyecto 35
4.3.1. Fase de Descubrimiento 37
4.3.2. Fase de Desarrollo 38
4.3.3. Fase de Documentación 38
5. Gestión de proyecto 38
5.1. Cronograma 39

5.2. Análisis del esfuerzo 39
5.3. Fase de descubrimiento 40
5.3.1. Adaptación de Kanban 41
5.3.2. Distribución del esfuerzo 42
5.4. Fase de desarrollo 43
5.4.1. Roles de Scrum 44
5.4.2. Artefactos 44
5.4.3. Eventos 47
5.4.4. Ejecución del *Release Plan* 49
5.4.5. Métricas 51
5.4.6. Distribución de esfuerzo 52
5.5. Fase de documentación 53
5.5.1. Adaptación de Kanban 54
5.5.2. Distribución del esfuerzo 54
5.6. Comunicación y coordinación del equipo 55
5.7. Gestión del riesgo 55
5.7.1. Identificación y análisis de riesgos 56
5.7.2. Monitoreo de riesgos principales 56
6. Ingeniería de requerimientos 61
6.1. *Design Thinking* 61
6.2. Relevamiento 63
6.2.1. Empatizar 63
6.3. Análisis 68
6.3.1. Definir 69
6.4. Especificación 71
6.4.1. Idear 71
6.5. Validación 75
6.5.1. Prototipar 75
6.5.2. Probar 76
6.6. Gestión del cambio 78
6.6.1. Nuevos requerimientos 78
6.7. Requerimientos funcionales 80
6.7.1. Usuario 80
6.7.2. Tienda 80
6.7.3. Fitit 81
6.7.4. Integraciones con terceros 81
6.8. Requerimientos no funcionales 82

6.9. Restricciones 85
6.9.1. Tecnologías 85
6.9.2. Integración con la Herramienta Actual 85
6.9.3. Usabilidad y Diseño 85
7. Diseño arquitectónico 87
7.1. Arquitectura a Alto Nivel 88
7.1.1. Representación Primaria 88
7.1.2. Catálogo de Elementos 89
7.2. Elección de Tecnologías 90
7.2.1. *Frontend* 90
7.2.2. *Backend* 91
7.2.3. Base de Datos 91
7.2.4. Cola de Mensajes 92
7.2.5. Proveedor en la Nube 92
7.3. Vista de Módulos 93
7.3.1. Vista de Capas 93
7.3.2. Vista de Descomposición 95
7.3.3. Vista de Clases 97
7.4. Vistas de Infraestructura 100
7.4.1. Representación Primaria 100
7.4.2. Catálogo de Elementos 101
7.5. Patrones de Diseño 101
7.5.1. Patrón Strategy 101
7.5.2. Publish/Subscribe 102
7.5.3. Federated Identity Pattern 103
7.6. Evolución de la arquitectura a lo largo del tiempo 104
7.6.1. Versión Inicial 104
7.6.2. Versión Final 104
8. Aseguramiento de la calidad 106
8.1. Objetivos de calidad 106
8.2. Plan de calidad 107
8.3. Aseguramiento de calidad 108
8.3.1. Estándares 108
8.3.2. Pruebas 110
8.3.3. Gestión de *bugs* 117
8.3.4. Revisiones 118
8.3.5. Validaciones 119

8.3.6. Métricas de calidad 120
8.4. Distribución del esfuerzo 125
8.4.1. Análisis general 126
9. Gestión de la configuración 127
9.1. Identificación de elementos de la configuración 127
9.1.1. Código 127
9.1.2. Metodologías 127
9.1.3. Documentación 128
9.2. Herramientas de gestión 128
9.2.1. Git y GitHub 128
9.2.2. GitHub Actions 129
9.2.3. ClickUp 129
9.2.4. Notion 129
9.2.5. Jira 129
9.2.6. Clockify 129
9.2.7. Google Docs 130
9.2.8. Google Slides 130
9.2.9. Escalidraw 130
9.2.10. Draw.io 130
9.2.11. Figma 130
9.2.12. WhatsApp 131
9.2.13. Slack 131
9.2.14. Google Meet 131
9.3. Gestión del repositorio 132
9.3.1. *Monorepo* 132
9.3.2. Gestión de dependencias 133
9.3.3. Estrategia de *branching* 134
9.3.4. *Branch protection rules* 134
9.3.5. Integración continua 135
9.4. Gestión de versiones 135
10. Conclusiones 137
10.1. Estado actual 137
10.2. Sobre los objetivos 138
10.2.1. Objetivos académicos 138
10.2.2. Objetivos del producto 138
10.2.3. Objetivos del proyecto 139
10.3. Cierre del proyecto con el cliente 139

10.4. Lecciones aprendidas 140
10.4.1. Gestión del alcance y estrategia de entregas 140
10.4.2. Producto y descubrimiento continuo 141
10.4.3. Arquitectura y decisiones técnicas 141
10.4.4. Calidad, pruebas y herramientas 141
10.4.5. Experiencia de usuario y accesibilidad 141
10.4.6. Proceso y colaboración 141
10.4.7. Gestión del tiempo y foco 141
10.4.8. Documentación viva 142
10.5. Reflexiones personales 142
10.6. Cierre final 143
11. Bibliografía 145
12. Anexo 150
12.1. Pantallas de *backoffice* 150
12.2. Pantallas en *mobile* 152
12.3. Product Backlog 155
12.4. Funcionalidades por *release* y descartadas 156
12.5. Evolución en la UI de pantallas principales 159
12.6. *Release Plan* 162
12.7. Escalas y metodología de análisis de riesgos. 163
12.8. Análisis cualitativo de riesgos 165
12.9. *Design Thinking* 167
12.9.1. Resultados de encuestas a usuarios 167
12.9.2. Email de devolución Renner 173
12.9.3. Preguntas para entrevista con tiendas 174
12.9.4. User Journey Map del proceso actual de devolución 175
12.9.5. Saturar 176
12.9.6. Agrupar 179
12.9.7. User Personas 182
12.9.8. Votación para priorizar ideas 183
12.9.9. Prototipo de baja fidelidad 184
12.9.10. Prototipos de alta fidelidad 185
12.10. Épicas 187
12.11. Resumen de la ingeniería de requerimientos 189
12.12. Proceso de gestión de cambios 191
12.13. Gráfica de descargas npm next vs. nuxt 192
12.14. Tabla comparativa tecnologías Credits API 193

12.15. Plan de calidad 194
12.16. Definición de estándares de codificación 198
12.17. Configuración y aplicación de estándares 202
12.18. Casos de pruebas funcionales 204
12.19. Pruebas de carga con k6 209
12.20. Pruebas de usabilidad 210
12.20.1. Aplicación web de devolución para usuarios finales 210
12.20.2. Panel web para tiendas 213
12.20.3. Sesiones externas (UserBrain) 215
12.21. Formato de registro de *bugs* en Jira 217
12.22. Detalle de *bugs* por iteración 219
12.23. Aplicación de Heurísticas de Nielsen 223
12.24. Configuración de package.json 224
12.24.1. Extracto de package.json 224
12.24.2. Github actions 226
12.25. Formulario inicial de devolución manual en fitit-store 227
12.26. Evidencia de pruebas en Cypress 228
12.27. Escalidraw 229
12.27.1. Diagrama de paquetes de *backend* en Escalidraw 229
12.27.2. Diagrama de paquetes de *frontend* en Escalidraw 230
12.28. Diseño y estilos de Fitit 231
12.29. Reunión en oficina de Fitit 233

1. Introducción

El presente documento corresponde al trabajo de tesis de la carrera de Ingeniería en Sistemas de la Universidad ORT Uruguay y tiene como objetivo documentar el desarrollo de **Fitit Returns**, una solución tecnológica orientada a optimizar los procesos de devolución y cambio de productos en el comercio electrónico (*e-commerce*) de moda.

Este proyecto se enmarca en un contexto académico, pero se desarrolla en colaboración con un cliente real, lo que permite abordar un problema concreto del mercado con una perspectiva práctica. La propuesta combina la aplicación de conocimientos adquiridos a lo largo de la carrera con el desafío de responder a necesidades actuales del cliente.

El propósito principal de **Fitit Returns** es diseñar e implementar una herramienta que contribuya a mejorar la experiencia del usuario final en los procesos de devolución y cambio, a la vez que proporciona a las tiendas un mecanismo más eficiente para gestionar estas operaciones y coordinarse con los demás actores del proceso.

De este modo, el proyecto no solo cumple con los objetivos académicos de la carrera, sino que también busca aportar valor al sector de la indumentaria, contribuyendo a experiencias digitales más completas y centradas en el usuario.

1.1. Entorno conceptual

En el sector de la moda, el comercio electrónico se ha consolidado como un canal fundamental en la relación entre usuarios y tiendas. Una página web dejó de ser un simple catálogo digital para convertirse en una extensión de la tienda física: un espacio donde los usuarios exploran colecciones, eligen productos, concretan compras y gestionan envíos de manera ágil y accesible. De este modo, el *e-commerce* no solo amplía el alcance de los comercios, sino que también redefine la experiencia de compra, adaptándola a las demandas de conveniencia y personalización que caracterizan al consumidor actual.

El crecimiento sostenido del *e-commerce* refuerza esta tendencia. Entre 2023 y 2026, se proyecta que el **mercado de comercio electrónico en América Latina crecerá un 22%** [1]. Este dinamismo obliga a las empresas a replantear sus procesos y garantizar que cada instancia de la experiencia digital esté a la altura de las expectativas de los usuarios.

Dentro de este marco, la **postventa** ocupa un lugar central. Procesos como devoluciones y cambios no son solo una necesidad operativa, sino también un factor determinante en la percepción de calidad del servicio. En el sector moda, la tasa promedio de devoluciones en el comercio electrónico está **entre el 16.7% y 30% de las compras online**, siendo esta categoría la que concentra las tasas más altas de devoluciones [2].

Al mismo tiempo, la experiencia de postventa impacta directamente en la confianza y fidelidad del cliente: un **96% de los consumidores afirma que es probable que vuelvan a comprar en una tienda en línea después de una experiencia de**

devolución sencilla [3]. En este sentido, un servicio deficiente puede traducirse en pérdida de confianza, abandono del canal digital y menor fidelización, mientras que un proceso ágil y estandarizado contribuye a consolidar relaciones de largo plazo.

1.2. Descripción del cliente

El proyecto se desarrolla en colaboración con **Fitit**, una *startup* uruguaya dedicada a la implementación de soluciones tecnológicas para el comercio electrónico de indumentaria. Su principal producto consiste en una **herramienta de recomendación de talles**, basada en la aplicación de técnicas de inteligencia artificial, donde a partir de la foto de una prenda proporcionada por el usuario, permite sugerir el talle más adecuado. Dicho producto fue desarrollado con el propósito de aumentar las ventas de las tiendas online, mejorar la experiencia del usuario final y reducir la cantidad de devoluciones asociadas a errores en la elección de talles.

La propuesta actual de la empresa combina el uso del recomendador de talles y visualización de la guía de talles con un modelo de suscripción mensual. En este esquema, cada tienda cuenta con un ejecutivo asignado, responsable de mantener actualizados los productos y garantizar que las medidas correspondientes estén correctamente registradas, asegurando así la efectividad del sistema.

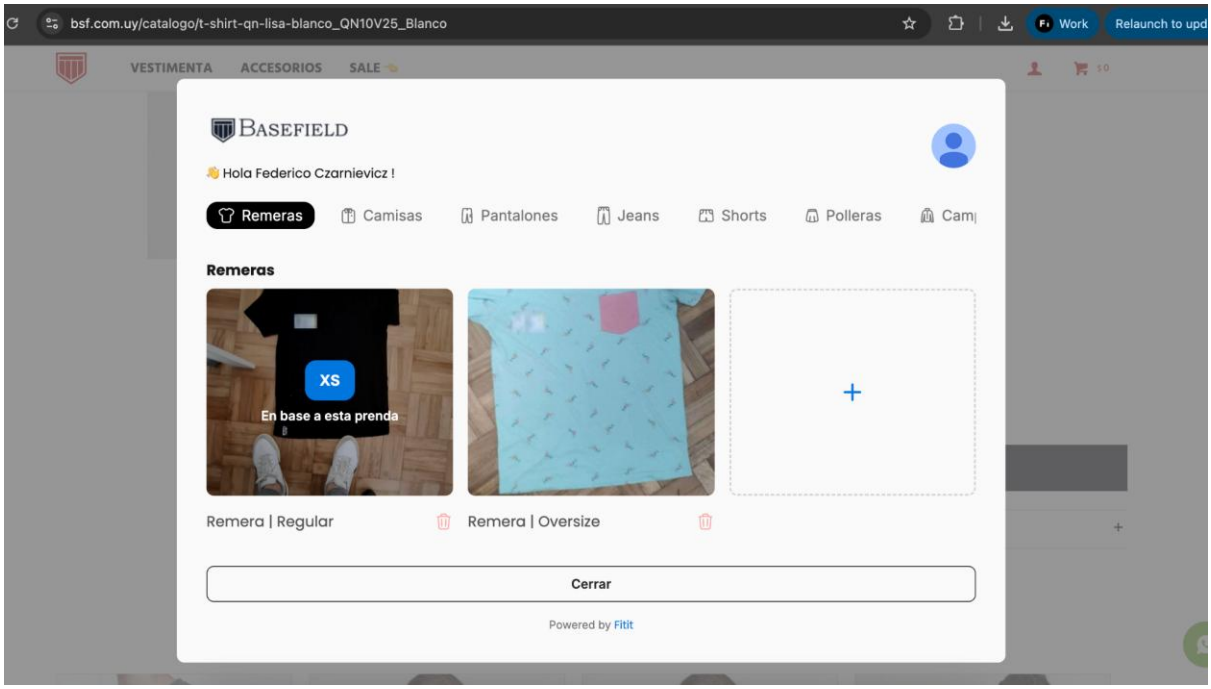


Ilustración 1 - Armario Virtual del recomendador de talles de Fitit

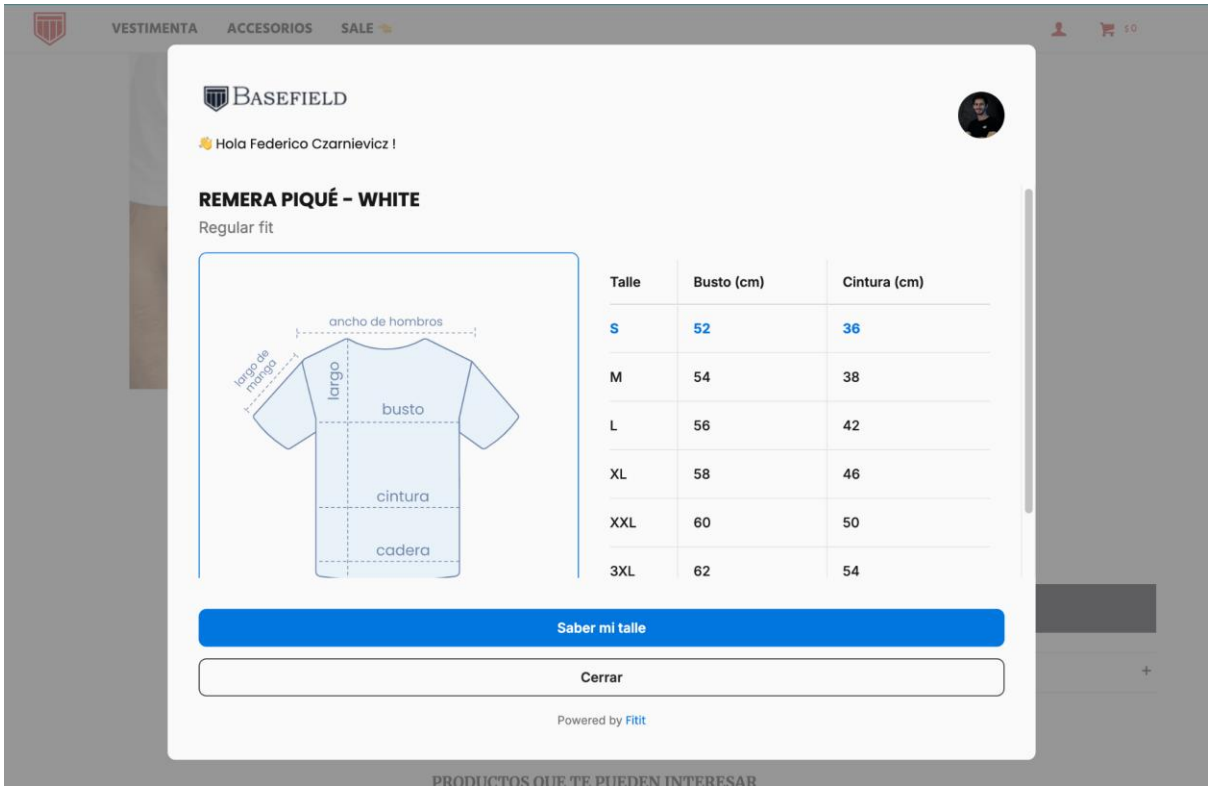


Ilustración 2 - Guía de talles de Fitit

Fitit se encuentra actualmente en una etapa de fuerte crecimiento en el mercado de Latinoamérica, con alrededor de 100 clientes distribuidos en más de ocho países. Sus principales clientes son tiendas de *e-commerce* del sector moda, que buscan optimizar la precisión en la selección de talles y ofrecer una experiencia diferenciada a sus consumidores.

1.3. Objetivos generales

A continuación se presentan los objetivos principales que guiaron el trabajo de Fitit Returns. Se agrupan en tres categorías para mantener el foco, alinear expectativas con el cliente y evaluar el éxito del desarrollo. En el capítulo 10. Conclusiones se analiza el grado de cumplimiento y la evidencia asociada a cada objetivo.

1.3.1. Objetivos académicos

- **Actualizar y fortalecer competencias técnicas.** Incorporar y aplicar tecnologías vigentes del ecosistema web para construir una solución moderna y mantenible.
- **Diseño y gestión de *software*.** Profundizar en prácticas de arquitectura, gestión ágil y optimización de sistemas, integrando lo aprendido en la carrera en un proyecto real con cliente.

1.3.2. Objetivos del producto

- **Gestionar y centralizar devoluciones en *e-commerce*.** Ofrecer a tiendas y clientes un flujo claro, trazable y consistente de devoluciones y cambios.

- **Mejorar la experiencia del usuario final.** Reducir fricción con mensajes claros, menos pasos y estados visibles que aumenten la confianza en la compra online.

- **Generar métricas útiles para el negocio.** Proveer información confiable para la toma de decisiones (motivos de devolución, tiempos de resolución, uso de créditos).

- **Aportar a Fitit un marco de validación.** Vincular las devoluciones con el impacto de su actual herramienta en el mercado a partir de evidencia objetiva.

- **Preparar integraciones externas.** Dejar una base fácilmente extensible para conectar con diferentes plataformas de *e-commerce* y cadeterías.

1.3.3. Objetivos del proyecto

- **Construir una plataforma eficiente y automatizada.** Estandarizar tareas repetitivas, minimizar errores y acortar tiempos del proceso de devolución.

- **Asegurar transparencia y control extremo a extremo.** Registrar decisiones, evidencias y estados para tiendas, usuarios y equipo interno.

- **Trabajar con método y trazabilidad.** Establecer prácticas de planificación, validación y documentación que permitan iterar con seguridad y aprender de cada entrega.

1.4. Integrantes del equipo y motivación para realizar este proyecto

El proyecto fue desarrollado por un equipo de tres estudiantes de la carrera de Ingeniería en Sistemas como tesis final. Todos los integrantes cuentan con experiencia laboral en la industria del *software* desde hace al menos dos años, desempeñándose en distintas áreas: **Federico** trabaja en Fitit desde los inicios de la empresa, **Nicolás** en una *startup* enfocada en ciberseguridad y **Florencia** en una *software factory*.

La idea del proyecto surgió a partir de la experiencia de **Federico** en Fitit, quien, en conversaciones con los fundadores de la empresa, identificó una oportunidad concreta de mejora vinculada a la gestión de devoluciones en *e-commerce.* Esta propuesta fue posteriormente compartida y discutida con **Nicolás** y **Florencia**, lo que permitió consolidar el inicio del proyecto.

La motivación principal radica en aplicar los conocimientos adquiridos a lo largo de la carrera para abordar un problema real del mercado, combinando la perspectiva

académica con la experiencia práctica de cada integrante. Asimismo, el desafío representa un valor agregado al abrir nuevas oportunidades de innovación en el ámbito del comercio electrónico, especialmente en lo relacionado a la eficiencia operativa y la mejora de la experiencia del usuario final.

2. Planteamiento del problema

2.1. Contexto del problema

En el comercio electrónico de indumentaria, los procesos de devolución y cambio representan un punto crítico tanto para la experiencia del usuario como para la gestión operativa de las tiendas. Aunque la compra online se ha vuelto cada vez más ágil y personalizada, la postventa no ha seguido el mismo ritmo de evolución, generando una de las principales fuentes de fricción en el sector.

2.1.1. Desde el punto de vista de las tiendas

La **evolución acelerada del *e-commerce*** exige replicar en el entorno digital servicios que antes estaban asociados únicamente a la tienda física, entre ellos los procesos de devoluciones y cambios. Sin embargo, estos procesos suelen estar poco desarrollados, carecen de estandarización y, en muchos casos, no existen mecanismos eficientes que permitan al usuario gestionar devoluciones de forma ágil y confiable.

El comercio electrónico enfrenta **altas tasas de devoluciones**, que representan un costo operativo significativo, afectan la rentabilidad y generan fricción en la experiencia de compra.

Las tiendas **no realizan un correcto seguimiento de la postventa**, lo que deriva en falta de trazabilidad, ausencia de métricas confiables y dificultades para tomar decisiones estratégicas.

2.1.2. Desde el punto de vista del usuario final

Actualmente, los canales digitales ofrecen **pocas alternativas claras** y **accesibles** para devolver o cambiar productos.

Las tiendas implementan distintos métodos de gestión de devoluciones, que suelen ser multicanal y poco estandarizados:

- **Correo electrónico:** el cliente envía un mail y recibe instrucciones de manera manual.
- **Teléfono o WhatsApp:** la gestión se realiza de forma conversacional, muchas veces sin registro formal en los sistemas de la tienda.
- **Formularios web o PDFs:** requieren varios pasos manuales y suelen ser poco intuitivos.
- **Redes sociales:** algunos clientes inician devoluciones por mensajes de Facebook, Instagram, entre otros.

Además, en la mayoría de los casos, las devoluciones sólo implican devolver el producto, sin ofrecer alternativas como cambio de talle o reembolso inmediato. Las

políticas suelen carecer de claridad, no existe un estándar común y, en muchos casos, las tiendas optan por no ofrecer esta opción.

Para el usuario, estas limitaciones se traducen en **frustración, pérdida de confianza y abandono de la marca**, ya que la posibilidad de devolver es un factor clave en la decisión de compra y recompra [4].

2.1.3. El desafío para Fitit

Fitit busca reducir las devoluciones en el e-commerce de moda a través de su **recomendador de talles**, ya que una gran parte de estas se origina por errores en la elección del tamaño. Sin embargo, actualmente **no existe una vinculación directa entre esta funcionalidad y las métricas de devolución**, lo que impide medir con precisión su impacto real en la reducción de devoluciones.

La ausencia de evidencia sistematizada dificulta demostrar de manera objetiva el verdadero impacto del recomendador de talles, en la reducción de devoluciones y, por lo tanto, el valor que aporta a los usuarios.

Este escenario muestra la necesidad de soluciones que centralicen, estandaricen y optimicen los procesos de postventa, conectando eficazmente a tiendas, proveedores y usuarios dentro del ecosistema de *e-commerce* de moda.

2.2. Actores principales y necesidades

El proceso de postventa involucra a **tres actores** fundamentales cuya interacción resulta clave para el desarrollo de Fitit Returns: los usuarios finales, las tiendas y Fitit. Cada uno cumple un rol clave dentro del ecosistema, y sus necesidades específicas orientan el diseño de la solución propuesta.

Usuarios finales: requieren un proceso de devolución claro, rápido y transparente que genere confianza al momento de realizar compras online.

Tiendas: necesitan visibilidad sobre las causas de devolución, herramientas de gestión y reportes que permitan tomar decisiones basadas en datos.

Fitit: requiere contar con evidencia sólida que permita validar y comunicar el valor de su solución en la reducción de devoluciones.

3. Descripción de la solución

Fitit Returns se presenta como una propuesta integral diseñada para centralizar, estandarizar y optimizar la gestión de devoluciones y cambios en los *e-commerce* de indumentaria. La plataforma aborda las principales problemáticas que enfrentan tanto los usuarios como las tiendas en los procesos de devolución.

Su implementación fortalece significativamente la confianza de los usuarios al proporcionar un proceso de devolución claro y automatizado, mejora la experiencia global de la postventa mediante interfaces intuitivas y multiidioma, y simplifica sustancialmente la operativa tanto de tiendas como de los administradores de la plataforma (Fitit). De esta forma, la solución contribuye al desarrollo del ecosistema digital de moda, alineando estratégicamente los intereses del usuario final, las tiendas participantes y la propia empresa Fitit.

El sistema está conformado por tres aplicaciones web desarrolladas en Next.js, cada una optimizada para su audiencia específica:

- **fitit-backoffice:** Orientada a los empleados de Fitit para la administración centralizada del sistema.
- **fitit-store:** Dedicada a las tiendas para la gestión autónoma de sus políticas, devoluciones, entre otras cosas.
- **fitit-user:** Interfaz principal para los usuarios finales que desean procesar devoluciones de productos adquiridos en las tiendas participantes.

3.1. Descripción funcional del producto

La arquitectura funcional de Fitit Returns se basa en un modelo de separación de responsabilidades que permite a cada tipo de usuario acceder únicamente a las funcionalidades relevantes para su rol, garantizando así la seguridad y eficiencia operativa del sistema.

3.1.1. Vista del cliente

La aplicación **fitit-backoffice** representa el centro de administración y monitoreo de toda la plataforma, diseñada específicamente para los empleados y administradores de Fitit. Esta aplicación proporciona visibilidad total sobre todas las operaciones generadas por usuarios y tiendas, así como la posibilidad de edición, creación y eliminación de diferentes recursos (devoluciones, usuarios, tiendas, etc.).

Dashboard:
El dashboard principal ofrece una visión integral y en tiempo real de las métricas críticas del negocio, organizado en un diseño de dos columnas que optimiza la presentación de la información. El **Resumen de Devoluciones y Cambios** presenta estadísticas calculadas automáticamente a partir de los datos del sistema, incluyendo totales globales, comparativas porcentuales mensuales, desgloses semanales y métricas por estado. La sección de **Distribución de Usuarios** clasifica la base en tres categorías, empleados de Fitit, administradores de tienda y usuarios finales, con

conteos precisos y visualizaciones que permiten comprender la proporción de cada grupo.

El **Rendimiento por Tienda** muestra, mediante gráficos de barras, el volumen de devoluciones y cambios de cada *e-commerce,* mientras que la **Tendencia Semanal** representa la actividad diaria con barras codificadas por colores que facilitan la detección de picos y la planificación de recursos. Por su parte, el **Análisis de Motivos de Devolución** procesa automáticamente las causas reportadas por los usuarios, calcula su porcentaje de incidencia y contribuye a identificar problemas recurrentes en productos o procesos.

Finalmente, la sección de **Actividad Reciente** funciona como un registro en tiempo real de las acciones del sistema, con marcas temporales exactas e iconografía diferenciada según el tipo de evento. Además, el dashboard incorpora la posibilidad de **exportar a CSV**, lo que permite generar reportes completos para su análisis o integración con sistemas externos.

Gestión de Usuarios:
El módulo de usuarios está implementado en dos pestañas que separa la gestión de Usuarios del Sistema (empleados internos y administradores de tienda) de Usuarios con Devoluciones (clientes que han procesado devoluciones). Para usuarios del sistema, proporciona funcionalidades completas de CRUD incluyendo creación de nuevos usuarios con asignación de roles, edición de información personal y eliminación. El sistema presenta la información en tablas organizadas con filtros por rol y fecha de creación. Para usuarios con devoluciones, muestra métricas específicas incluyendo número total de devoluciones realizadas, fecha de la última devolución, y enlaces directos para revisar los detalles del mismo y su historial.

Gestión de Tiendas:
Permite el control completo de las distintas tiendas dentro de Fitit Returns. Incluye funcionalidades para registro de nuevas tiendas con información completa incluyendo nombre, país, dirección física, URL del logo, y configuración de métodos de envío permitidos (recogida en tienda, envío a domicilio, puntos de recogida). El sistema presenta la información en formato de tabla con visualización de logos, banderas de países, y etiquetas por colores para métodos de envío. Proporciona capacidades de edición completa de información de tiendas existentes y eliminación con confirmación para casos excepcionales.

Gestión de Devoluciones:
El módulo de devoluciones es el núcleo operativo de la aplicación, proporcionando visibilidad completa sobre todas las transacciones de devolución del sistema. Presenta métricas de estado mediante tarjetas que muestran conteos de devoluciones pendientes, aprobadas, en camino y rechazadas, funcionando también como filtros clickeables para análisis específicos. La tabla principal muestra información detallada de cada devolución incluyendo ID único, estado, tipo de transacción (devolución, cambio o compra con créditos), precio original, créditos generados, productos involucrados, y fecha de creación. Implementa paginación avanzada con opciones de tamaño de página configurable y filtrado por estado con indicadores visuales de filtros activos. Proporciona vista detallada mediante modales que muestran información completa. Incluye capacidades de edición de estado y

creación manual de devoluciones para casos especiales. La funcionalidad de **exportación a CSV** permite generar reportes filtrados para análisis externo y presentaciones ejecutivas.

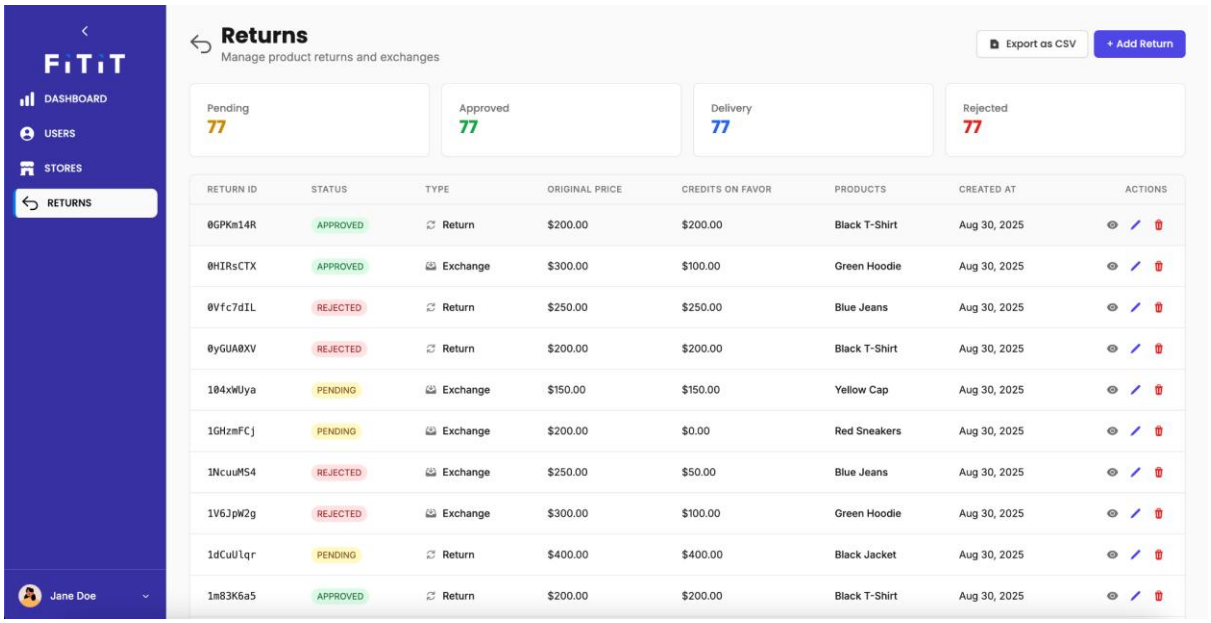


Ilustración 3 - Gestión de devoluciones en fitit-backoffice

Las capturas correspondientes a las pantallas se incluyen en el Anexo 12.1. Pantallas de *backoffice*.

3.1.2. Vista de la tienda

La aplicación **fitit-store** está específicamente diseñada para propietarios y administradores de tiendas, proporcionándoles autonomía completa en la gestión de sus procesos de devolución.

Dashboard:
Constituye el centro de control principal para los administradores de tienda, proporcionando una vista integral y en tiempo real de todas las métricas relacionadas con devoluciones, cambios y el impacto financiero de estos procesos. La interfaz se organiza en un diseño de dos columnas que maximiza la visualización de información crítica mientras mantiene una experiencia de usuario intuitiva y profesional.

El componente principal presenta un **Resumen de Devoluciones y Cambios** que muestra las métricas fundamentales del negocio, incluyendo el total de devoluciones y cambios procesados con comparativas porcentuales respecto al mes anterior.

La **Tendencia Semanal** se visualiza mediante un gráfico de barras interactivo que presenta la distribución diaria de devoluciones y cambios durante los últimos siete días, utilizando barras de progreso por colores (azul para devoluciones, verde para cambios) que facilitan la identificación rápida de picos de actividad y permiten a los administradores planificar recursos y personal según los patrones históricos observados.

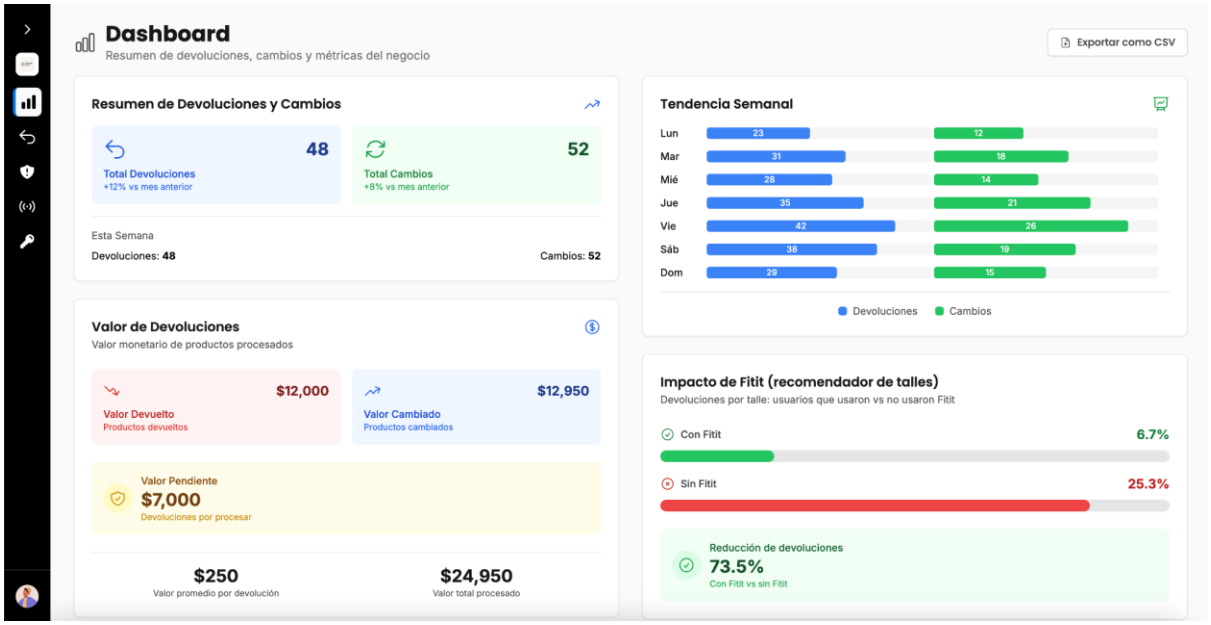


Ilustración 4 - Panel de métricas en fitit-store

El **Análisis de Razones de Devolución** procesa automáticamente los motivos reportados por los usuarios, presentándolos con sus respectivos porcentajes de incidencia, lo que permite identificar problemas recurrentes en productos específicos o procesos de la tienda.

La sección de **Impacto Financiero** calcula y presenta métricas monetarias incluyendo el valor total de productos devueltos, el valor de productos cambiados, el valor promedio por devolución y el valor total de devoluciones pendientes de procesamiento.

El **Gráfico de Impacto de Fitit** presenta una comparativa visual del porcentaje de devoluciones por cambio de talle entre usuarios que utilizaron el recomendador de talles contra aquellos que no lo usaron durante su proceso de compra. La visualización incluye un indicador de reducción de devoluciones que cuantifica la mejora porcentual lograda cuando los usuarios emplean el recomendador de talles de Fitit.

La sección de **Actividad Reciente** mantiene un detalle en tiempo real de las últimas acciones realizadas en el contexto de la tienda específica, incluyendo devoluciones aprobadas, rechazadas, enviadas y nuevos usuarios registrados, que permiten monitorear la actividad del sistema y responder rápidamente a situaciones que requieran atención inmediata.

Finalmente, el dashboard incluye funcionalidad de **exportación a CSV** que permite generar reportes detallados con todas las métricas principales del período.

Gestión de Devoluciones:
El módulo de devoluciones proporciona control completo sobre todas las solicitudes de devolución dirigidas a la tienda específica. Es análogo a lo ya explicado en la sección 3.1.1. Vista del cliente, salvo por dos diferencias significativas:

- Las devoluciones visualizadas son solamente las de la tienda en cuestión, a diferencia del *backoffice* donde se muestran todas las devoluciones del sistema.
- En vez de exponer una vista de detalle, edición y eliminación, hay dos acciones claras aprobar y rechazar, que luego abren un modal de confirmación para que el administrador de tienda pueda revisar en detalle antes de seguir con el flujo.

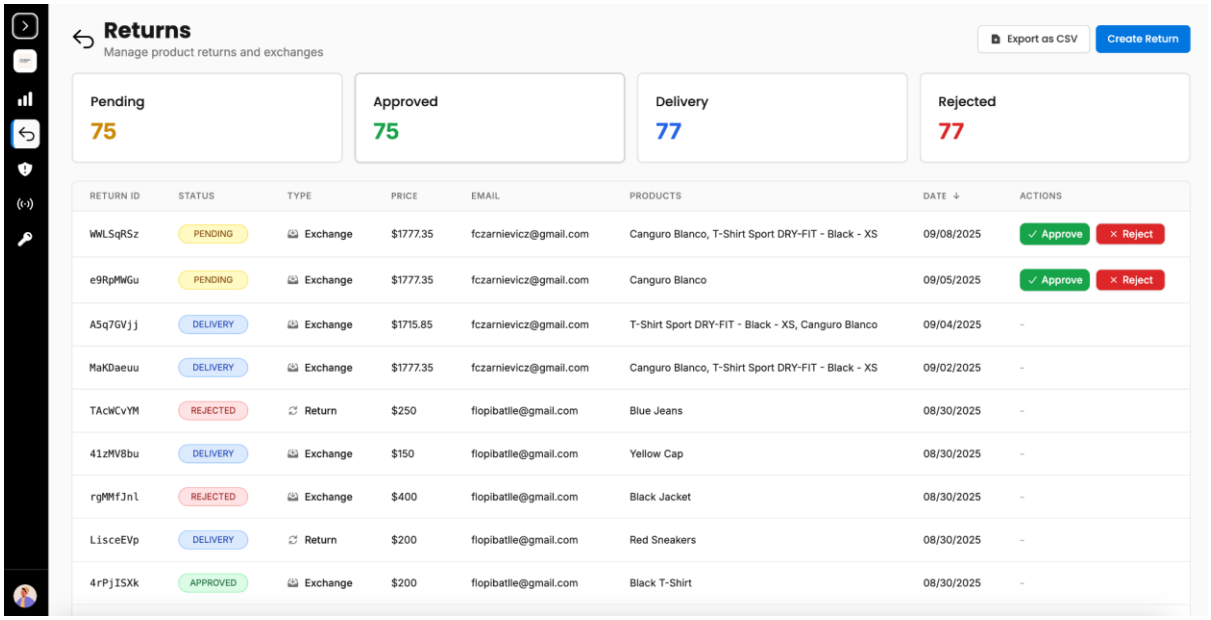


Ilustración 5 - Gestión de devoluciones en fitit-store

Procesamiento de Devolución en Tienda Física:
Incluye una funcionalidad especializada para el procesamiento de devoluciones presenciales que permite a los empleados de tienda gestionar devoluciones directamente en el punto de venta físico. A diferencia de las devoluciones online que requieren revisión y aprobación posterior, las devoluciones procesadas en tienda física pasan automáticamente al estado aceptado, ya que los empleados pueden inspeccionar físicamente los productos antes del procesamiento, eliminando completamente la necesidad de envío y verificación posterior.

El flujo operativo permite que los empleados busquen órdenes específicas utilizando el número de orden y email del cliente para localizar compras elegibles para devolución, tras lo cual pueden seleccionar qué productos específicos de una orden serán devueltos o intercambiados mediante una interfaz intuitiva. Esta modalidad elimina completamente la necesidad de configurar opciones de envío ya que el producto está físicamente presente, permitiendo que el cliente reciba confirmación y procesamiento de créditos en el momento mismo de la transacción. El formulario se puede ver en el Anexo 12.25. Formulario inicial de devolución manual en fitit-store.

Configuración de Políticas de Devolución:
El módulo de políticas permite a cada tienda definir y gestionar sus propias reglas comerciales de devolución. Implementa un sistema de pestañas que separa políticas activas de políticas archivadas. Proporciona interfaces intuitivas para crear políticas específicas por categoría de producto, incluyendo períodos de devolución personalizables, condiciones de aceptación específicas, tipos de reembolso permitidos (crédito en tienda vs. reembolso original), y restricciones especiales para productos personalizados, en liquidación, o de temporada. Las políticas incluyen configuración de métodos de envío permitidos (recogida en tienda, envío a domicilio, puntos de recogida) y reglas de elegibilidad basadas en tiempo transcurrido desde la compra, estado del producto, y otras condiciones comerciales.

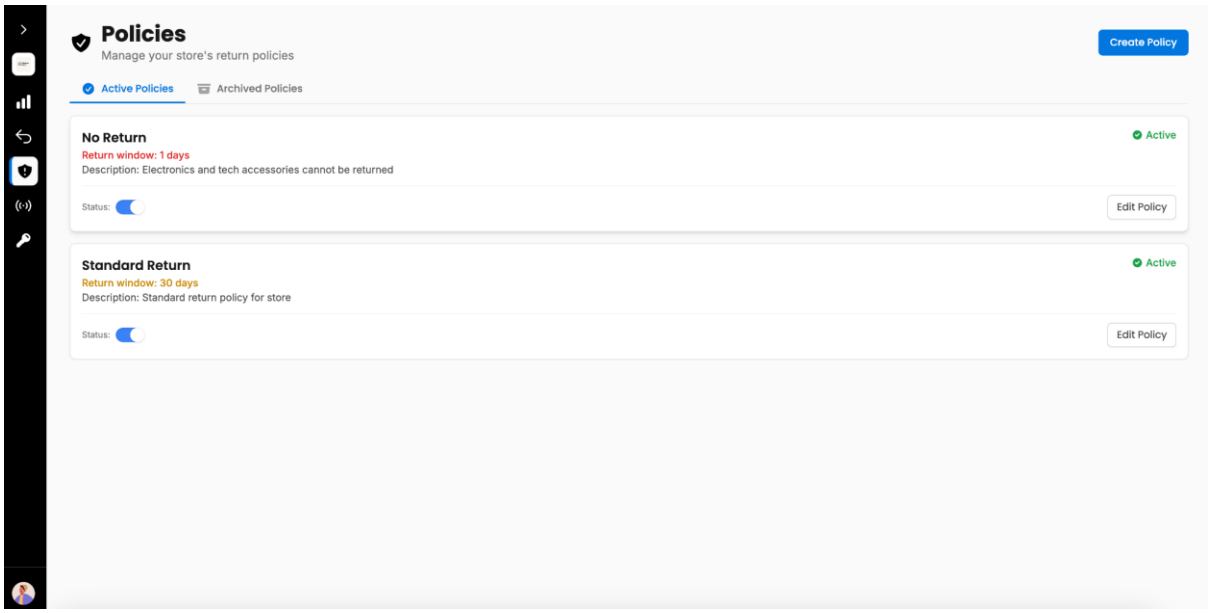


Ilustración 6 - Gestión de políticas de tienda en fitit-store

Gestión de *Webhooks*:
La **gestión de *webhooks*** [5] [6] en Fitit Returns fue incorporada para dar respuesta a la necesidad de las tiendas de poder ejecutar acciones cuando ocurren determinados eventos específicos. A través de esta funcionalidad, cada vez que ocurre un evento relevante en el sistema, por ejemplo, la creación de una solicitud de devolución o la aprobación de un cambio, la información puede transmitirse de inmediato a otros sistemas de la tienda.

Esto permite una **integración en tiempo real** con plataformas de *e-commerce*, sistemas de inventario, sistemas de planificación de recursos empresariales (*ERP*) o herramientas de atención al cliente, evitando procesos manuales y reduciendo el riesgo de errores. En la práctica, una devolución registrada en Fitit Returns puede reflejarse automáticamente en el stock disponible, disparar un flujo en el sistema de gestión de relaciones con los clientes (*CRM*) del equipo de soporte o incluso actualizar métricas internas de logística.

Se debe indicar la URL del endpoint que recibirá los mensajes, agregar una descripción para facilitar la identificación en escenarios con múltiples integraciones y seleccionar los eventos del sistema que disparará el *webhook*. De este modo, las

tiendas cuentan con un control granular que les permite decidir qué información necesitan recibir y a qué sistema externo enviarla, reforzando así la flexibilidad y adaptabilidad de la solución dentro de su ecosistema tecnológico.

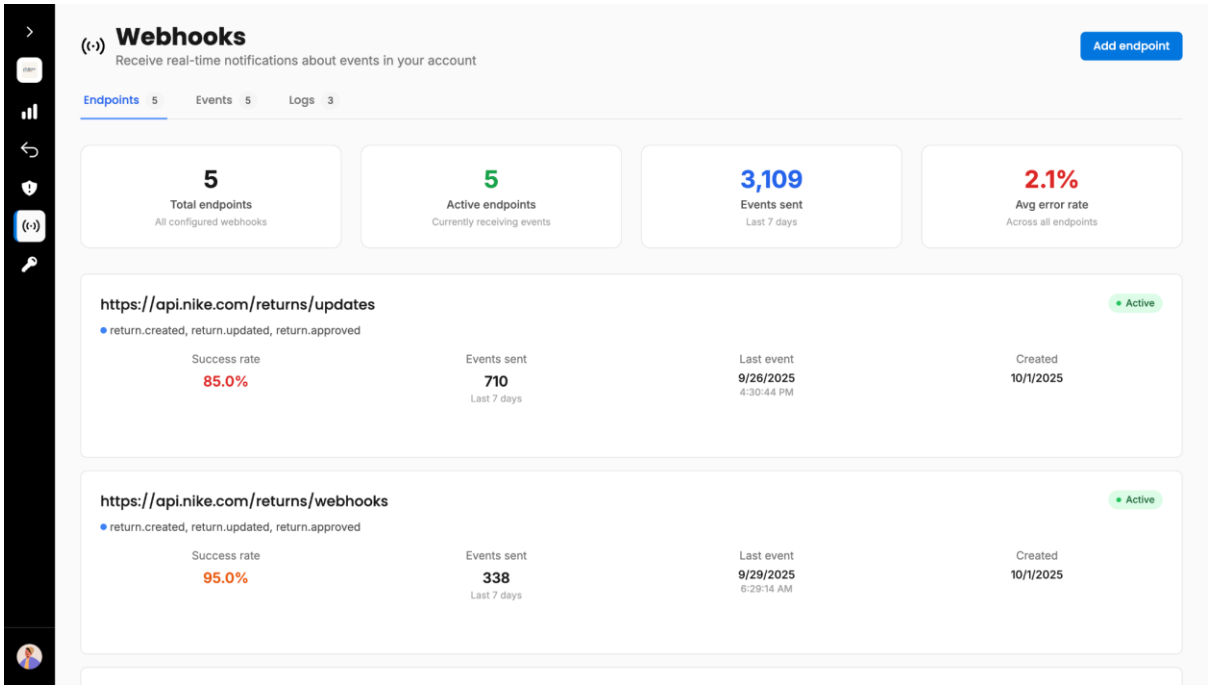


Ilustración 7 - Gestión de *webhooks* en fitit-store

La decisión de incluir esta capacidad responde a la importancia de ofrecer una **solución flexible** que se adapte al ecosistema tecnológico de cada tienda, sin obligarlas a cambiar la forma en que ya trabajan.

Administración de *API Keys*: La administración de ***API Keys*** se diseñó con el objetivo de ofrecer a las tiendas un mecanismo seguro y escalable para conectar Fitit Returns con sus propios sistemas. La rotación periódica de claves y el control centralizado aportan un nivel extra de seguridad, reduciendo riesgos frente a accesos no autorizados [7] [8].

El administrador puede asignar un nombre para identificar la clave, definir un período de expiración (siendo recomendable optar por duraciones cortas) y, de manera opcional, restringir el acceso mediante **listas blancas de direcciones IP**. Esto último garantiza que sólo solicitudes provenientes de ubicaciones autorizadas puedan interactuar con la *API*, reduciendo el riesgo de accesos indebidos.

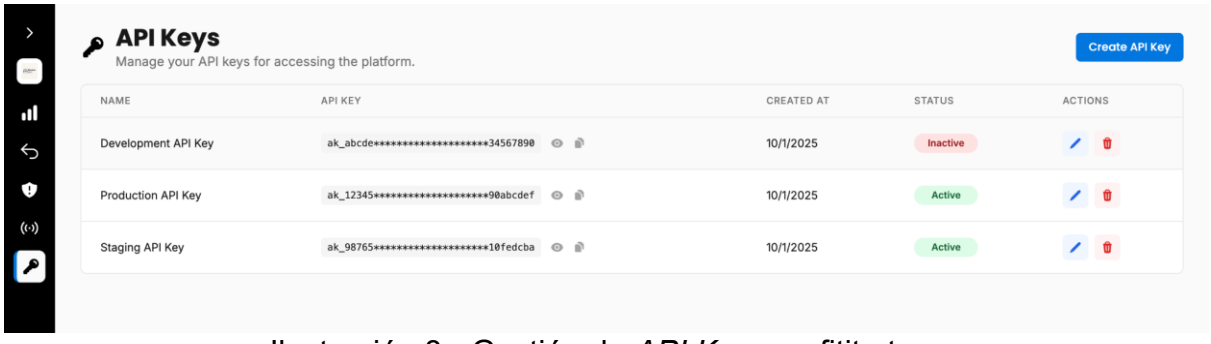


Ilustración 8 - Gestión de *API Keys* en fitit-store

Tener una *API* accesible por los usuarios convierte a Fitit Returns en una plataforma abierta a integraciones avanzadas, que no depende exclusivamente de sus interfaces gráficas, sino que permite a los equipos técnicos de las tiendas extender y personalizar los procesos de devolución de acuerdo con sus propias reglas de negocio.

3.1.3. Vista del usuario final

La aplicación **fitit-user** representa la interfaz principal orientada al consumidor final, diseñada con un enfoque centrado en la experiencia del usuario y la simplicidad del proceso de devolución. Es una aplicación web *responsive* diseñada para usar tanto en el celular como en la computadora. Para poder ver con mayor claridad las pantallas y funcionalidades se muestran las imágenes en *desktop*, pero en el Anexo 12.2. Pantallas en *mobile* quedan también las análogas para *mobile*. Desde la perspectiva pública, esta aplicación es la que se presenta bajo el nombre Fitit Returns, es decir, el producto visible para los usuarios finales.

El flujo de devolución puede iniciarse de dos maneras. En primer lugar, desde el *e-commerce* que tiene integrado Fitit Returns, mediante un botón destacado con un texto como “Solicitar devolución”. Este botón actúa como un *Call to Action* (llamado a la acción) que dirige al usuario al proceso de devoluciones. Al hacer click, se abre en una nueva pestaña la página de Fitit Returns, donde el usuario continúa el procedimiento de forma guiada y centralizada. Alternativamente, también se puede acceder directamente a Fitit Returns e iniciar el proceso desde el mismo formulario inicial, sin necesidad de pasar previamente por la tienda. Esta flexibilidad elimina barreras de acceso y se adapta a diferentes comportamientos del consumidor.

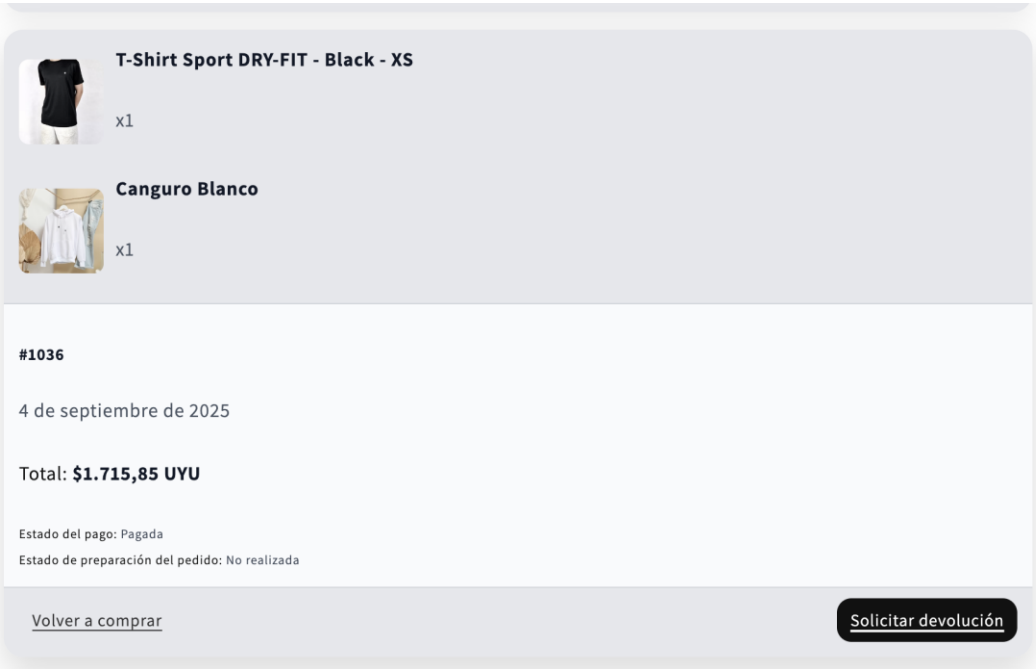


Ilustración 9 - Botón de “solicitar devolución” en un *e-commerce* de Shopify

Proceso de Devolución:
El formulario de inicio presenta una interfaz limpia donde el usuario proporciona tres datos esenciales: la tienda de origen, el número de orden y el email asociado a la

compra. Cabe destacar que si el usuario viene desde el *Call To Action* ubicado en el *e-commerce*, entonces la información ya queda cargada por defecto. La validación en tiempo real verifica la existencia de la orden y muestra una vista previa de los productos elegibles, estableciendo desde el primer momento la transparencia que caracteriza todo el proceso.

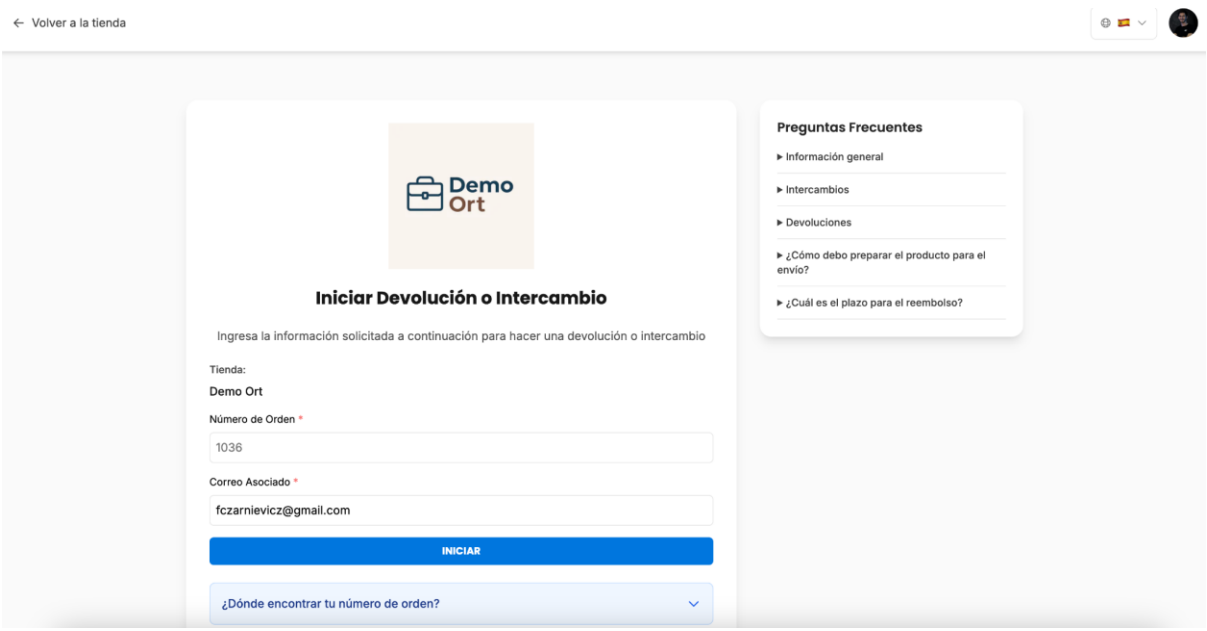


Ilustración 10 - Formulario inicial de devolución

Tras la validación, el sistema despliega una vista previa con todos los productos asociados a la orden. Cada producto se muestra con imagen, descripción y estado de elegibilidad para devolución. Aquellos artículos que no cumplen con las políticas de la tienda aparecen bloqueados con un mensaje explicativo que detalla el motivo (por ejemplo, artículos de liquidación o fuera de plazo), lo que aporta claridad y transparencia al usuario.

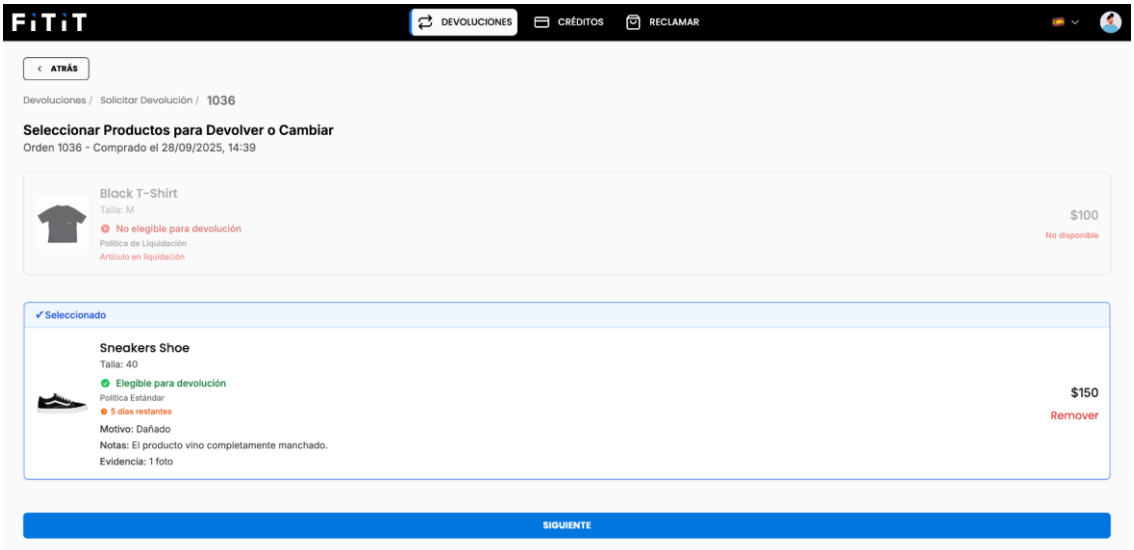


Ilustración 11 - Listado de productos elegibles y no elegibles para devolución

El cliente puede seleccionar los productos que desea devolver y, para cada uno, completar un breve formulario donde especifica el motivo de la devolución mediante opciones predefinidas (talle incorrecto, dañado, no cumple expectativas, etc.), además de un campo abierto para notas adicionales. En esta misma pantalla, se exige la carga de fotografías como evidencia, lo que permite al usuario documentar el estado del producto y agilizar el proceso de revisión por parte de la tienda.

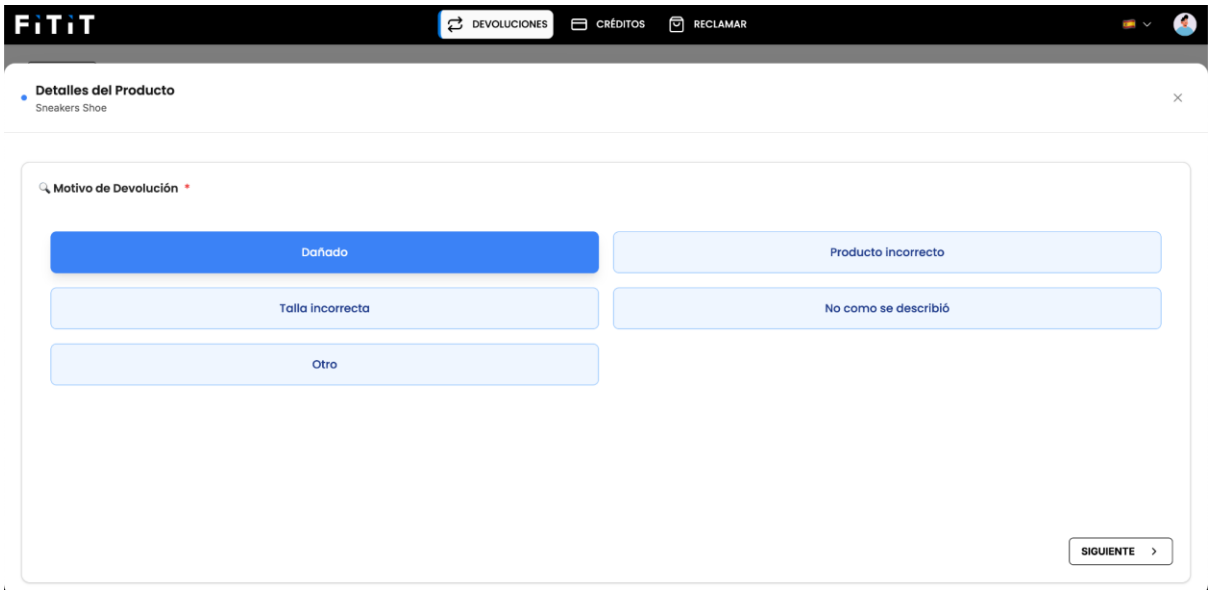


Ilustración 12 - Formulario por producto en la sección de motivo de devolución

Una vez completado el formulario con el motivo de devolución y, en caso de corresponder, cargadas las fotografías como evidencia, el producto queda marcado como **seleccionado para devolución** dentro de la orden. Al continuar, se presenta un modal de confirmación donde el usuario debe indicar si desea un **reembolso en créditos** para utilizar en futuras compras o un **cambio por otro producto**. Esta decisión determina el flujo posterior del proceso: en el caso de créditos, se acreditan automáticamente en la cuenta del usuario al aprobarse la devolución; en el caso de cambios, el flujo continúa.

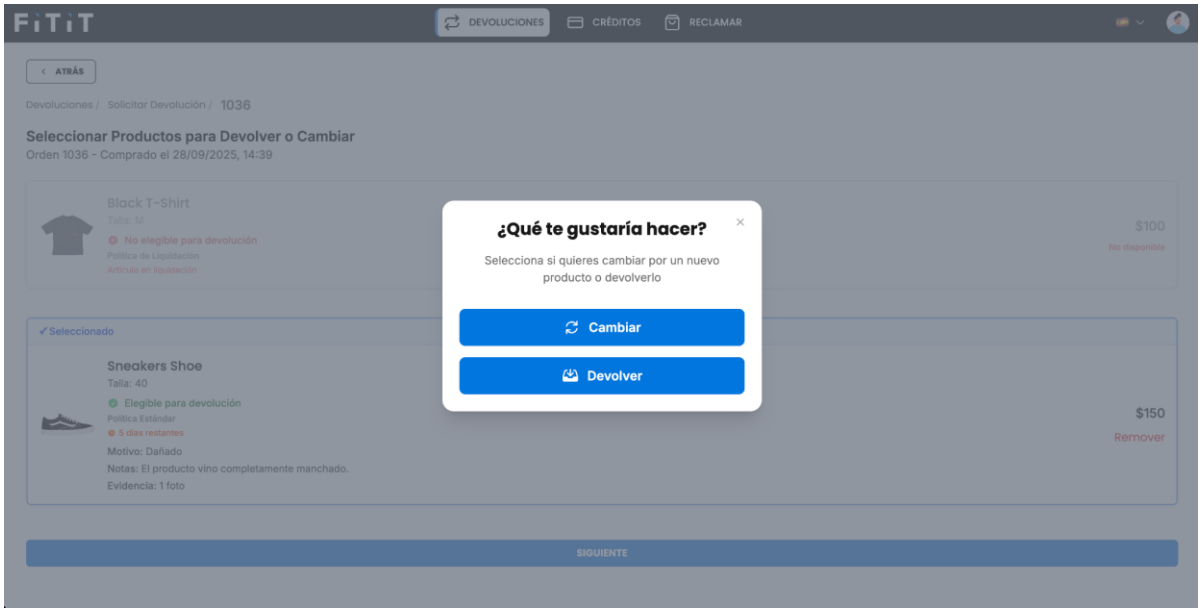


Ilustración 13 - Modal de decisión entre devolución o cambio de producto

Si el usuario elige la opción **cambio** en el modal de decisión, el sistema lo dirige a un selector de productos de la misma tienda para completar el intercambio. En esta vista se presenta un catálogo navegable con buscador y filtros por categoría, talle y rango de precio. El objetivo es que el cliente encuentre, sin salir de Fitit Returns, el artículo sustituto que desea recibir. La interfaz indica en todo momento cuántos ítems de la orden original están siendo cambiados y mantiene visible el progreso del flujo.

Previo a la confirmación, se define la **logística del cambio**. El usuario elige el método según las opciones habilitadas por la tienda: envío del nuevo producto a domicilio, retiro en punto de entrega, o **cambio presencial en tienda** cuando esté disponible. La interfaz deja claras las condiciones (por ejemplo, si el envío del nuevo producto se despacha una vez recibido y verificado el artículo devuelto, o si la tienda ofrece envío cruzado).

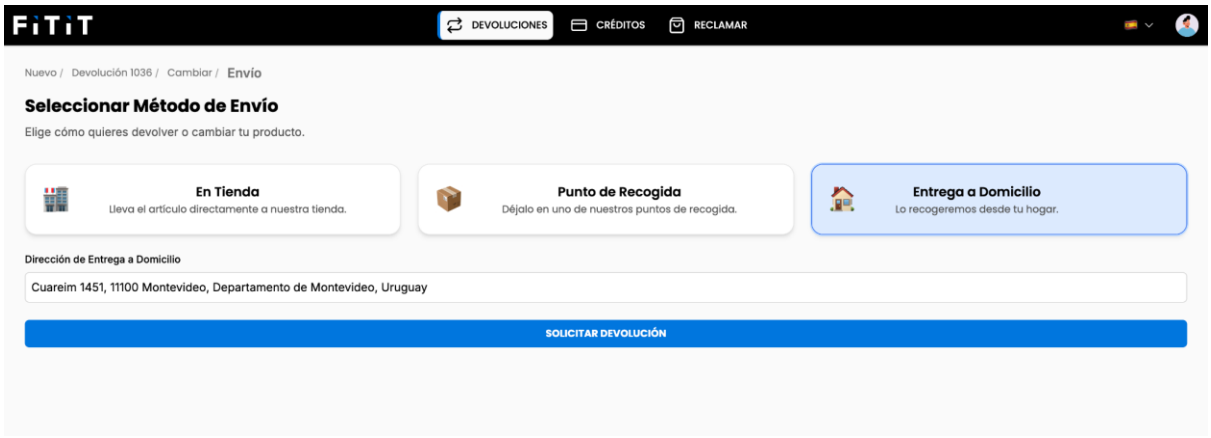


Ilustración 14 - Pantalla de metodo de envio

El proceso concluye con una pantalla de confirmación donde se muestra un resumen de toda la transacción de manera clara y trazable. También tiene las indicaciones a seguir con el producto a devolver.

Seguimiento de Devoluciones:

Proporciona visibilidad completa sobre el estado de cada solicitud mediante una línea de tiempo visual que muestra las etapas del proceso: solicitud enviada, aprobación/rechazo, envío de productos (cuando aplica) y procesamiento final. Cada etapa incluye *timestamps* precisos y descripciones claras del estado actual, eliminando la incertidumbre que tradicionalmente caracteriza los procesos de devolución.

La interfaz de detalles presenta tanto los productos devueltos como los nuevos productos seleccionados (en caso de cambios), incluyendo fotografías, precios y razones de devolución. Esta información completa permite a los usuarios revisar y confirmar todos los aspectos de su transacción, reforzando la confianza en el proceso.

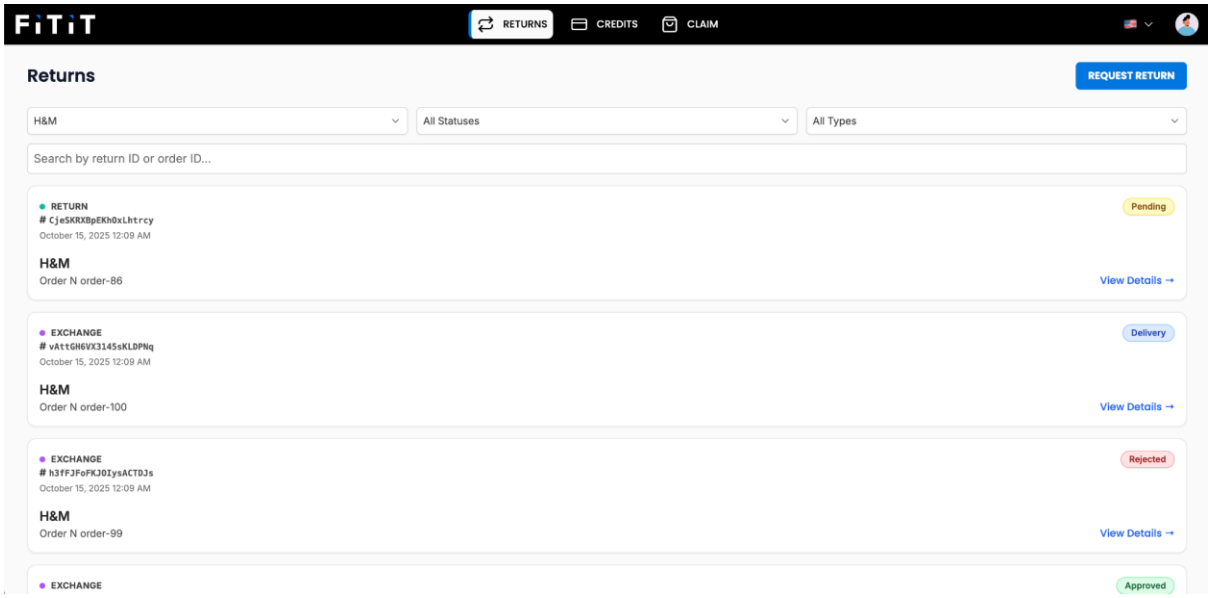


Ilustración 15 - Pantalla de historial de devoluciones

Gestión de Créditos:
El sistema ofrece a los usuarios una billetera digital donde se centralizan todos los créditos disponibles, organizados por tienda. Desde esta sección, el usuario puede consultar su saldo actual, revisar el historial completo de transacciones y utilizar los créditos acumulados en futuras compras. Cada operación queda registrada, garantizando trazabilidad total y confianza en el uso de los saldos. Los créditos se presentan como ***vouchers*** **digitales** asociados a cada tienda, mostrando el monto disponible de manera clara y lista para ser usado de forma inmediata.

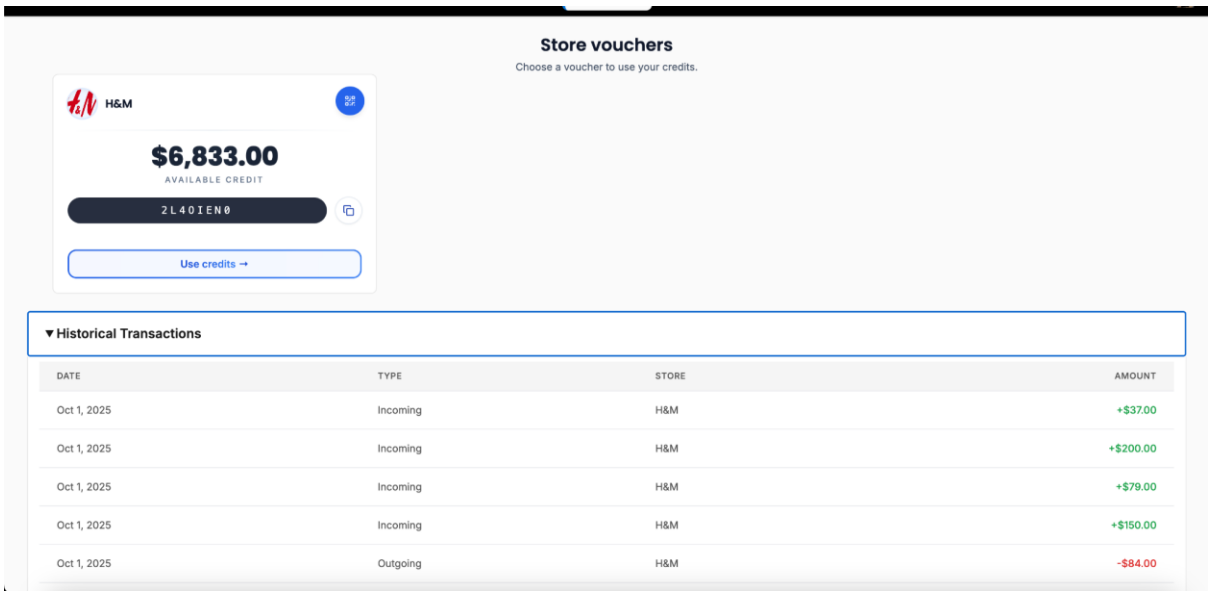


Ilustración 16 - Pantalla de créditos disponibles y transacciones

Integración con *E-commerce*:
La integración con el sistema de *e-commerce* se encuentra implementada con Shopify, permitiendo que el flujo de devoluciones se inicie directamente desde el entorno de compra del usuario (ver Ilustración 9) y a su vez se consume automáticamente la información de las órdenes y productos asociados.

Integración con Cadeterías:
Si bien actualmente no se cuenta con una integración directa con servicios de cadetería, el sistema fue diseñado para soportar integraciones futuras con operadores logísticos externos. Ver más detalles en la sección 7.5.1. *Patrón Strategy*.

Gestión de Perfil Personal:
Permite a los usuarios mantener actualizada su información personal, direcciones de envío, preferencias de comunicación y historial completo de devoluciones realizadas. Incluye configuraciones de privacidad y opciones de notificación personalizables.

Soporte Multiidioma:
Implementa internacionalización completa, utilizando la herramienta i18n [9], con soporte inicial para inglés y español, permitiendo a usuarios de diferentes regiones utilizar la plataforma en su idioma nativo. El sistema detecta automáticamente el idioma preferido del navegador y permite cambios manuales.

Autenticación con Google:
Soporta autenticación mediante Google, reduciendo fricciones en el proceso de registro y acceso a la plataforma.

4. Ciclo de vida

En este capítulo se detallan las principales características del proyecto, las cuales influyeron directamente en la elección de metodologías y en la definición de la estrategia de trabajo más adecuada. Cabe destacar que las metodologías y fases mencionadas en este capítulo se profundizan en detalle en el capítulo 5. Gestión de proyecto.

4.1. Características del proyecto

Proyecto de alta complejidad
El desarrollo de Fitit Returns se concibió como un proyecto de alta complejidad, debido a que abordaba un problema poco estandarizado: los procesos de devoluciones de tiendas. Cada tienda maneja políticas distintas y los requerimientos de usuarios finales son variables, lo que obligó a construir un sistema flexible y validarlo continuamente.

Contexto del equipo
El equipo está conformado por tres integrantes, que debieron compatibilizar el proyecto con responsabilidades académicas y personales. Este entorno particular con limitaciones de tiempo, recursos reducidos y requerimientos cambiantes reforzó la decisión de utilizar un enfoque ágil, capaz de adaptarse rápidamente a contingencias sin perder de vista los objetivos generales.

Requerimientos cambiantes
Desde la fase inicial, los requerimientos no fueron estáticos. A medida que avanzaron las validaciones con *stakeholders*, surgieron nuevas funcionalidades y otras se modificaron o descartaron. Un ejemplo claro fue la incorporación del sistema de créditos, que implicó una re-planificación completa de los *releases*. Este escenario evidenció la necesidad de contar con un marco metodológico que absorbiera el cambio como parte natural del ciclo de vida.

Producto internacional
Otro aspecto distintivo del proyecto fue su alcance internacional, dado que Fitit es una empresa con presencia en varios países, como Uruguay, Argentina, Chile y otros. Esto implicó diseñar una solución adaptable a distintos lenguajes y distintos contextos de *e-commerce*, contemplando diferencias en políticas de devolución, prácticas comerciales y requisitos normativos de cada país. La naturaleza internacional de la empresa elevó la exigencia del sistema, que debía ser lo suficientemente flexible y escalable para funcionar en múltiples mercados sin perder consistencia en su propuesta de valor.

En conjunto, la combinación de complejidad, contexto, variabilidad en los requerimientos y proyección internacional determinó que la adopción de metodologías ágiles fuera la elección más adecuada para sostener el proyecto. Esta elección se alinea con los valores y principios establecidos en el Manifiesto Ágil [10], que promueven la entrega temprana y continua de *software* con valor, la capacidad de respuesta ante cambios y la colaboración estrecha con los *stakeholders*, todos elementos presentes en el desarrollo de Fitit Returns.

4.2. Roles definidos en el equipo

El equipo de Fitit Returns está integrado por tres personas, que compartieron la responsabilidad de ser **desarrolladores y testers** del sistema. Más allá de esta función común, cada integrante asumió además un rol específico de gestión definido en función de sus habilidades y experiencia personal. Esta combinación permitió mantener una distribución clara de responsabilidades sin perder la colaboración en las tareas de construcción y prueba del producto.

Florencia – Gestión general del proyecto
Fue responsable de coordinar el trabajo y asegurar que la dinámica de trabajo ágil se aplique de forma ordenada. Se encargó de facilitar la comunicación fluida entre los integrantes, colaborar en la resolución de impedimentos y mantener la visión global del proyecto, alineando las decisiones diarias con los plazos y objetivos definidos. Este rol le fue asignado por su experiencia previa en coordinación de equipos y tiempos, lo que le permitió sostener la organización y la disciplina del grupo.

Nicolás – Arquitecto de *software* y responsable de calidad
Se ocupó de definir la arquitectura técnica del sistema, tomando decisiones sobre tecnologías, librerías y diseño. También estableció estándares de calidad y ponía especial atención en las revisiones de código, asegurando consistencia y buenas prácticas. Su rol fue definido en base a su experiencia en desarrollo y diseño de *software*, lo que aportó bases técnicas sólidas, escalabilidad y menor riesgo de errores en la integración de los distintos módulos.

Federico – Ingeniero de requerimientos
El encargado de analizar y traducir las demandas en historias de usuario, además de ser el principal responsable de la priorización del *backlog* y representar la visión del cliente dentro del equipo. El hecho de trabajar en Fitit le otorga un conocimiento directo del dominio y de las necesidades reales de negocio. Su rol se asignó también por su experiencia en análisis de requerimientos, además de su vinculación con usuarios finales y tiendas. Mantuvo una comunicación constante con Fitit, lo que permitió obtener retroalimentación continua y ajustar rápidamente el rumbo del proyecto según los requisitos detectados.

Esta organización favoreció la autonomía, la corresponsabilidad y la alineación continua con los objetivos del proyecto.

4.3. Fases del proyecto

El ciclo de vida de Fitit Returns se diseñó siguiendo un enfoque **incremental-iterativo** [11], particularmente adecuado para proyectos con alta complejidad técnica y requerimientos cambiantes. Este enfoque parte de la premisa de que el conocimiento no se obtiene únicamente en la fase de análisis inicial, sino que se construye progresivamente durante el desarrollo, validando y ajustando funcionalidades en cada paso. Este modelo combina dos principios complementarios:

- El **enfoque incremental**, que divide el sistema en partes entregables y funcionales, cada una de las cuales amplía y consolida la versión anterior.

- El **enfoque iterativo**, que permite revisar y mejorar continuamente lo desarrollado, incorporando aprendizajes y retroalimentación obtenida durante el proceso.

Frente a modelos tradicionales como el de cascada [12], que proponen una secuencia rígida y lineal de etapas (análisis, diseño, desarrollo, pruebas y despliegue), el proyecto demandaba aplicar un ciclo de vida incremental-iterativo, permitiendo:

- Dividir los objetivos en entregas parciales, cada una con valor funcional concreto.
- Responder rápidamente a cambios de requerimientos.
- Aprender de cada iteración y usar ese conocimiento para mejorar la siguiente.
- Disminuir la incertidumbre de manera progresiva, ajustando el sistema en función de la experiencia.
- Generar confianza en los *stakeholders*, al mostrar avances tangibles en lapsos cortos y obtener retroalimentación temprana.

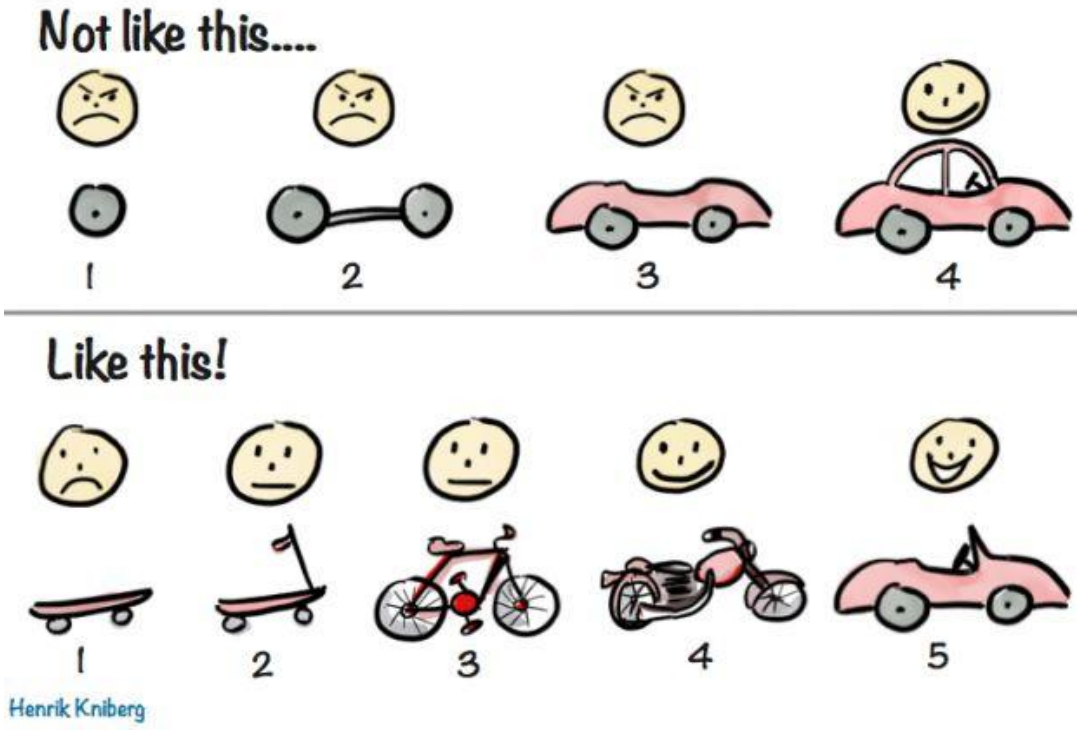


Ilustración 17 - Ciclo de vida incremental-iterativo [13]

En este marco, se eligió una combinación de metodologías ágiles: Scrum [14] y Kanban [15]. La elección se basó en las características de cada fase del proyecto: Scrum aporta estructura y ritmo de entregas en entornos donde se busca construir un producto de manera incremental, mientras que Kanban permite gestionar actividades con alto nivel de incertidumbre, donde el trabajo fluye de forma continua y emergente.

El ciclo de vida incremental-iterativo se aplicó a lo largo de todo el proyecto, abarcando las tres fases principales: **descubrimiento, desarrollo y documentación**.

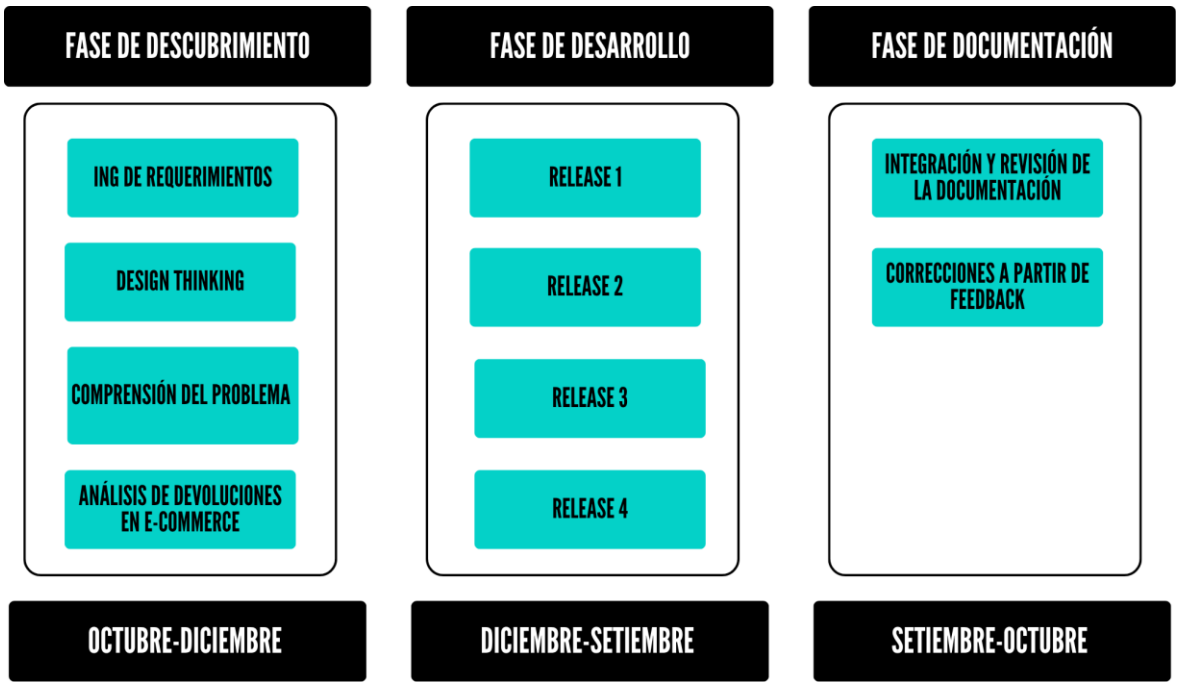


Ilustración 18 - Fases del proyecto

4.3.1. Fase de Descubrimiento

La fase de descubrimiento marcó el punto de partida del ciclo de vida. En ella el objetivo fue entender en profundidad el problema a resolver, identificando los procesos, actores y necesidades vinculadas a las devoluciones en el comercio electrónico. El trabajo inicial se concentró en recopilar información, analizar los flujos actuales y determinar las principales limitaciones de las tiendas y usuarios en torno a los procesos de cambio y devolución.

Durante este período, el equipo transitó por sucesivos ciclos de análisis, validación y reformulación, en los que las ideas iniciales se fueron refinando hasta transformarse en requerimientos concretos y priorizados. Estos ciclos de exploración permitieron identificar patrones comunes entre diferentes tiendas, detectar necesidades recurrentes de los usuarios y definir los criterios que debían guiar el diseño del sistema.

El carácter iterativo de la fase fue clave: cada avance generó nueva evidencia que confirmó o desafió decisiones previas. Integrando de forma incremental, consolidando progresivamente para tener una base suficientemente sólida como para iniciar el desarrollo sin perder flexibilidad ante descubrimientos posteriores. Se evitó suposiciones infundadas y promovió una comprensión progresiva y realista del problema. Resultando en un conjunto validado de requerimientos funcionales y no funcionales, una arquitectura conceptual inicial y una visión compartida del alcance.

4.3.2. Fase de Desarrollo

La segunda fase representó la consolidación del ciclo de vida **iterativo–incremental**, en la que el sistema evolucionó de forma controlada a través de múltiples versiones. Cada ciclo de trabajo dio lugar a un incremento funcional del producto, que integraba mejoras respecto al anterior y era sometido a revisión y validación. Esta forma de avanzar permitió obtener una retroalimentación constante, detectar tempranamente los errores y alinear el rumbo del desarrollo con las necesidades del negocio.

Esta fase no fue solo programar; implicó un proceso de construcción, aprendizaje y reajuste continuo. Cada incremento aportó nueva información sobre el comportamiento del sistema, sobre la experiencia de uso y sobre la viabilidad técnica de las decisiones tomadas. Esto generó un efecto acumulativo: el conocimiento adquirido en una iteración se aplicaba inmediatamente en la siguiente, mejorando la calidad del código, la arquitectura y la lógica de negocio.

El carácter incremental se reflejó en la manera en que el sistema fue creciendo: desde una versión mínima capaz de ejecutar el flujo básico de devoluciones, hasta un producto completo, estable y preparado para su despliegue. La evolución del sistema no solo implicó la adición de funcionalidades, sino también la consolidación de los mecanismos de validación, el fortalecimiento de los procesos internos y la madurez del propio equipo.

4.3.3. Fase de Documentación

La última fase del ciclo de vida tuvo como propósito integrar, formalizar y consolidar los resultados del proyecto, tanto en su dimensión técnica como académica. Más que una instancia de cierre, funcionó como una etapa de síntesis e integración, en la que se organizaron los conocimientos, decisiones y evidencias acumuladas a lo largo del proceso.

Durante esta fase, el trabajo se centró en la preparación de los entregables finales: documentación final y anexos. El proceso mantuvo la lógica iterativa e incremental del ciclo de vida: cada capítulo y documento fue elaborado, revisado y mejorado en sucesivas versiones, incorporando observaciones del tutor y del propio equipo de manera incremental.

Esta dinámica permitió garantizar la coherencia entre las distintas partes del proyecto y asegurar que la documentación reflejara fielmente la realidad del sistema construido. Al mismo tiempo, sirvió como espacio de reflexión sobre las decisiones tomadas, los desafíos enfrentados y los aprendizajes obtenidos durante el desarrollo.

5. Gestión de proyecto

En este capítulo se presenta cómo se gestionó el proyecto Fitit Returns a lo largo de sus diferentes etapas. Se describe el cronograma general, la forma en que se distribuyó el esfuerzo, la estructura de trabajo elegida y las fases que lo conformaron. Además, se explican los mecanismos utilizados para organizar las tareas y controlar el tiempo, así como las herramientas que apoyaron la coordinación del equipo.

5.1. Cronograma

El cronograma del proyecto se planificó en tres fases principales: descubrimiento, desarrollo y documentación, distribuidas a lo largo de un año de trabajo.

- **Fase de descubrimiento:** 8 de Octubre de 2024 - 16 de Diciembre de 2024
- **Fase de desarrollo:** 16 de Diciembre de 2024 - 6 de Septiembre de 2025
- **Fase de documentación:** 6 de Septiembre de 2025 - 16 de Octubre de 2025

Este cronograma permitió organizar el trabajo en etapas claramente diferenciadas, garantizando que cada fase produjera insumos concretos para la siguiente y que los avances pudieran evaluarse de forma continua mediante entregas parciales.

Ilustración 19 - Cronograma

5.2. Análisis del esfuerzo

Durante todo el proyecto se buscó llevar un control detallado del tiempo invertido. Para ello se utilizó la herramienta Clockify, que permitió clasificar las horas de trabajo según las tareas realizadas y luego obtener reportes automáticos que facilitaron el análisis. Esta práctica aseguró que la planificación inicial pudiera contrastarse con datos concretos y no solo con estimaciones.

El esfuerzo total ascendió a **2.326 horas**, distribuidas en tres etapas bien diferenciadas.

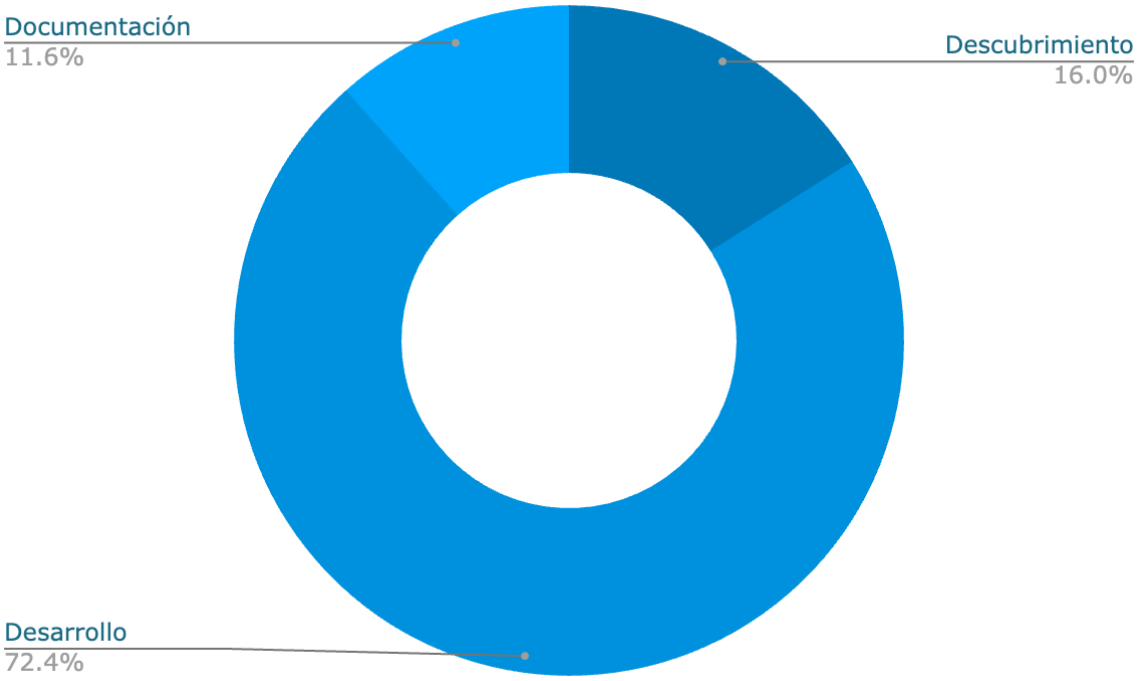


Ilustración 20 - Distribución de horas por fase

La **fase de descubrimiento** supuso **373 horas** a lo largo de 12 semanas, en las que predominó la captura y análisis de requerimientos, la definición de la arquitectura preliminar y la coordinación con los distintos actores. Si bien fue la etapa más breve, resultó clave para orientar el rumbo del proyecto.

En la **fase de desarrollo** se concentró la mayor carga de trabajo, con **1.685 horas** en 36 semanas. Esta etapa incluyó la implementación del sistema, realización de pruebas y actividades de aseguramiento de calidad, junto con reuniones periódicas para revisar avances y resolver incidencias. El alto porcentaje de horas dedicadas a esta fase refleja lo técnico del proyecto y confirma que la mayor parte del esfuerzo se destinó a la construcción efectiva de la solución.

Finalmente, la **fase de documentación** sumó **268 horas** en 6 semanas. Aunque su peso en el total fue menor, requirió una intensidad de dedicación mayor por semana, ya que implicó preparar la documentación final y anexos.

En la Ilustración 20 puede observarse esta distribución: el **72,4%** del tiempo se asignó al desarrollo, mientras que el **16%** correspondió al descubrimiento y el **11,6%** a la documentación. Esta proporción pone en evidencia que, aunque la construcción del sistema absorbió la mayor parte del esfuerzo, las etapas inicial y final cumplieron un papel fundamental al proveer los insumos necesarios y consolidar el resultado del proyecto.

5.3. Fase de descubrimiento

En los primeros meses de trabajo, el equipo destinó sus esfuerzos a la fase de descubrimiento, con el objetivo de comprender el dominio del problema y establecer

las bases conceptuales y técnicas del sistema Fitit Returns. Esta etapa resultó fundamental para ordenar las ideas iniciales, identificar los requerimientos esenciales y definir la dirección del proyecto. Entre las actividades más relevantes se incluyeron la **ingeniería de requerimientos**, la **definición de la arquitectura** preliminar, la elaboración de un plan de **gestión de riesgos**, el **plan de calidad** y múltiples instancias de **reuniones** con el cliente, tutor y terceros vinculados a posibles integraciones.

En esta fase también se llevó a cabo la especificación y validación del problema, junto con la definición de los procesos de negocio relacionados con devoluciones y cambios en el comercio electrónico. La validación de los requerimientos se realizó mediante reuniones con el cliente y terceros (tiendas y cadeterías), con el fin de asegurar una comprensión compartida sobre el alcance de la solución y confirmar que los objetivos iniciales respondieran a las necesidades reales de las tiendas y usuarios finales.

Todas estas actividades no solo marcaron el rumbo del desarrollo, sino que también representaron resultados clave para la planificación de la siguiente etapa.

5.3.1. Adaptación de Kanban

Para organizar las tareas de esta fase, se optó por aplicar la metodología ágil **Kanban**, que resultó especialmente adecuada para un escenario en el cual las tareas eran cambiantes y muchas veces dependían de información aún en validación. El tablero permitió visualizar de forma sencilla el estado de cada actividad y garantizar un flujo de trabajo continuo.

El tablero se estructuró en cuatro columnas principales:

- To Do
- In Progress
- Done
- Cancelled

El uso de la columna Cancelled resultó útil para mantener la trazabilidad de decisiones, ya que permitió visualizar qué actividades habían sido desestimadas y documentar los motivos de su cancelación. De este modo, el equipo evitó perder de vista las consideraciones iniciales y pudo justificar los cambios de rumbo cuando fue necesario. También, uno de los principios aplicados en este tablero fue el límite de tareas en progreso *(*WIP – Work In Progress*)*, de manera que cada integrante solo podía tener asignada una actividad a la vez. Esto ayudó a prevenir cuellos de botella y fomentó la finalización de tareas antes de asumir nuevas responsabilidades.

El seguimiento de las actividades se realizó de manera colaborativa y en línea, a través de ClickUp, herramienta que ofreció una interfaz clara y accesible para todo el equipo. Cada semana se llevaban a cabo reuniones internas en las que se repasaba el estado de las tareas, se resolvían bloqueos y se re-priorizaban actividades según la evolución del proyecto.

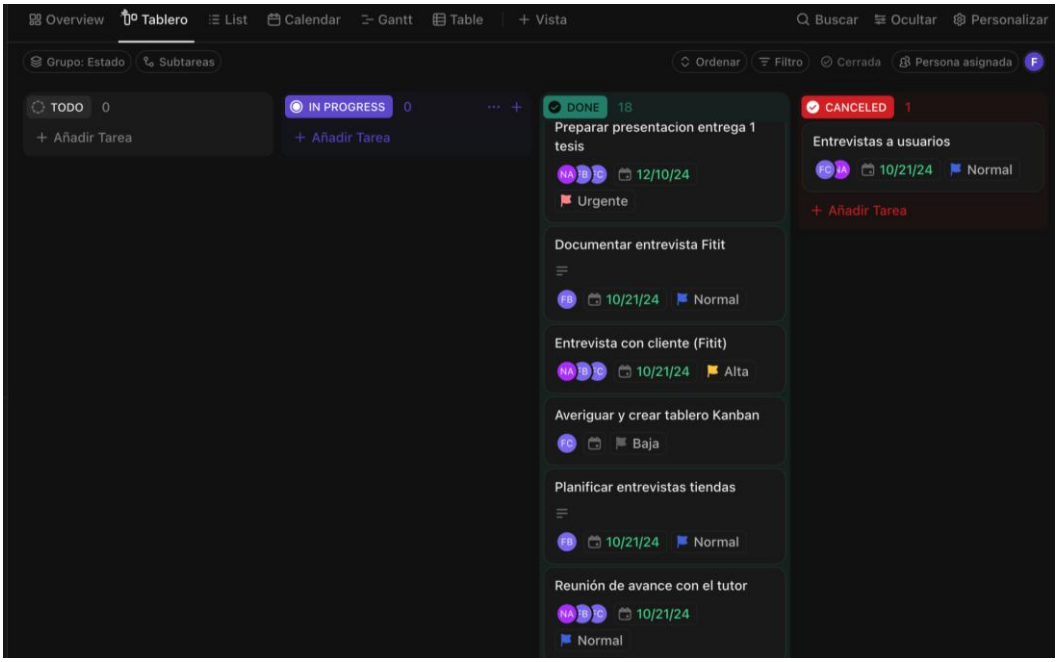


Ilustración 21 - Tablero Kanban de ClickUp

5.3.2. Distribución del esfuerzo

El análisis de la fase de descubrimiento permitió visualizar cómo se distribuyeron las **373 horas** de trabajo de esta etapa.

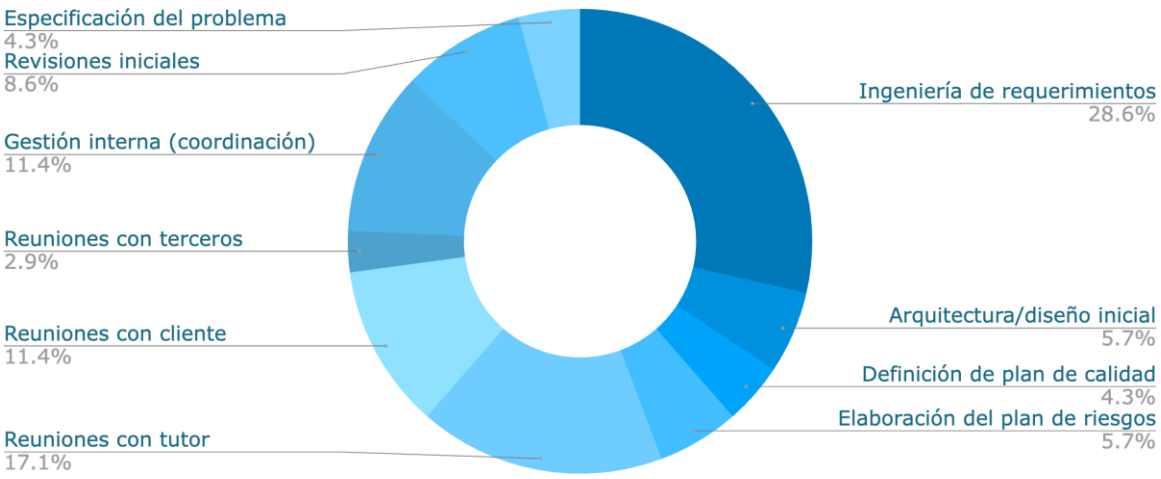


Ilustración 22 - Distribución de esfuerzo en la fase de descubrimiento

Uno de los primeros aspectos que se puede destacar es la cantidad de tiempo que se destinó a las **instancias de comunicación**, tanto en reuniones con el tutor, cliente, terceros y dentro del propio equipo. Este resultado era esperable en una etapa inicial, donde las decisiones más relevantes debían ser validadas y discutidas de manera frecuente. La metodología ágil adoptada favoreció este tipo de dinámicas, priorizando la interacción entre las personas por sobre la producción de documentación extensa. La inversión de horas en estas actividades permitió alinear expectativas, resolver

dudas de manera temprana y validar el rumbo del proyecto antes de avanzar hacia fases posteriores.

Otro bloque importante de esfuerzo se concentró en la **gestión interna**, que incluyó la organización del tablero, la definición de herramientas de trabajo, la coordinación entre los integrantes y la planificación inicial. Este tipo de tareas, aunque no aportan directamente funcionalidades visibles, fueron clave para dar claridad al equipo respecto a la forma de trabajo, los criterios de calidad y los pasos a seguir.

En paralelo, una parte significativa del tiempo se destinó a la **ingeniería de requerimientos**, que representó el núcleo de esta fase. Allí se trabajó en identificar los procesos de devolución y cambio en el contexto de *e-commerce*, así como en relevar los requerimientos funcionales, los no funcionales y las restricciones del sistema. El esfuerzo invertido en esta actividad resultó fundamental, ya que proporcionó al equipo una comprensión profunda del problema y permitió construir una base sólida sobre la cual se apoyaría todo el desarrollo.

Finalmente, actividades como la definición del **plan de calidad**, el diseño de la **arquitectura**, la elaboración del plan de **gestión de riesgos** y la primera revisión académica complementaron la fase, asegurando que además de entender el problema, el equipo cuente con lineamientos técnicos y organizativos claros. Estas tareas no representaron el mayor volumen de horas, pero su impacto fue relevante en términos de prevención de problemas futuros y establecimiento de estándares de trabajo.

5.4. Fase de desarrollo

En la fase de desarrollo se decidió trabajar con **Scrum**, dado que esta metodología se adapta mejor a las necesidades de construcción del sistema. Mientras que Kanban resultó apropiado en la etapa anterior, cuando predominaban tareas exploratorias y de difícil estimación, en esta fase se necesitaba un marco que ofreciera estructura, planificación temporal y entregas incrementales verificables.

Las principales razones para adoptar Scrum en esta etapa fueron:

- La posibilidad de planificar iteraciones cortas con metas definidas, en lugar de trabajar con un flujo continuo sin plazos.
- La entrega de incrementos funcionales, que aseguraron avances visibles y validaciones periódicas con el tutor y el cliente.
- El soporte de artefactos y métricas propias (como el Sprint Backlog, la Definition of Done o los burndown charts), que aportaron transparencia y seguimiento detallado del progreso.
- La presencia de eventos de revisión y retrospectiva, que favorecieron la comunicación, la mejora continua y la detección temprana de problemas.

Estas características hicieron que Scrum resultara más beneficioso para la etapa de desarrollo, al proporcionar orden, control y una dinámica **iterativa-incremental** que

permitió cumplir los objetivos del proyecto de forma sostenida y con resultados verificables en cada ciclo.

A partir de esta fase se migró la gestión del proyecto a Jira y el registro de tiempo a Clockify, reemplazando ClickUp porque la suscripción gratuita ofrecía un límite de memoria que el equipo consumió por completo en la primera etapa.

5.4.1. Roles de Scrum

Los roles personales del equipo se describen en la sección 4.2. Roles definidos en el equipo, donde se detallan las responsabilidades técnicas y organizativas de cada integrante. Durante la fase de desarrollo, estos mismos integrantes asumieron las funciones propias del marco Scrum, adaptadas a la escala y características del proyecto.

- **Product Owner (PO):**
  El rol fue asumido por **Federico**, quien trasladó su conocimiento del dominio del negocio y de las necesidades del cliente al proceso de desarrollo.

- **Scrum Master (SM):**
  Florencia desempeñó este rol, garantizando que el proceso de trabajo se mantuviera alineado con los principios ágiles. Se encargó de facilitar la coordinación, promover la mejora continua y resolver los impedimentos que pudieran afectar el avance del equipo.

- **Equipo de desarrollo:**
  Integrado por los tres miembros del proyecto, responsables de implementar las funcionalidades, realizar las pruebas, documentar los resultados y revisar el código de forma colaborativa.

El equipo mantuvo una estructura pequeña y multifuncional, donde todos participaron activamente en el desarrollo y en las decisiones técnicas.

5.4.2. Artefactos

A continuación, se describen los principales artefactos empleados en la fase de desarrollo:

Product Backlog
El Product Backlog es la lista priorizada de todos los requerimientos y funcionalidades que debía contemplar el sistema. En el caso de Fitit Returns, este *backlog* se construyó en la fase de descubrimiento y fue enriquecido a lo largo del proyecto. Contenía épicas y *user stories* detalladas, cada una con criterios de aceptación definidos. Jira facilitó su gestión, ya que permitió ordenar las historias por prioridad, asignarlas a *releases* y vincularlas con épicas (ver Anexo 12.3. Product Backlog).

Sprint Backlog
En cada planificación de sprint se seleccionaban las historias más relevantes del Product Backlog y se trasladaban al Sprint Backlog. Allí se desglosaban en tareas más pequeñas, lo que facilitaba la estimación y asignación de responsabilidades. Este

artefacto permitió que cada *sprint* tuviera un objetivo claro y alcanzable. En Jira, el Sprint Backlog se representaba directamente en el tablero, mostrando qué tareas estaban pendientes, en progreso o completadas.

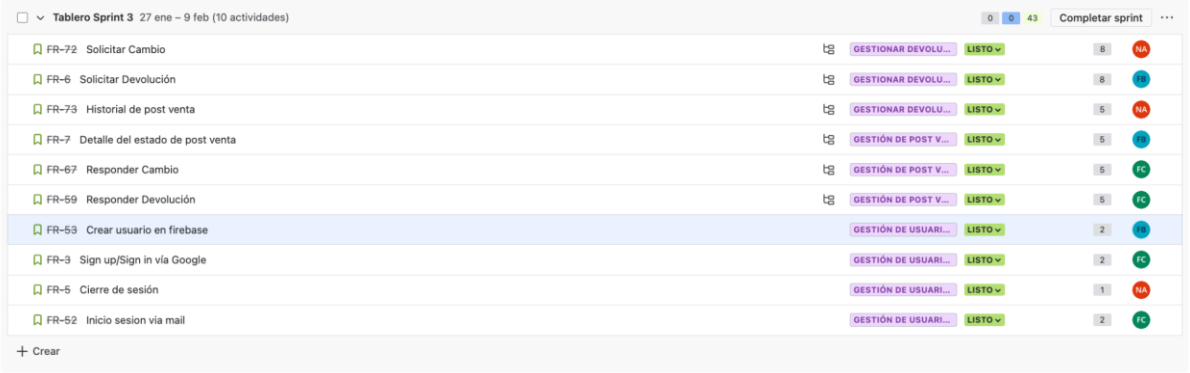


Ilustración 23 - Sprint Backlog

Épicas

Las épicas agruparon grandes bloques de funcionalidades. Por ejemplo, todo lo relacionado a la creación, seguimiento y validación de devoluciones se centralizó en la épica “Gestionar devoluciones y cambios”. De esta manera, se podía dar trazabilidad a las historias vinculadas y medir el avance de cada bloque en particular. Jira permitió visualizar el estado de las épicas en un *roadmap*, lo que resultó útil para comunicar avances al tutor y al cliente.

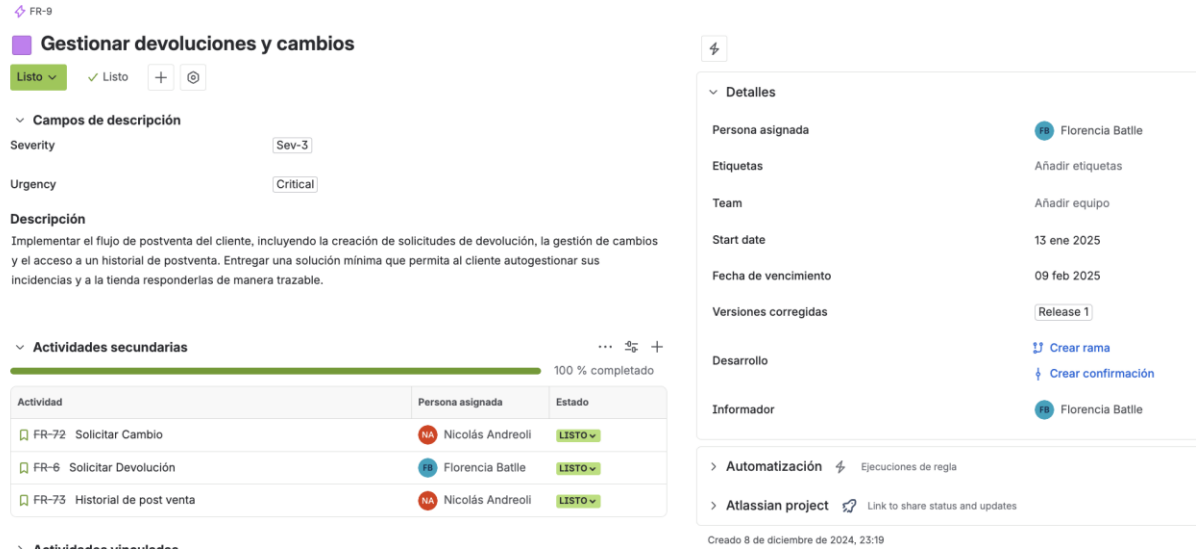


Ilustración 24 - Épica

User Stories

Las *user stories* son el mínimo aporte de valor a un usuario. Cada una describía una funcionalidad desde la perspectiva del usuario, por ejemplo: “El usuario debe poder iniciar una solicitud de devolución indicando el motivo y subiendo una foto como evidencia”.

En Jira, las historias se creaban siguiendo un formato estandarizado que incluía la descripción en el formato estándar:

Como [rol], quiero [acción], para [beneficio]

La historia también se componía por los **criterios de aceptación**, la **urgencia**, el **impacto**, la **estimación** en *story points* y las actividades secundarias vinculadas a tareas de *frontend* y *backend*. Este nivel de detalle permitió que cada historia fuera fácilmente comprendida por cualquier integrante del equipo y que se mantuviera trazabilidad entre los requerimientos y el desarrollo.

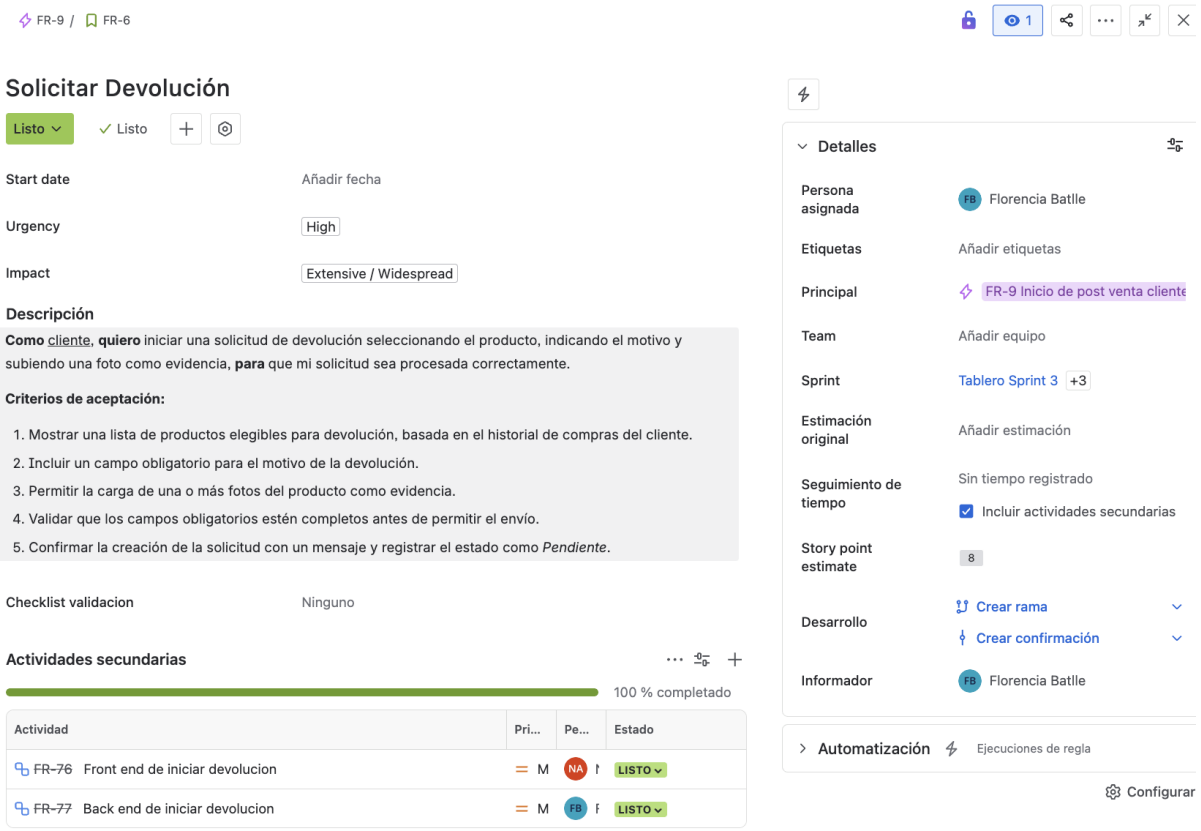


Ilustración 25 - Historia de usuario

Tablero
El tablero de Jira fue la herramienta principal de trabajo diario. Se configuró con las columnas:

- Por hacer
- En Análisis
- En Desarrollo
- En Validación
- Listo
- Descartada

Este último estado se incorporó específicamente para registrar aquellas tareas descartadas por cambios en el alcance, lo que permitió mantener un historial completo de decisiones. El tablero facilitó la transparencia, ya que cualquier integrante podía ver en qué estado se encontraba cada tarea.

Producto incrementado
El producto incrementado fue el resultado de cada *sprint*: un sistema que acumulaba las funcionalidades entregadas en ciclos anteriores más las nuevas historias completadas. Al finalizar cada sprint se validaba este incremento, garantizando que el *software* estuviera siempre en condiciones de ser utilizado.

5.4.3. Eventos

En el marco de **Scrum**, los eventos constituyen instancias fundamentales para organizar, revisar y mejorar el trabajo del equipo. Durante la fase de desarrollo de Fitit Returns se mantuvieron los eventos principales, aunque con algunas adaptaciones a la dinámica de un equipo reducido y a las necesidades particulares del proyecto.

Sprints
Antes de iniciar los *sprints* regulares, el equipo realizó un ***sprint* cero**, cuya duración fue excepcionalmente de tres semanas, desarrollándose durante las últimas semanas de Diciembre. Su propósito fue preparar el entorno técnico y organizativo del proyecto, incluyendo la configuración inicial del *monorepo* (ver detalles en la sección 9.3.1. *Monorepo*) y las herramientas principales de trabajo. Además, se priorizó el *backlog* y se generó el Sprint Backlog para los primeros dos *sprints*. Este *sprint* funcionó como una etapa de transición entre la fase de descubrimiento y el desarrollo iterativo, asegurando que el equipo comience con una base técnica estable y un *backlog* inicial claro.

El equipo decidió estructurar la fase de desarrollo en *sprints* de **dos semanas**. Este ciclo fijo permitió dar un ritmo constante al trabajo y aseguró que cada iteración tuviera un objetivo claro. Al inicio de cada *sprint* se establecían las metas a alcanzar y al final se entregaba un incremento del producto listo para ser validado. La duración de dos semanas se consideró óptima porque era lo suficientemente corta como para detectar

problemas rápidamente, pero lo bastante larga como para completar historias de usuario de valor.

Sprint Planning

Cada *sprint* comenzaba con una reunión de planificación, en la cual se seleccionaban las *user stories* del Product Backlog que pasarían al Sprint Backlog. Durante esta instancia se estimaba el esfuerzo requerido en *story points* utilizando la técnica de Planning Poker [16], apoyados en herramientas en línea [17]. Cada integrante participaba en la estimación y la asignación de tareas, lo que fomentaba la colaboración y compromiso colectivo. El resultado de la Sprint Planning era un *backlog* acotado y un objetivo de *sprint* definido.

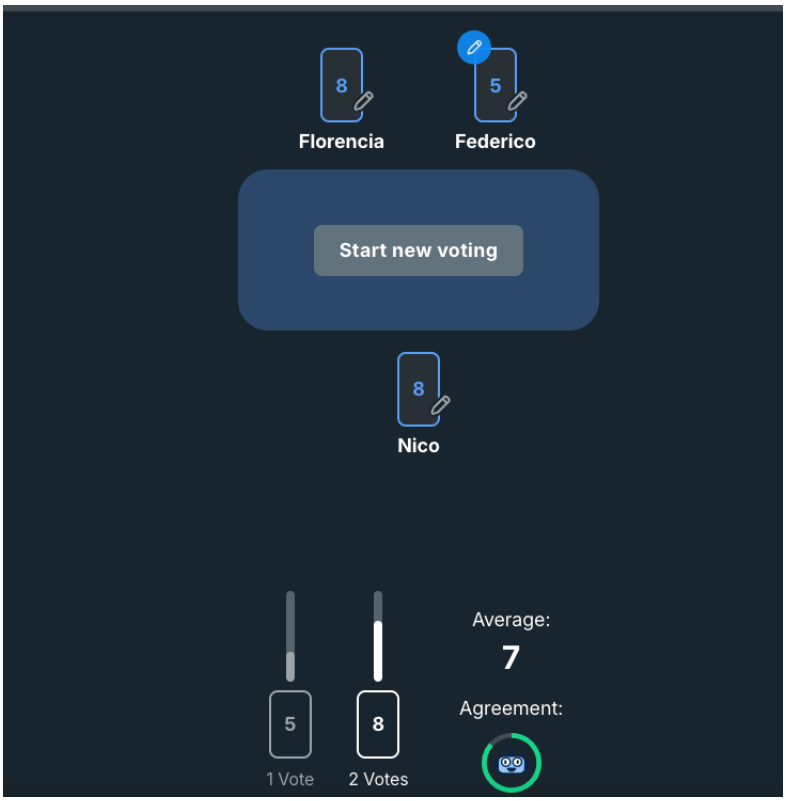


Ilustración 26 - Planning poker

Bi-Weekly Standups

A diferencia de Scrum tradicional, el equipo decidió reemplazar las *daily meetings* por reuniones de coordinación dos veces por semana. La razón principal fue que, en un equipo pequeño, no siempre había avances relevantes para reportar de forma diaria. Estas reuniones se realizaban en formato virtual y tenían una duración breve, en las que **cada integrante comentaba qué había hecho, qué estaba haciendo y si tenía algún inconveniente**. La comunicación cotidiana se reforzaba con un grupo de WhatsApp, donde cada miembro podía compartir avances o levantar alertas sin necesidad de esperar al encuentro formal.

Sprint Review

Al finalizar cada *sprint*, el equipo realizaba una Sprint Review en la que se mostraba el incremento desarrollado. Estas revisiones se hicieron regularmente con el tutor y, en algunos hitos clave, también con el cliente. La revisión permitió validar que las

funcionalidades implementadas cumplían con los **criterios de aceptación** y recibir ***feedback*** temprano. Cuando surgían nuevas necesidades o ideas de mejora, estas se discutían primero dentro del equipo y se evaluaban junto al cliente para analizar su relevancia, alcance e impacto en las entregas en curso. Una vez acordado su valor y prioridad, se decidía si incorporarlas al *sprint* actual o registrarlas en el Product Backlog para ser consideradas en futuras iteraciones. Este proceso permitió mantener un control sobre los cambios y asegurar que cada incorporación respondiera a una necesidad validada.

Sprint Retrospective

Las retrospectivas no se realizaron al final de cada *sprint*, sino que fueron realizadas **cada cuatro iteraciones**. Esta decisión respondió a la intención de evitar reuniones sin contenido real y centrarse en instancias con valor. En estas retrospectivas se discutía qué estaba funcionando bien, qué debía mejorarse y qué acciones concretas podrían aplicarse en los siguientes sprints. A pesar de la frecuencia reducida, cada retrospectiva generó cambios importantes en la dinámica de trabajo, como ajustes en la distribución de tareas o mejoras en la comunicación interna.

Ilustración 27 - Retrospectiva *sprint* 1

Estos eventos adaptados mantuvieron el espíritu de Scrum: fomentar la transparencia, inspección y adaptación. Las planificaciones y revisiones garantizaron la entrega incremental del producto, los *standups* bisemanales aseguraron la coordinación del equipo sin sobrecargarlo, y las retrospectivas permitieron mejorar continuamente el proceso de manera pragmática.

5.4.4. Ejecución del *Release Plan*

Al ser un proyecto de *software* con un enfoque ágil, es natural que se produjeran cambios en el proceso y en el alcance a medida que avanzaba el desarrollo. La capacidad de adaptación fue fundamental, ya que permitió que las prioridades pudieran redefinirse según los aprendizajes de cada *sprint* y el *feedback* recibido en

las instancias de validación. Este dinamismo, lejos de ser un problema, fue una de las fortalezas del proceso, ya que el grupo pudo ajustar el sistema de manera temprana a las necesidades reales de las tiendas y de los usuarios finales.

Inicialmente, se estableció un *MVP* con las funcionalidades más básicas y de mayor prioridad: la gestión de devoluciones desde el lado del cliente y la aprobación de las mismas por parte de la tienda. El objetivo era contar rápidamente con un flujo mínimo funcionando, de forma que las tiendas pudieran probar el sistema en etapas tempranas y el equipo pudiera empezar a recoger *feedback*. Esto aseguró que la solución no quede solamente en un plano teórico, sino que desde el primer *release* existiera un producto funcional sobre el cual iterar.

A partir de este *MVP*, el *backlog* fue evolucionando hacia *releases* más completos, cada uno de ellos con objetivos específicos. En total se planificaron cuatro *releases*, que marcaron hitos en la evolución del proyecto:

- **Release 1 (08/03/2025):** *MVP* básico, centrado exclusivamente en la creación de solicitudes de devolución por parte del cliente y la aprobación por parte de la tienda. No incluyó políticas, reportes ni notificaciones.

- **Release 2 (03/05/2025):** Incorporación del sistema de créditos de tienda, que surgió como un requerimiento adicional. Este cambio representó una ampliación significativa del alcance e impactó la arquitectura inicial. Además, se realizaron mejoras en la validación de devoluciones.

- **Release 3 (12/07/2025):** Inicio de las políticas de devolución configurables por tienda, con reglas básicas de plazos y excepciones. Se añadieron validaciones más avanzadas y nuevas integraciones externas, y se implementaron las primeras notificaciones por email.

- **Release 4 (06/09/2025):** Consolidación de las funcionalidades previas y el desarrollo de *webhooks* y Public API. Además, se tomó la decisión de realizar la migración a una nueva librería de UI, lo que implicó rehacer parte de la interfaz para mejorar su usabilidad. Este *release* dejó el sistema en condiciones estables para pasar a la fase de documentación.

La gestión de estos *releases* combinó dos enfoques complementarios. En los primeros se aplicó un modelo *feature driven* [18], donde la prioridad fue liberar funcionalidades clave para validar el producto, aunque eso implicara flexibilidad en las fechas. En los últimos releases, en cambio, se aplicó un enfoque *deadline driven* [19], donde las fechas de entrega se volvieron prioritarias y el alcance se ajustaba a la capacidad disponible. Esta combinación resultó fundamental para asegurar entregas tempranas de valor y, al mismo tiempo, cumplir con los tiempos establecidos para la documentación final sin comprometer la calidad.

En el Anexo 12.4. Funcionalidades por *release* y descartadas están las evidencias de las entregas de las funciones principales y las descartadas en el proceso.

Algunas de las funcionalidades implementadas en los *releases* (gestión de créditos y migración de librería de UI) fueron fruto de **cambios en el alcance**, un proceso que va a ser detallado en la sección 6.6. Gestión del cambio.

5.4.5. Métricas

***Story points* estimados vs. realizados**

Como se mencionó anteriormente, cada historia de usuario se estimó en *story points* lo que permitió al equipo tener una referencia relativa de esfuerzo y complejidad. Una vez concluido cada *sprint*, se comparan los puntos estimados con los efectivamente realizados. Esta práctica sirvió no solo para medir productividad, sino también para aprender a estimar con mayor precisión.

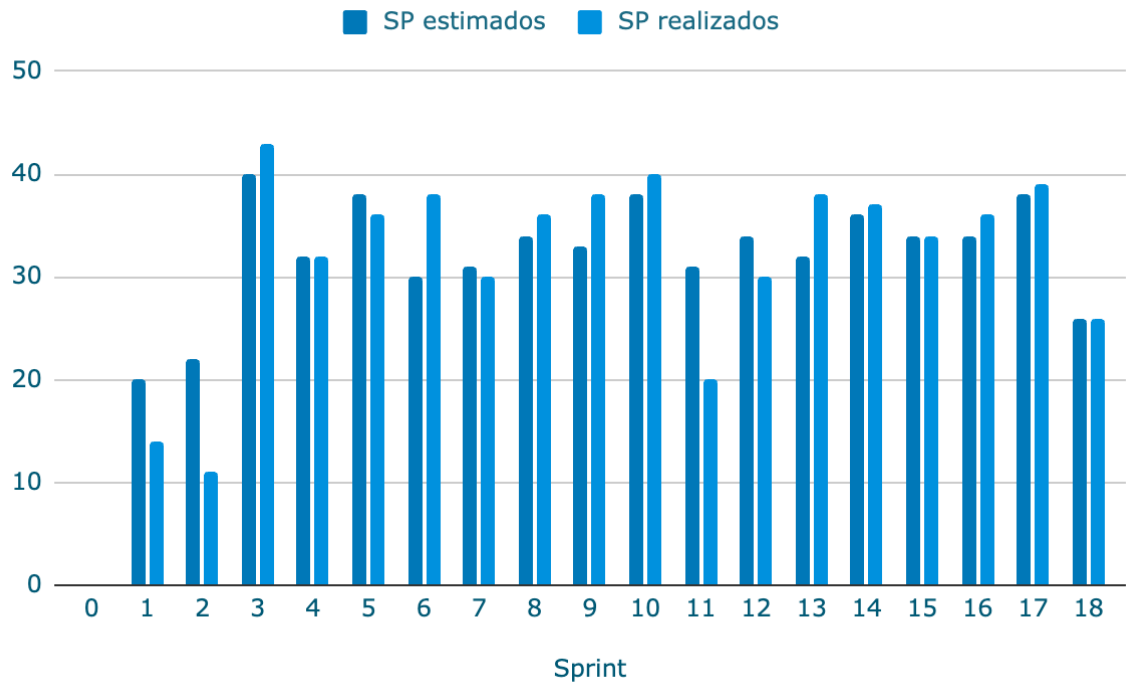


Ilustración 28 - Gráfica de *story points* estimados vs. *story points* realizados

En los primeros *sprints* se observó una tendencia a subestimar el esfuerzo requerido, lo que provocó que varias tareas quedarán incompletas. Con el tiempo, el equipo mejoró la calibración de sus estimaciones, alcanzando una mayor coherencia entre lo planificado y lo logrado. El *sprint* 11 representó una excepción, con un desvío alto entre los *story points* estimados y los realizados, debido a la menor disponibilidad del equipo durante ese período por compromisos académicos. Esta situación fue puntual y no afectó la tendencia general de mejora en las estimaciones. En conjunto, la gráfica refleja un proceso de aprendizaje progresivo y de ajuste continuo, propio de la naturaleza iterativa del enfoque ágil.

Burndown chart

Durante el desarrollo, se utilizaron *burndown charts* para visualizar cómo descendía el trabajo pendiente a lo largo de los días de cada *sprint*. Esta herramienta permitió identificar rápidamente desvíos respecto a la curva ideal y tomar decisiones de corrección en tiempo real.

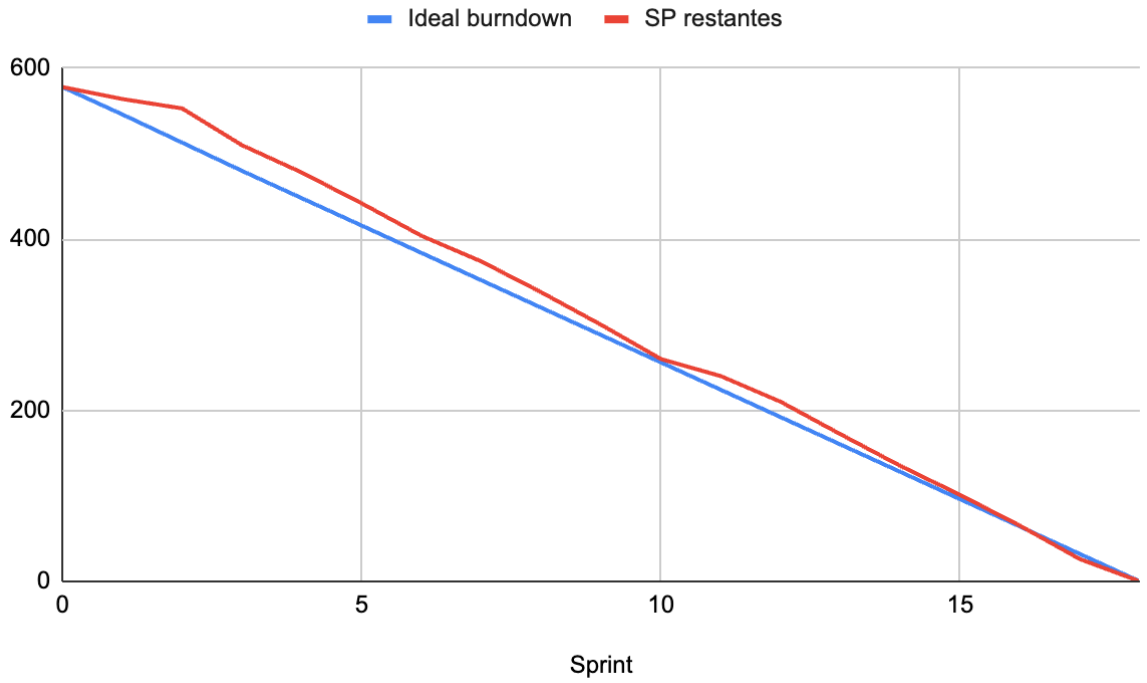


Ilustración 29 - Burndown chart por Sprint

Cuando la curva mostraba que el ritmo de avance era más lento que lo esperado, el equipo podía redistribuir tareas, redefinir prioridades o incluso ajustar el alcance del *sprint*. Por el contrario, cuando la curva descendía alineada con la planificación, se confirmaba que el flujo de trabajo estaba bajo control. En este sentido, los *burndown charts* actuaron como una brújula para asegurar que el esfuerzo diario estuviera orientado al cumplimiento de los objetivos del proyecto.

Releases como hitos de control
Más allá del seguimiento en *sprints*, el ciclo de vida se midió también a nivel de *releases*, que funcionaron como puntos de control macro. Cada *release* tenía un objetivo específico y su cumplimiento permitió validar la evolución del sistema de manera incremental. Nuestro objetivo era seguir el *Release Plan* que se puede ver en el Anexo 12.6. *Release Plan*.

5.4.6. Distribución de esfuerzo

En la etapa de desarrollo, la mayor parte del esfuerzo estuvo dedicada a la implementación del sistema y a la corrección de errores detectados en cada *release*.

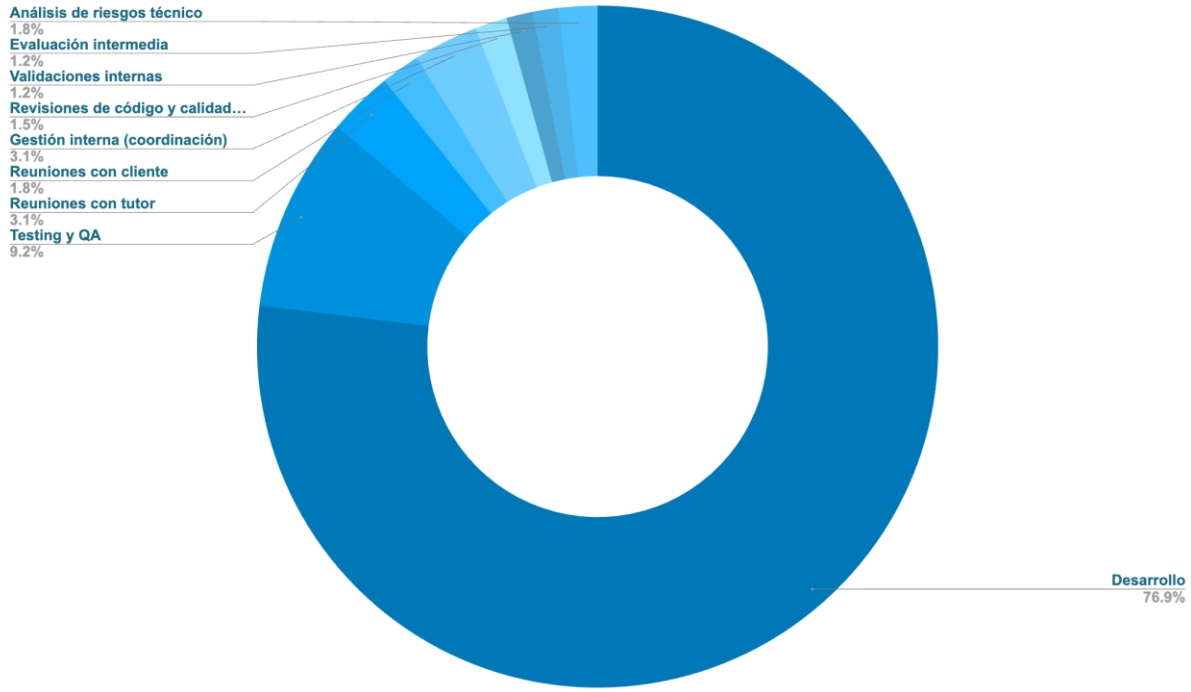


Ilustración 30 - Esfuerzo en la fase de desarrollo

Dentro de la categoría de desarrollo se incluyó el **desarrollo del sistema** y numerosos *fixes* que surgieron a medida que se validaban los incrementos. Otra parte importante del trabajo se centró en la estabilización de las entregas, lo que implicó resolver errores detectados en ambientes de pruebas y accionar sobre comentarios de las validaciones.

El desarrollo también abarcó tareas de **automatización** y **control de calidad**, como la creación de *tests*, revisiones de código entre pares y validaciones internas, que fueron clave para mantener la solidez de la solución a medida que el proyecto crecía en complejidad. En paralelo, se llevaron adelante tareas de coordinación interna, reuniones con el tutor y el cliente, que ayudaron a mantener el *backlog* actualizado, resolver dudas de alcance y alinear la priorización de funcionalidades en cada *sprint*.

5.5. Fase de documentación

La fase de documentación representó la instancia final del proyecto y tuvo como propósito consolidar todos los entregables técnicos y académicos. En esta etapa, el trabajo se organizó de manera incremental, permitiendo que los avances se integren progresivamente a la documentación final de tesis.

Las actividades realizadas abarcaron la redacción del informe principal, la preparación de anexos y evidencias, la elaboración de manuales de usuario e instalación y la documentación de la arquitectura del sistema. Además, se destinaron esfuerzos importantes a las revisiones cruzadas entre integrantes, que facilitaron la detección temprana de inconsistencias, así como a las correcciones derivadas del *feedback* del tutor y de los procesos de validación.

5.5.1. Adaptación de Kanban

Al igual que en la fase de descubrimiento, el equipo utilizó un tablero **Kanban** para organizar el trabajo de documentación. En este caso, la naturaleza de las tareas -escritura, revisión, integración y correcciones- hacía poco práctico planificarlas en iteraciones cortas con Scrum. El enfoque Kanban permitió dar flujo continuo al trabajo, priorizando tareas a medida que surgían observaciones o nuevas secciones a completar.

El tablero se estructuró con columnas que representaban los distintos estados del trabajo:

- Por hacer
- En curso
- Revisión cruzada
- Revisión tutor
- Correcciones
- Listo

Esto facilitó visualizar en todo momento qué se estaba escribiendo, qué estaba en revisión y qué partes ya estaban finalizadas. Al igual que en fase de investigación, una regla importante fue limitar el *WIP* (*Work In Progress*), lo que evitó que varios integrantes estuvieran corrigiendo la misma sección al mismo tiempo y ayudó a mantener el orden en los avances.

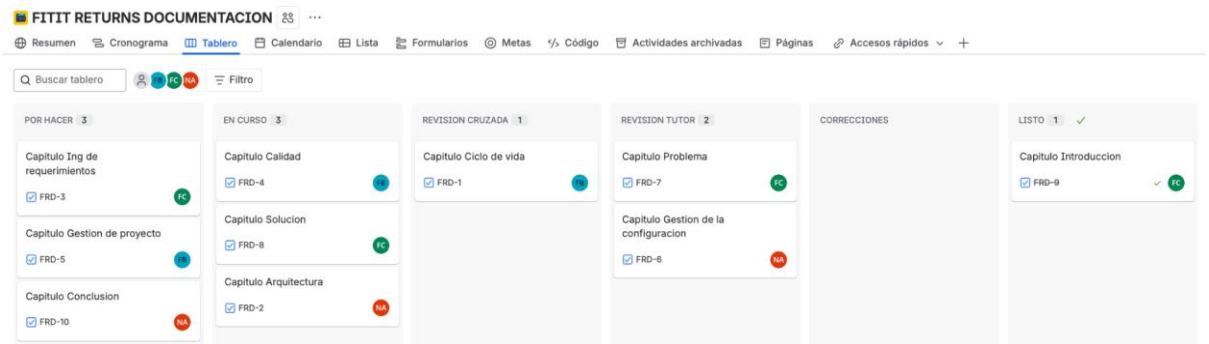


Ilustración 31 - Tablero Kanban para fase de documentación

5.5.2. Distribución del esfuerzo

Un aspecto clave fue que la documentación no se trató únicamente de escribir, sino de garantizar la coherencia entre las distintas piezas del proyecto. La distribución de esfuerzo refleja el tiempo invertido en reuniones de equipo, que permitieron coordinar tareas y asegurar que los anexos técnicos, las decisiones de arquitectura y las explicaciones metodológicas quedarán correctamente integradas. Finalmente, se

realizaron ajustes de formato y calidad para cumplir con los requisitos formales de la universidad.

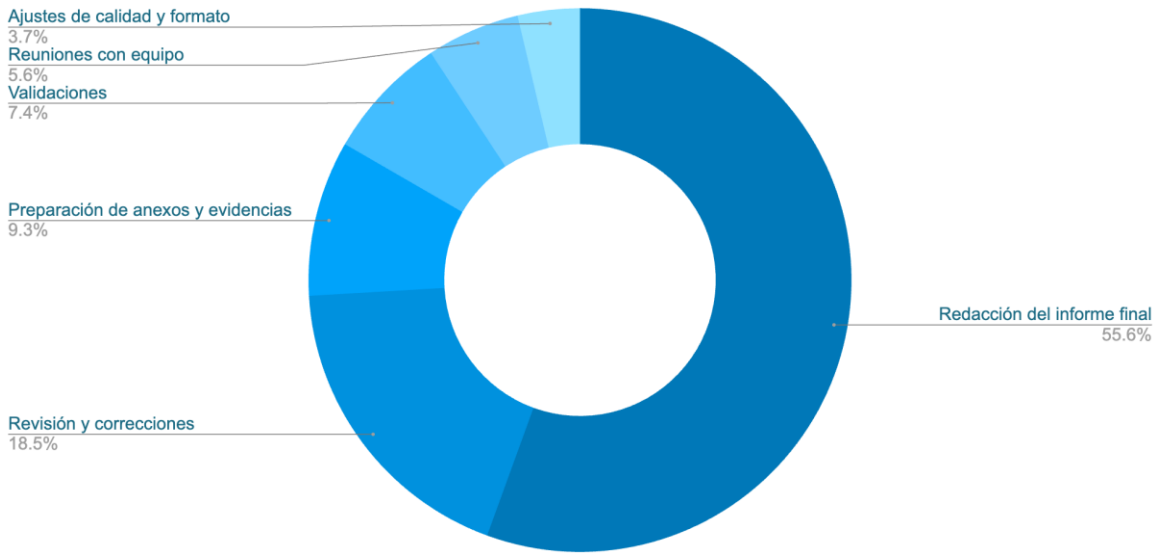


Ilustración 32 - Esfuerzo en la fase de documentación

5.6. Comunicación y coordinación del equipo

Reuniones y comunicación virtual
En el capítulo 9. Gestión de la configuración se detallan las herramientas que se utilizaron para la comunicación online con el cliente, el tutor y entre el equipo.

Reuniones presenciales
En las primeras etapas del proyecto, especialmente durante la fase de descubrimiento, se realizaron reuniones presenciales con el cliente en las oficinas de Fitit (ver Anexo 12.29. Reunión en oficina de Fitit). Estas instancias fueron útiles para llevar adelante entrevistas, actividades de ingeniería de requerimientos y validaciones iniciales. Con el avance del proyecto, las reuniones presenciales se combinaron con las virtuales, dependiendo de la disponibilidad de las partes.

5.7. Gestión del riesgo

En Fitit Returns la gestión de riesgos fue un proceso constante que buscó anticipar situaciones que podían afectar al proyecto y definir de antemano cómo actuar en caso de que se presentaran. Para identificarlos, el equipo recurrió a dos prácticas simples: revisar la documentación existente y realizar sesiones de *brainstorming*. Esto permitió detectar riesgos probables desde el inicio (como la sincronización del equipo o los cambios en requerimientos), pero también anticipar otros que se hicieron visibles recién cuando avanzó el desarrollo.

5.7.1. Identificación y análisis de riesgos

Para evaluar los riesgos del proyecto se realizó un **análisis cualitativo** en el que se consideró, para cada uno, la probabilidad de que ocurriera y el impacto que tendría sobre los objetivos del proyecto. Esta evaluación permitió estimar cómo cada riesgo podría afectar aspectos clave como el costo, el tiempo, el alcance y la calidad.

El análisis se llevó a cabo de forma **mensual**, revisando y actualizando los valores a medida que avanzaban las etapas del proyecto. A partir de estas revisiones se calculó la magnitud del riesgo, combinando la probabilidad y el impacto, y se los clasificó en tres niveles: bajo, medio o alto.

Las escalas de medición utilizadas y el detalle del proceso de análisis se incluyen en el Anexo 12.7. Escalas y metodología de análisis de riesgos.

Según la magnitud estimada, se definieron distintas acciones de respuesta: cómo eliminar el riesgo, transferirlo a un tercero, mitigarlo mediante medidas preventivas o aceptarlo cuando su impacto era bajo. El resultado de este trabajo puede verse en el Anexo 12.8. Análisis cualitativo de riesgos.

5.7.2. Monitoreo de riesgos principales

RK1 – Problemas de sincronización del equipo

Este riesgo estuvo presente sobre todo en los primeros meses del proyecto. Durante Noviembre y Diciembre de 2024, cuando se realizaban las primeras tareas de descubrimiento y de preparación técnica, el equipo aún no había consolidado sus mecanismos de coordinación. La falta de rutinas claras llevó a que algunos integrantes avanzaran en paralelo sin suficiente alineación, lo que ocasionó pequeños cuellos de botella.

La magnitud del riesgo se intensificó en enero de 2025 debido a un factor externo: dos integrantes del equipo, Nicolás y Florencia, se encontraban de viaje, lo que redujo la capacidad operativa y dificultó la sincronización de avances. Esto provocó que ciertas tareas quedarán estancadas, ya que dependían directamente de la participación de esos miembros.

La mitigación de este riesgo pasó por institucionalizar mecanismos de coordinación más frecuentes y formales. Se reforzó el uso de Slack para acuerdos rápidos y de Jira para centralizar la información del *backlog* y el avance de tareas. Estas medidas, sumadas al regreso de los integrantes en Febrero, hicieron que el riesgo comenzará a disminuir hasta estabilizarse en niveles bajos hacia mediados de año.

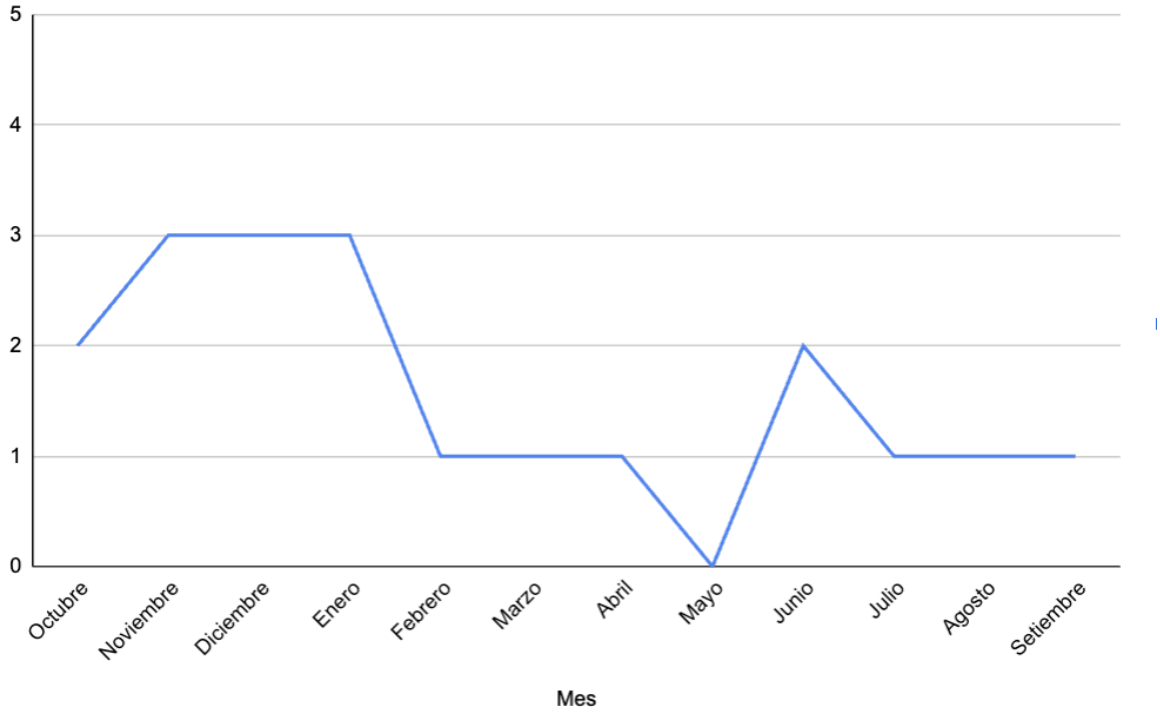


Ilustración 33 - Gráfica de monitoreo de RK1

RK2 – Cambios en requerimientos

Desde las primeras etapas del proyecto, el riesgo de cambios en los requerimientos fue identificado como uno de los factores más relevantes a monitorear. Esto se debía a que el alcance del sistema podía modificarse a medida que avanzara el conocimiento del dominio y surgieran nuevas necesidades del cliente o del mercado. Por ello, este riesgo se mantuvo activo a lo largo de todo el ciclo de vida del proyecto y fue evaluado periódicamente durante las reuniones de seguimiento.

Los cambios en los requerimientos fueron un riesgo que acompañó al proyecto a lo largo de todo su ciclo de vida. Se manifestó con mayor fuerza en Noviembre de 2024, cuando durante la fase de descubrimiento aparecieron dudas sobre el alcance inicial del sistema, y volvió a intensificarse en marzo de 2025, ya en plena fase de desarrollo. La decisión de incluir el **sistema de créditos para tiendas** en el *release* 2 fue de gran impacto, ya que fue una funcionalidad no contemplada al inicio, pero que surgió como crítica para aumentar el valor del producto frente a los comercios.

Posteriormente, en Julio de 2025, el riesgo volvió a aumentar debido a la necesidad de realizar la **migración a una nueva librería de UI**. Esta decisión técnica no estaba prevista en el plan original, pero se consideró imprescindible para mejorar la experiencia de usuario y asegurar la escalabilidad del sistema. El impacto fue notable, ya que la migración obligó a rehacer varios componentes de interfaz y ajustar tareas en curso, lo que generó replanificaciones en los últimos *sprints* antes del *release* 4.

El impacto de este riesgo en general fue bien gestionado: tanto la incorporación de cambios como la mitigación del impacto se abordaron al reorganizar el *backlog* y al postergar tareas de menor relevancia, como algunas mejoras cosméticas o

funcionalidades no prioritarias. La mitigación consistió en apoyarse en la flexibilidad del marco ágil: los *sprints* se organizaron con objetivos alcanzables, dejando un margen para absorber cambios, y los *releases* se concibieron con metas estratégicas que podían redefinirse según las prioridades emergentes.

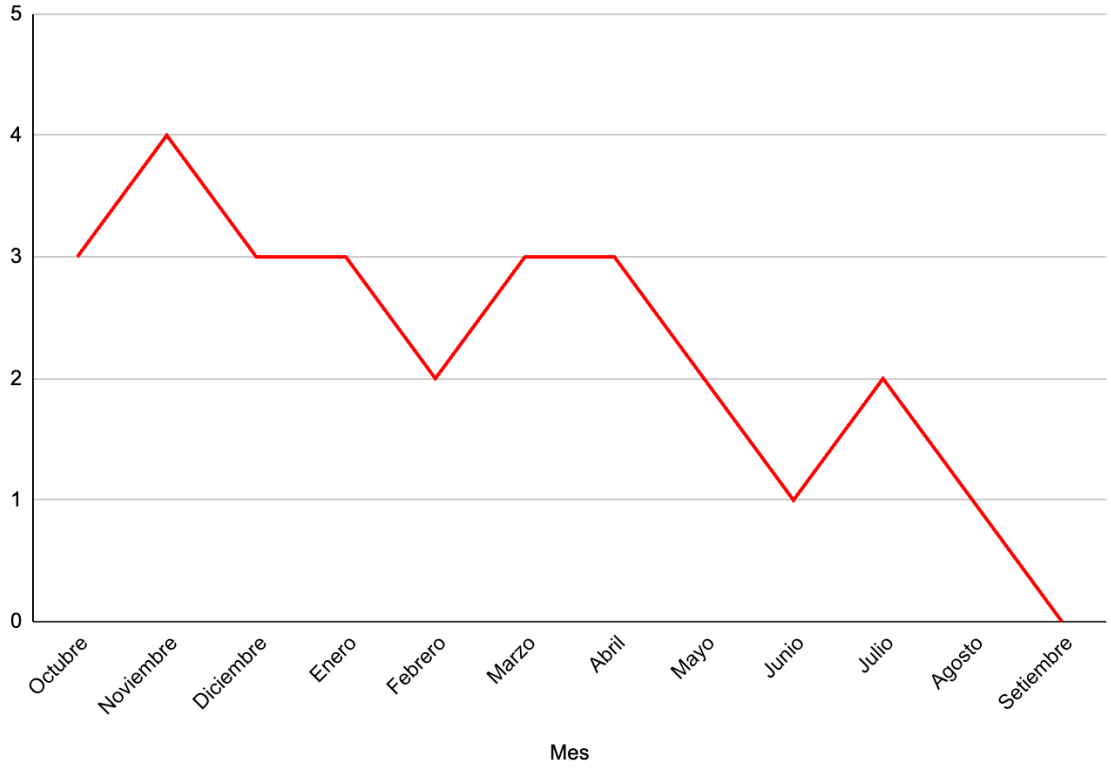


Ilustración 34 - Gráfica de monitoreo de RK2

RK3 – Productividad del equipo

La productividad del equipo fue un riesgo que se vio de manera clara en dos momentos del proyecto. El primero fue en Enero de 2025, cuando la ausencia de dos integrantes de viaje redujo la capacidad operativa y provocó que varias tareas quedaran pendientes. En la gráfica, este mes coincide con el pico más alto de magnitud del riesgo. El segundo momento fue en Junio de 2025, cuando el equipo enfrentó una alta carga académica en la facultad. La coincidencia de parciales y entregas universitarias impactó directamente en el tiempo disponible para el proyecto, reduciendo el ritmo de avance y obligando a reprogramar algunos compromisos.

La mitigación se apoyó en dos líneas de acción. En primer lugar, se ajustaron los criterios de estimación de *story points*, haciéndolos más realistas en función de la disponibilidad real de horas. En segundo lugar, se redistribuyeron las tareas de forma más equitativa, para evitar que la menor disponibilidad de uno o dos integrantes afectara el progreso total. A partir de estas medidas, la productividad logró estabilizarse.

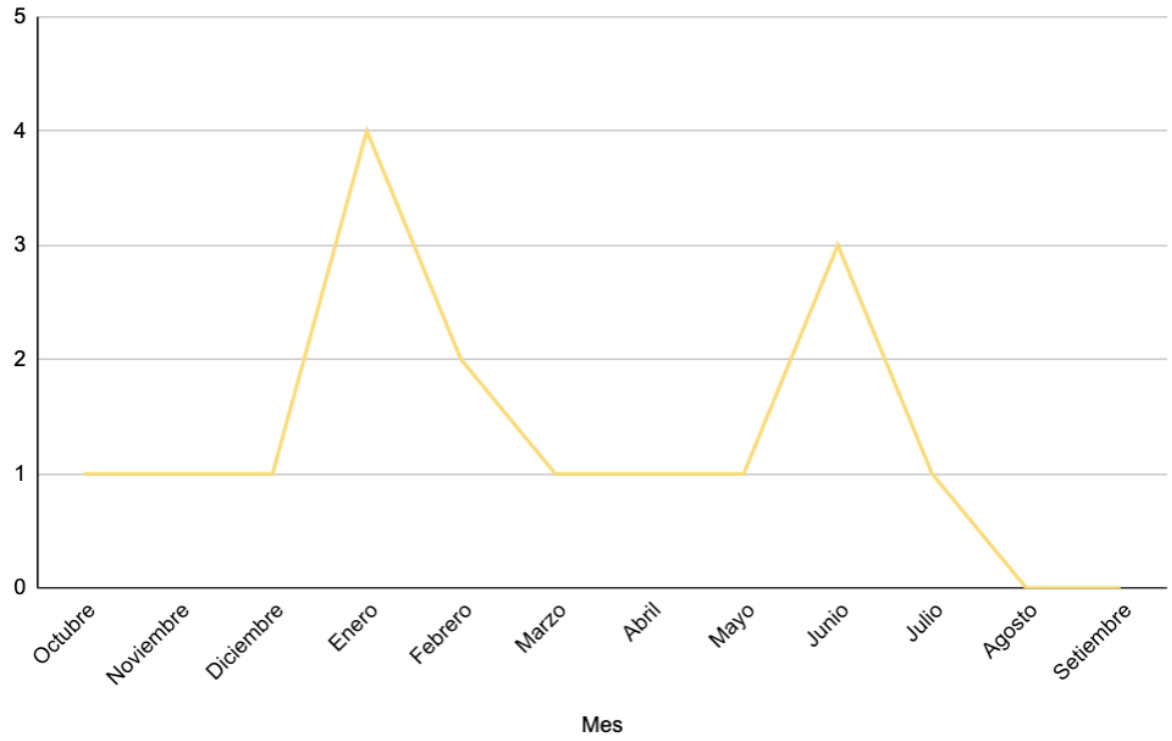


Ilustración 35 - Gráfica de monitoreo de RK3

RK7 – Comunicación con el cliente

La relación con los *stakeholders* y la validación de avances constituyeron otro riesgo importante. Durante los primeros meses, el riesgo se mantuvo bajo porque la interacción era constante y las decisiones se validaban de manera informal. Sin embargo, en Marzo de 2025 alcanzó un nivel crítico debido a la demora en instancias de *feedback*, lo que puso en riesgo la validación temprana de funcionalidades. Esta situación generó incertidumbre en el equipo, que necesitaba asegurarse de que el sistema respondía a las expectativas del cliente.

La mitigación se centró en calendarizar con mayor rigor las *demos* al cierre de cada *release*, asegurando que siempre hubiera instancias de validación formal. Además, se documentaron los avances en Jira y se compartieron capturas y reportes de progreso, lo que permitió mantener la transparencia incluso en los períodos donde las reuniones con el cliente y todo el equipo no podían concretarse con regularidad. El vínculo laboral de Federico con Fitit, permitió que el riesgo no sea lo suficientemente alto, debido a la actualización informal que se fue dando en momentos de menor comunicación grupal. Estas medidas lograron reducir la magnitud del riesgo en los meses siguientes, manteniéndolo en niveles medios-bajos hasta el cierre del proyecto.

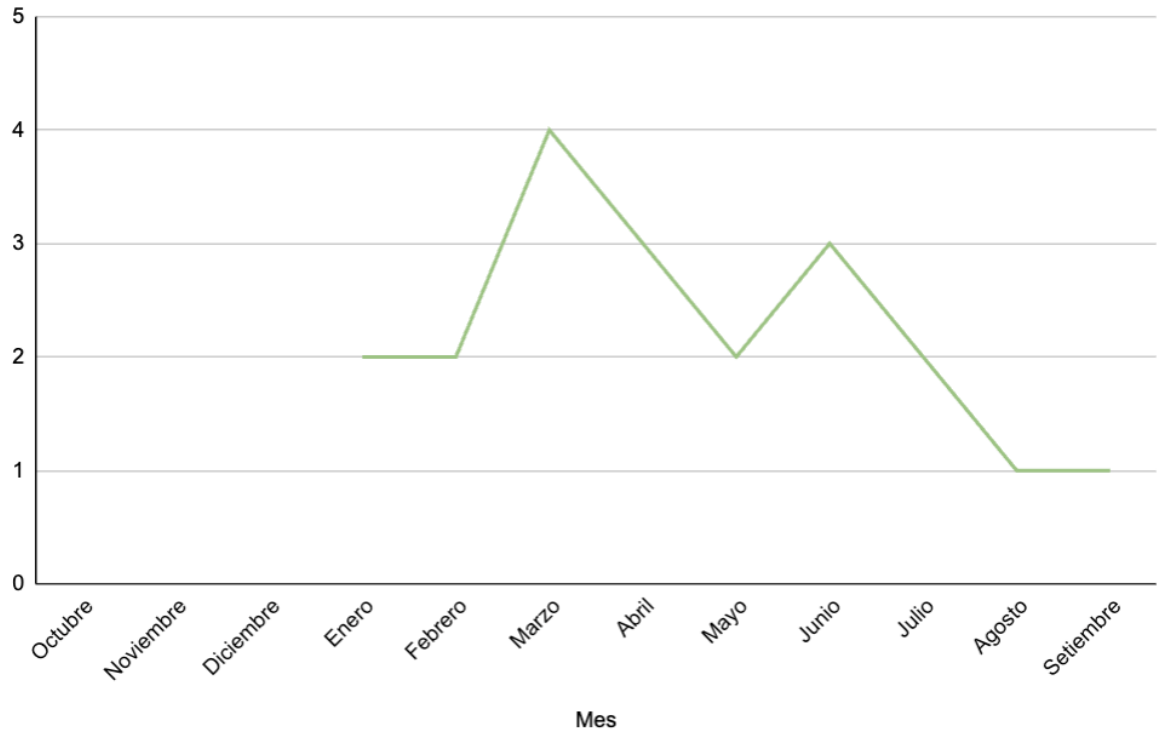


Ilustración 36 - Gráfica de monitoreo de RK7

La gestión de riesgos fue un proceso continuo durante todo el proyecto. Algunos riesgos, como los cambios en requerimientos y la productividad del equipo, se manifestaron en distintos momentos y debieron ser tratados varias veces. Otros, como la complejidad técnica y las dependencias externas, no llegaron a generar problemas críticos gracias a las medidas preventivas.

En general, la adopción de prácticas ágiles (*sprints* cortos, *backlog* flexible, *demos* al cliente) fue fundamental para que los riesgos pudieran ser absorbidos sin comprometer los plazos ni la calidad final del sistema. El aprendizaje principal fue que la anticipación y la transparencia en la comunicación son las mejores herramientas para mantener los riesgos en niveles controlados.

6. Ingeniería de requerimientos

"El éxito o fracaso de un sistema de software depende, en gran medida, de la precisión y la estabilidad de sus requerimientos." - Ian Sommerville, *Software Engineering* [20].

La ingeniería de requerimientos constituye una de las etapas más críticas del ciclo de vida del desarrollo de *software*, ya que define **qué debe hacer el sistema y bajo qué condiciones debe operar**. Su propósito es asegurar una comprensión compartida entre los usuarios, *stakeholders* y el equipo técnico, reduciendo ambigüedades y minimizando el riesgo de fallos en etapas posteriores.

De acuerdo con el *IEEE Standard 29148-2018* [21], esta disciplina abarca actividades sistemáticas de **elicitación, análisis, especificación, validación y gestión** de requerimientos, permitiendo transformar necesidades del negocio en soluciones verificables y medibles.

En el contexto de Fitit Returns, la ingeniería de requerimientos se centró en comprender las necesidades del negocio –que involucran al cliente, las tiendas, los usuarios finales y las cadeterías– junto con el entorno técnico existente, con el fin de definir con claridad el alcance, las capacidades funcionales, los atributos de calidad y las restricciones del nuevo sistema de gestión de devoluciones.

Este proceso se abordó siguiendo un ciclo de vida estructurado que incluyó:

- **Relevamiento** de información, orientado a identificar problemas, expectativas y oportunidades.
- **Análisis**, donde se depuraron, priorizaron y clasificaron los requerimientos, evaluando su consistencia y alineación con los objetivos del negocio.
- **Especificación**, mediante la formalización de los requerimientos funcionales y no funcionales en un formato claro y verificable.
- **Validación**, a través de revisiones con los distintos actores para garantizar que los requerimientos reflejen de forma precisa las necesidades planteadas.
- **Gestión** del cambio, asegurando trazabilidad y actualización de los requerimientos a lo largo del ciclo de desarrollo.

6.1. *Design Thinking*

Design Thinking es un enfoque metodológico **centrado en las personas** que busca generar soluciones innovadoras y alineadas con **necesidades reales**. A diferencia de los modelos tradicionales de recolección de requerimientos, incorpora actividades iterativas de **empatizar**, **definir**, **idear**, **prototipar** y **probar**, que enriquecen cada etapa del ciclo de vida de la ingeniería de requerimientos. Este enfoque permite garantizar que las soluciones sean deseables para los usuarios, viables desde el punto de vista del negocio y factibles técnicamente [22].

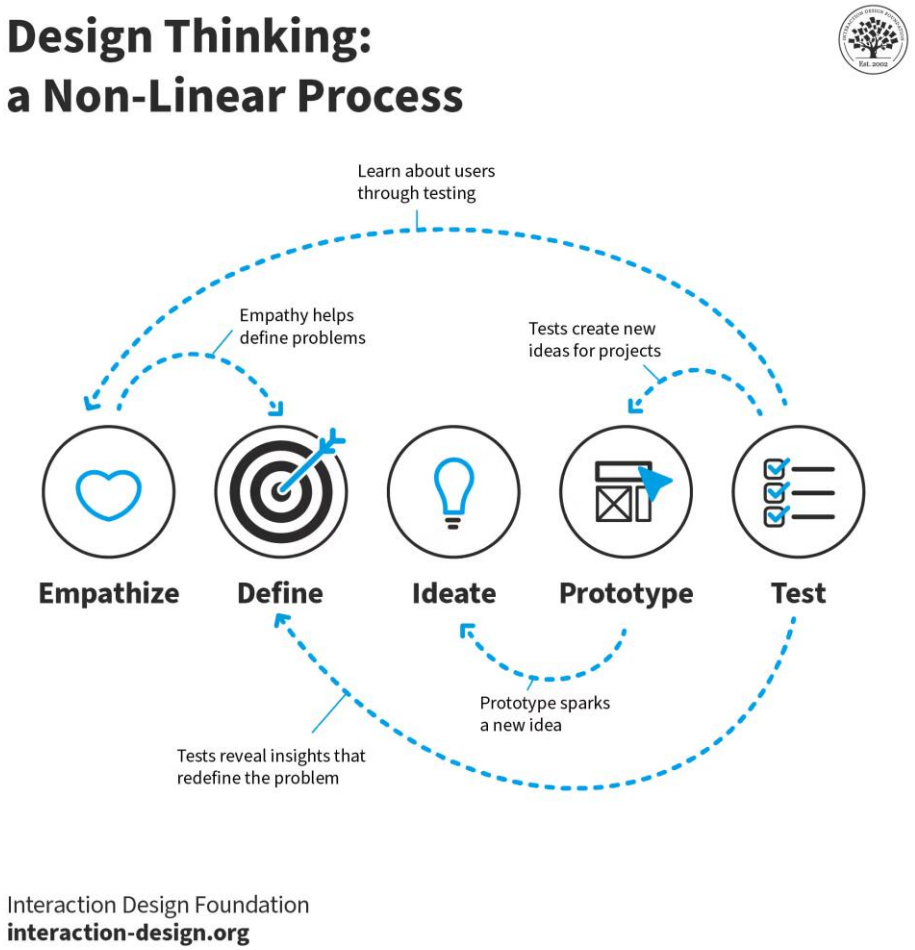


Ilustración 37 - Etapas de *Design Thinking* como proceso iterativo [23]

En este proyecto, *Design Thinking* se adoptó como marco de apoyo al ciclo de vida de la ingeniería de requerimientos. De esta manera, su aplicación sirvió para comprender de manera profunda las necesidades de Fitit, los usuarios, las tiendas y las cadeterías, y transformarlas en requerimientos claros y priorizados. Este enfoque iterativo permitió definir funcionalidades concretas y valiosas, asegurando que la solución fuera relevante, efectiva y técnicamente viable para los *e-commerce* de indumentaria.

6.2. Relevamiento

El **relevamiento** es la fase del ciclo de vida de la ingeniería de requerimientos destinada a identificar y recopilar de manera sistemática las necesidades, expectativas y restricciones de los distintos actores involucrados en el sistema. Su objetivo es generar una base sólida de información que sirva de insumo para las siguientes fases.

Se realizaron encuestas a usuarios finales, así como también reuniones con dueños de tiendas y encargados de cadeterías. Esto permitió identificar problemas recurrentes como tiempos de procesamiento prolongados, falta de trazabilidad y claridad en la gestión de postventa, asegurando que las soluciones propuestas respondieran a necesidades reales y concretas.

6.2.1. Empatizar

El objetivo principal fue comprender en profundidad las necesidades, frustraciones y expectativas de todos los actores clave:

- Usuario
- Tienda
- Fitit
- Cadeterías

6.2.1.1. Encuestas a Usuarios

El primer paso fue realizar una **encuesta** a los usuarios finales (ver detalles en Anexo 12.9.1. Resultados de encuestas a usuarios), para poder entender mejor los problemas y frustraciones a la hora de devolver ropa en los *e-commerce*. Se obtuvieron **91 respuestas** y una gran cantidad de *insights* (hallazgos clave) como por ejemplo:

- "*Es una **pérdida de tiempo** enorme tener que ir presencialmente a una tienda a devolver o cambiar una prenda*"
- "*Me genera mucha **frustración** tener que ir a la tienda solo porque me quedó mal el talle de lo que compré*"
- "*Las tiendas **no dan mucha visibilidad** en sus sitios web sobre sus políticas de devolución*"

6.2.1.2. *Benchmarking* de procesos de devolución

Dentro de la fase de empatizar, se llevó a cabo un **análisis comparativo de los procesos de devolución existentes** en comercios electrónicos de referencia en Uruguay, con el objetivo de comprender las prácticas actuales, identificar barreras y detectar oportunidades de mejora para los usuarios finales.

Por ejemplo, se evaluó el procedimiento de devoluciones de **Renner**, donde se constató que el proceso se realiza únicamente por **correo electrónico** (ver Anexo 12.9.2. Email de devolución Renner), bajo ciertas condiciones estrictas: se requiere detallar dirección completa, número de pedido, número de documento de identidad y productos a devolver; además, solo se efectúa reembolso del valor pagado, sin envío automático de un producto de reemplazo, y se excluyen ciertos artículos como moda íntima, perfumes y cosméticos. Esta restricción evidenció cómo las **políticas de devolución** son un tema importante para las tiendas. Esta experiencia evidenció que, aunque el proceso es formalmente correcto, resulta engorroso, manual y poco flexible para el usuario.

En el caso de **Jack and Jones**, al intentar realizar un cambio o devolución, no se identificó ninguna opción clara en la web referente a estos procesos, por lo que se recurrió al **chatbot integrado en el *e-commerce***. Este chatbot ofrecía 10 opciones de ayuda, incluyendo una para consultas sobre devoluciones, pero al seleccionarla únicamente se proporcionaba un enlace a un documento extenso con los términos y condiciones para cambios y devoluciones, sin asistencia interactiva. De acuerdo con la política del comercio, los cambios de productos se pueden realizar en cualquier sucursal dentro de los 30 días siguientes a la recepción, mientras que las devoluciones deben gestionarse en un plazo de 5 días hábiles, conforme al Derecho de Arrepentimiento establecido en la legislación de consumo [24]. Los productos deben estar sin uso, con etiquetas y empaques originales; la ropa interior, calcetines y trajes de baño no pueden devolverse ni cambiarse. Dependiendo del tipo de devolución (error del cliente vs. falla del producto), los costos de envío corren por el cliente o por la tienda. Además, se requiere notificar al área de atención al cliente y adjuntar comprobantes de envío para seguimiento, mientras que el reembolso se realiza mediante Mercado Pago, con tiempos variables según el medio de pago.

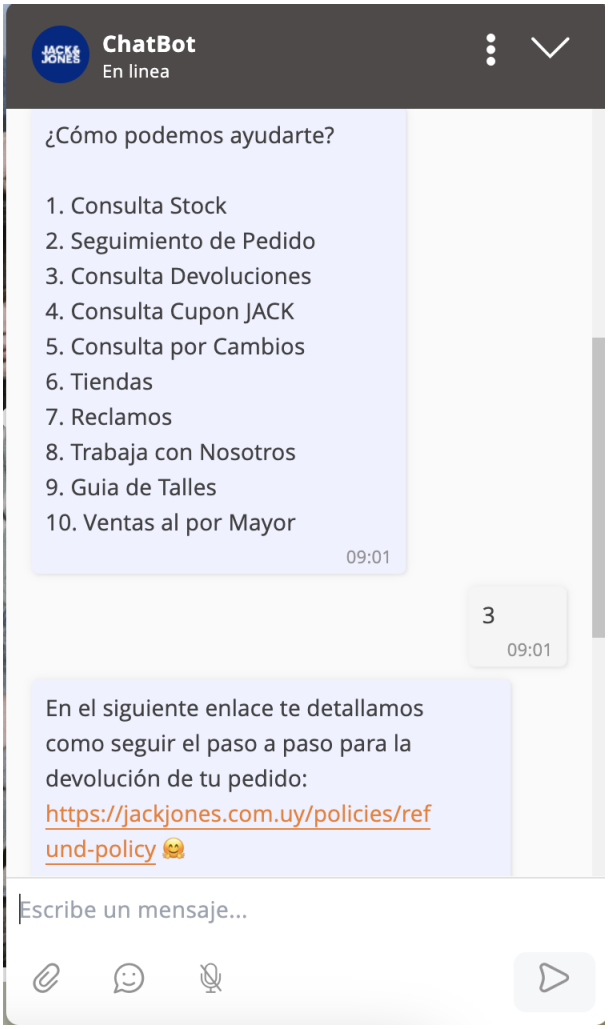


Ilustración 38 - Proceso de devolución en Jack&Jones Uruguay

6.2.1.3. Entrevistas

También se realizaron **entrevistas** a empleados de cadeterías y a tiendas, para poder entender con más profundidad el problema.

El resumen de los *insights* establece que el proceso de devolución en el *e-commerce* es hoy mayoritariamente manual (gestionado por email, WhatsApp o teléfono). Esto genera demoras, pérdida de información y una gran carga operativa para los empleados de las tiendas.

Desde el lado de los usuarios, las encuestas y entrevistas confirmaron tres hallazgos clave:

- **Frustración**: mala experiencia general, falta de seguimiento, poca claridad en las instrucciones y demoras excesivas.
- **Importancia del talle (Fitit)**: aproximadamente el 70% de las devoluciones se debe a problemas de talles, según varios reportes de la industria. [25] [26]

Entrevistas con Tiendas:

Se realizaron entrevistas con diferentes tiendas de Uruguay (Mundo Trabajo, Route 66, Basefield), con el objetivo de entender cómo gestionan actualmente sus devoluciones y qué oportunidades existen para mejorar la experiencia de postventa.

En el Anexo 12.9.3. Preguntas de entrevista para tiendas, se encuentran las preguntas realizadas de forma estructurada.

Los puntos en común a destacar entre las **tiendas** se traducen en los siguientes problemas principales:

- **Pérdida de información y falta de tracking**: El trámite puede tardar varios días, no hay visibilidad sobre el estado del producto devuelto y se depende del cliente para obtener actualizaciones.
- **Impacto operacional**: La planificación de inventarios y la reposición de stock se vuelven ineficientes. Así como también se consume mucho tiempo del personal en gestionar reclamos manualmente.
- **Retención de valor:** La mayoría de las devoluciones significan una pérdida de valor. Las tiendas prefieren potenciar los cambios por sobre las devoluciones, incentivando la recompra mediante créditos de tienda o selección inmediata de nuevos artículos.
- **Importancia del registro de motivos:** Ninguna tienda lleva métricas sobre causas de devolución. Esto limita la capacidad de detectar patrones y tomar decisiones sobre talles, descripciones de productos o calidad.
- **Integración logística:** La diversidad de cadeterías utilizadas (DAC, Mirtrans, UES, etc.) muestra que una integración flexible con operadores logísticos es clave.
- **Los talles como origen del problema:** En varias entrevistas (ej. Route 66), los talles mal elegidos aparecen como el principal motivo de devolución.
- **Canales de comunicación dispersos:** WhatsApp, email y llamadas siguen siendo los principales medios de gestión, lo que refleja la necesidad de unificar y profesionalizar el canal de postventa.

Entrevista con UES:

Otra de las actividades realizadas en esta fase fue una **entrevista con Lucía Benítez**, ejecutiva comercial y encargada de marketing de **UES**, empresa uruguaya de servicios logísticos especializada en soluciones de envíos y distribución para *e-commerce*. El objetivo fue comprender de primera mano cómo operan los procesos de logística inversa y cuáles son las limitaciones actuales en el manejo de devoluciones.

Durante la entrevista se discutieron distintos aspectos relevantes:

- **Modalidades de integración:** UES puede integrarse con los comercios a través de *API*, plugins o de forma manual (esta última menos utilizada).
- **Gestión de alto volumen:** En clientes con gran caudal de envíos se establecen horarios fijos de retiro, lo que obliga a coordinar con anticipación mediante el sistema.
- **Opciones de despacho:** Además del retiro programado, los clientes pueden utilizar puntos de **pick-up cercanos** para despachar productos.
- **Notificaciones de envíos:** Al estar UES integrado con las plataformas web de los comercios, recibe automáticamente las notificaciones de envíos, sin diferenciar si corresponden a una compra o a una devolución.
- **Etiquetas de devolución:** En algunas tiendas, UES envía junto con el pedido dos etiquetas: una adherida al paquete original y otra adicional para que el cliente pueda usarla en caso de devolución.

Reunión con Keep Returns:

Otra actividad que surgió como parte de la fase de empatizar fue una reunión con la plataforma argentina **Keep Returns**, especializada en la gestión de la postventa. El encuentro se generó a partir de un interés de colaboración, motivado por el **alto volumen de devoluciones en *e-commerce* derivadas de errores de talle de prendas**, una problemática directamente alineada con los objetivos de Fitit.

Durante la reunión, se relevó el funcionamiento de Keep Returns y su propuesta de valor: la plataforma busca **conectar lo online con lo presencial**, gestionando no solo devoluciones digitales, sino también procesos físicos en tiendas. Su flujo estándar contempla cinco pasos simples (ingreso de datos de la compra, selección del artículo, elección entre cambio o devolución, definición del método de devolución y resumen final), y se integra únicamente a los sitios desarrollados en ***VTEX*** [27] mediante plugins. Asimismo, la solución incentiva los **cambios por sobre las devoluciones**, ofreciendo descuentos para quienes eligen reemplazar un producto en lugar de solicitar reembolso. Otros hallazgos importantes a destacar son:

- La integración con medios de pago para gestionar devoluciones y reembolsos sigue siendo una de las **mayores dificultades técnicas**, lo que representa un desafío para cualquier plataforma de devoluciones.
- La **integración con sistemas de logística y cadetería** resulta esencial para ofrecer una experiencia fluida de postventa.

6.2.1.4. User Journey Map

Para profundizar en la comprensión de la experiencia de los usuarios que realizan devoluciones, se elaboró un **User Journey Map** del proceso actual de devoluciones, centrado exclusivamente en el usuario final. El objetivo es visualizar de manera

estructurada todas las etapas, acciones, emociones y puntos de dolor que se atraviesan al devolver o cambiar un producto.

En cada etapa se documentaron acciones del usuario, emociones y frustraciones, permitiendo identificar oportunidades de mejora y sirviendo como base para la definición de requerimientos futuros que apunten a simplificar y mejorar la experiencia de devolución.

Ilustración 39 - Evidencia del User Journey Map parcial

Las ilustraciones completas del User Journey Map se encuentran en el Anexo 12.9.4. User journey map del proceso actual de devolución, mostrando visualmente las etapas y puntos de dolor del usuario.

6.3. Análisis

El **análisis** consiste en procesar, organizar y priorizar la información obtenida durante el relevamiento. Implica identificar conflictos, dependencias y requerimientos críticos, con el objetivo de establecer una visión coherente de las necesidades del sistema y preparar la documentación de especificación.

En Fitit Returns, se utilizó la fase de **Definir** de *Design Thinking* para transformar los hallazgos del relevamiento en problemas claros y oportunidades de mejora. Se categorizaron los requerimientos por tipo (funcionales y no funcionales), se priorizaron según impacto y factibilidad, y se establecieron criterios de aceptación para futuras funcionalidades.

6.3.1. Definir

En la fase de **Definir** dentro del marco de *Design Thinking*, el objetivo es transformar la información obtenida en la fase de **Empatizar** en una declaración clara del problema, identificando necesidades reales de los usuarios y oportunidades de mejora para orientar el desarrollo de la solución.

Para lograrlo, se aplicó un proceso de **ingeniería inversa** sobre diferentes plataformas de referencia en el mercado (Reversso, Returnly, Keep Returns). Este análisis permitió identificar tanto buenas prácticas a conservar como debilidades a superar, sirviendo como insumo para definir los **requerimientos diferenciadores de Fitit Returns**.

Puntos fuertes identificados en soluciones existentes:

- Flujos guiados y simples para el usuario final.
- Incentivo a priorizar cambios por sobre devoluciones, fomentando la recompra.
- Integraciones listas con algunas plataformas de *e-commerce* y sistemas de logística.

Puntos débiles detectados:

- **Limitaciones tecnológicas:** la mayoría de las soluciones solo funcionan en plataformas específicas (VTEX, Shopify, TiendaNube).
- **Déficit en métricas y tracking:** la visibilidad del proceso y el análisis de motivos de devolución son incompletos.
- **Alta incidencia de devoluciones por talle:** sin un abordaje predictivo o preventivo.
- **Comunicación con el usuario poco clara o fragmentada:** Las soluciones actuales suelen tener procesos de contacto limitados o inconsistentes, lo que genera confusión en el cliente.

Posteriormente, durante la fase de **Definir**, se buscó sintetizar y priorizar la información obtenida.

Para ello se llevaron a cabo las siguientes actividades:

- **Saturar:** Se revisaron y consolidaron todos los datos recolectados para tener una visión completa de los problemas y oportunidades. Ver Anexo 12.9.5. Saturar.
- **Agrupar:** Se organizaron los hallazgos por similitudes y patrones, por ejemplo separando problemas relacionados con procesos internos, experiencia del usuario y logística de cadeterías. Ver Anexo 12.9.6. Agrupar.
- **Identificar perfiles:** Se detectaron los distintos tipos de usuarios y actores involucrados (usuarios finales, tiendas, cadeterías) para entender sus

necesidades específicas. Esto permitió definir grupos homogéneos con comportamientos y expectativas similares. Se extrajeron las necesidades clave de cada perfil, como simplificación de procesos, visibilidad de devoluciones, o flexibilidad en las políticas.

Perfiles

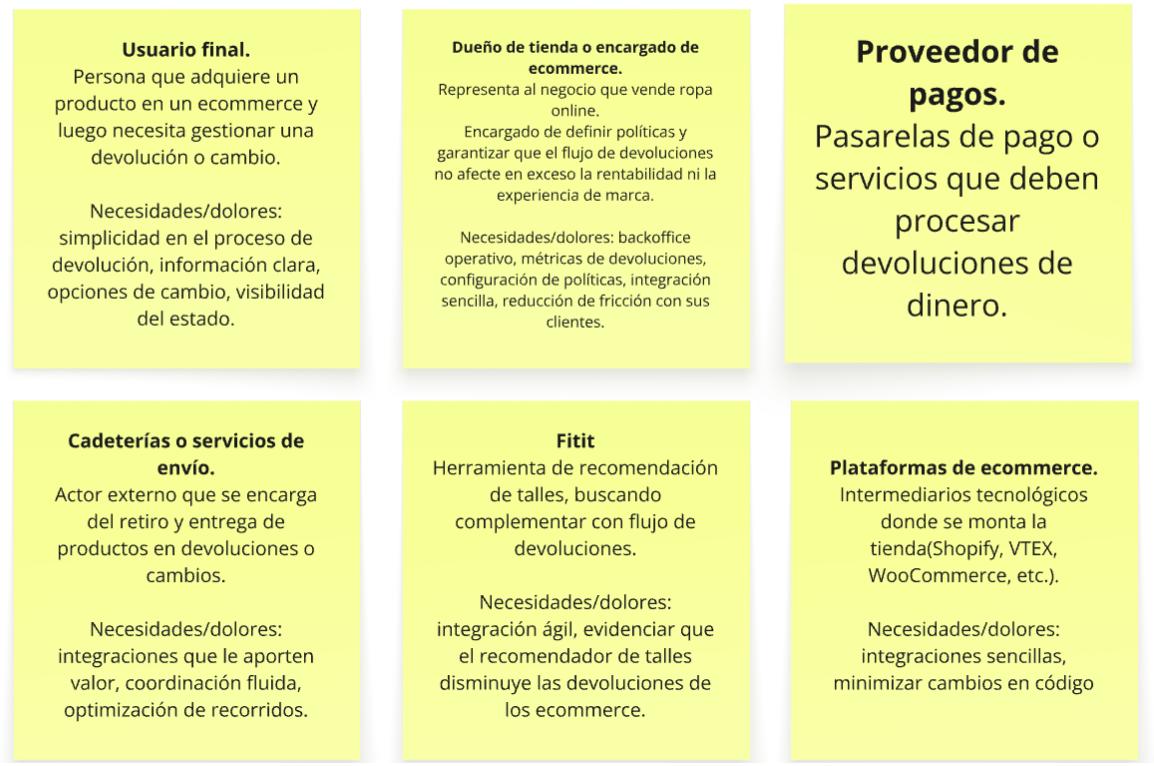


Ilustración 40 - Evidencia de identificación de perfiles

- **User Personas:** Se desarrollaron *user personas* para representar a los actores clave del proceso de devolución. Estas representaciones, basadas en la información recopilada en la fase de **Empatizar** (entrevistas, encuestas y observación de procesos), permitieron identificar patrones de necesidades, frustraciones y comportamientos de los usuarios finales y de los encargados de *e-commerce*, sirviendo como evidencia para sintetizar hallazgos, priorizar requerimientos y fundamentar decisiones de diseño posteriores. Ver Anexo 12.9.7. User personas.

- **Declaración del problema:** El análisis integral de las actividades anteriores permitió obtener una comprensión profunda de la situación actual y formular declaraciones de problema que orientaron la definición de los requerimientos del sistema **Fitit Returns**.

Estas actividades permitieron generar una visión clara de los problemas principales y formular la declaración del problema que guiaron el desarrollo de requerimientos diferenciables para Fitit Returns.

Declaración del problema principal:

“La gestión de devoluciones propia de los e-commerce es manual e ineficiente, lo que genera una experiencia de usuario deficiente, caracterizada por la falta de tracking, demoras y comunicación poco clara.

Por otro lado, las tiendas enfrentan pérdidas operacionales significativas, incluyendo sobrecarga operativa, falta de métricas y desajustes de stock.

Además, la alta incidencia de devoluciones asociadas a problemas de talle (aproximadamente el 70 % según estudios del sector) evidencia la necesidad de una solución integral que automatice el proceso de devolución y se integre con herramientas de ayuda como el recomendador de talles de Fitit, permitiendo así reducir la tasa de devoluciones y optimizar la experiencia del cliente.”

6.4. Especificación

La etapa de **especificación** dentro del ciclo de vida de la ingeniería de requerimientos documenta con detalle los requerimientos funcionales y no funcionales del sistema, estableciendo cómo debe comportarse y qué restricciones debe cumplir. Esta fase asegura que el equipo de desarrollo tenga criterios claros y verificables para implementar la solución.

En el contexto de Fitit Returns, esta etapa tomó como base las conclusiones derivadas de la fase de **Idear** del enfoque *Design Thinking*, donde se generaron y priorizaron soluciones que respondían directamente a los problemas detectados durante el proceso de devolución en los *e-commerce* de las tiendas asociadas.

De este modo, la especificación sirvió como **puente entre el pensamiento creativo y el desarrollo técnico**, estructurando la solución a partir de épicas, historias de usuario y criterios de aceptación claros y medibles.

6.4.1. Idear

La fase de **Idear** en *Design Thinking* se centra en generar un amplio conjunto de soluciones posibles a los problemas previamente definidos, fomentando la creatividad y el pensamiento divergente antes de priorizar las ideas más viables. El objetivo principal es transformar la comprensión profunda de los usuarios obtenida en etapas anteriores (**Empatizar** y **Definir**) en **conceptos concretos y accionables**, que luego se traducen en requerimientos formales durante la especificación

En este proyecto, para la fase de ideación se realizaron las siguientes actividades:

- **Sesiones de brainstorming** con el equipo y con Fitit como *stakeholder* principal, utilizando la herramienta *Miro* para capturar y organizar visualmente las ideas.

- **Priorización de ideas** mediante votación, evaluando impacto, valor para el usuario y viabilidad técnica. Ver Anexo 12.9.8. Votación para priorizar ideas.

- **Definición de flujos de devolución y cambio**, mediante diagramas que representaron las interacciones principales del sistema.

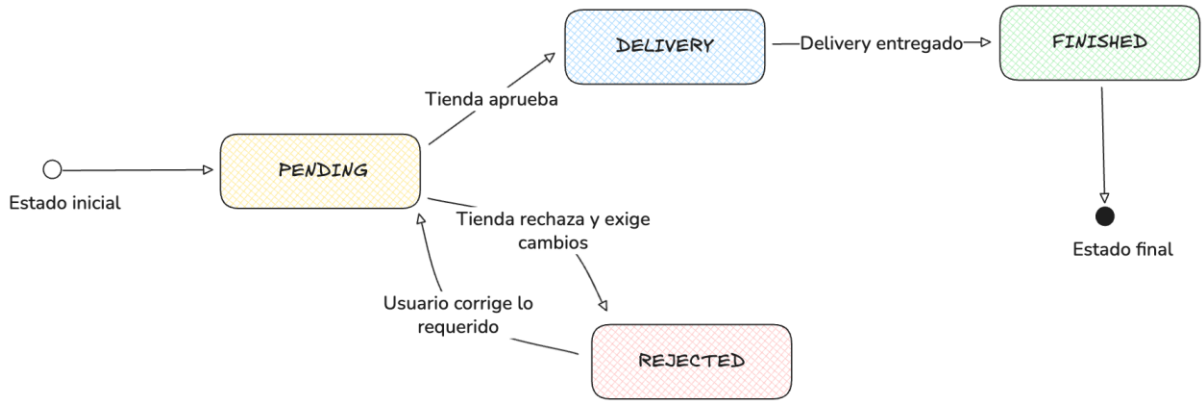


Ilustración 41 - Diagrama de estados de una devolución y cambio

- **Creación de épicas e historias de usuario**, asegurando la alineación entre las necesidades de los distintos actores (usuarios finales, tiendas, y Fitit) y los requerimientos técnicos del producto.

6.4.1.1. Priorización y justificación de funcionalidades relevantes

En esta etapa se identificaron y priorizaron las funcionalidades clave del sistema **Fitit Returns**, basándose en su impacto en la experiencia del usuario, la eficiencia operativa de las tiendas y la viabilidad técnica de implementación. Además, se documentan las principales decisiones de diseño y justificaciones que influyeron en la definición del alcance funcional del *MVP*.

Usuario final

- **Simplificación del proceso de devolución e integración con Fitit para recomendar cambios de talle o modelo.**

  Esta funcionalidad se priorizó por su alineación directa con el valor principal de Fitit: reducir devoluciones por error de talle. Permite ofrecer al usuario opciones de cambio sugeridas en lugar de devoluciones completas, mejorando la retención y reduciendo costos logísticos.

- **Experiencia 100 % online, con opciones de pickup o retiro en domicilio.**

  Se buscó minimizar la fricción del proceso de devolución, garantizando comodidad y flexibilidad para el usuario, elementos críticos detectados en la etapa de Empatizar.

- **Visibilidad del estado de la devolución mediante notificaciones automáticas.**

  Se priorizó para aumentar la confianza del usuario y reducir el volumen de consultas manuales al servicio de atención de las tiendas.

Tiendas

- **Dashboard de métricas sobre motivos de devolución y desempeño del recomendador de talles.**

  Su inclusión responde a la necesidad de las tiendas de disponer de indicadores claros que cuantifiquen el impacto del sistema y permitan tomar decisiones basadas en datos.

- ***Backoffice* operativo para revisar, aprobar o rechazar devoluciones.**

  Se definió como funcionalidad esencial para otorgar autonomía a las tiendas y centralizar la gestión del proceso postventa.

- **Configuración de políticas de devolución diferenciadas.**

  Permite flexibilidad en la operación, adaptando las políticas según tipo de producto o condiciones comerciales.

Cadeterías

- **Integración con servicios logísticos y de *delivery*.**

  Priorizada para optimizar la recolección y entrega de productos, reduciendo la carga operativa sobre las tiendas y garantizando trazabilidad.

E-commerce

- **Integración con plataformas de *e-commerce* para acceder a detalles de órdenes y productos.**

  Esta integración permite validar automáticamente los datos de la compra, evitando errores y simplificando la experiencia del usuario.

En cuanto a la **forma de integrar la solución en los *e-commerce***, se evaluaron varias alternativas:

1. **Script embebido:** es la forma que actualmente tiene Fitit de integrarse en los *e-commerce*. Requiere modificar el código fuente del *e-commerce* para integrar la experiencia directamente en la web.

2. **Linking independiente:** un sistema autónomo que se conecta mediante parámetros en la URL (por ejemplo, *storeId* y *orderId*), permitiendo acceso al flujo de devolución sin modificar la tienda.

3. ***SDK* o Plugin:** integraciones empaquetadas para mayor escalabilidad y facilidad de adopción.

Luego de evaluar ventajas y desventajas de cada opción, el equipo se decidió por la **opción 2 (linking independiente)**.

Esta elección no solo mejora la experiencia de la tienda, que únicamente necesita configurar una URL con dos parámetros de consulta, sino que también facilita la puerta de entrada a Fitit Returns, incluso bajo el envío de notificaciones vía correo electrónico o teléfono, permitiendo que el usuario inicie el flujo de devolución o cambio con un simple link.

Para Fitit, esta decisión generó cierta polémica, ya que su enfoque actual busca que el usuario permanezca dentro del *e-commerce* sin salir de la página. Sin embargo, considerando que se trataba de un *MVP* —una versión mínima y funcional del sistema destinada a validar hipótesis y recopilar retroalimentación temprana— se aprobó la opción debido a sus ventajas de facilidad y flexibilidad para desarrollar, implementar y probar rápidamente. El resto de la justificación se encuentra en el Anexo 12.4. Funcionalidades por release y descartadas.

6.4.1.2. Épicas

Las **épicas** representan grandes unidades funcionales que agrupan un conjunto de requerimientos relacionados bajo un mismo objetivo estratégico. Según la **Agile Alliance (2022)**, una épica es *"una colección coherente de historias de usuario que describen un flujo o funcionalidad de alto nivel, orientada a entregar valor significativo al negocio o al usuario"* [28].

En Fitit Returns, las épicas se definieron como el primer nivel de abstracción dentro de la especificación, conectando la visión funcional del sistema con las necesidades identificadas en el análisis. Cada épica se descompuso posteriormente en historias de usuario y criterios de aceptación verificables, permitiendo mantener la trazabilidad entre el requerimiento inicial y su implementación técnica.

Algunos de los temas principales de las épicas incluyen:

- Gestión de devoluciones y cambios.
- Panel de control para tiendas.
- Configuración de políticas de devolución.
- Integración con plataformas de *e-commerce*.
- Métricas y reportes operativos.
- Gestión de usuarios y tiendas.
- Notificaciones y seguimiento de estado.

Las épicas completas se encuentran documentadas en el Anexo 12.10. Épicas.

6.4.1.3. Historias de usuario

A partir de las épicas, se elaboraron **historias de usuario** siguiendo las buenas prácticas del enfoque **ágil** descrito por *Mike Cohn* en *"User Stories Applied: For Agile Software Development"*, referente en la gestión de requerimientos ágiles [29].

Las historias se formularon utilizando el formato **INVEST** propuesto por *Bill Wake* [30], asegurando que cada historia fuera:

- **I**ndependiente: no depender de otras.
- **N**egociable: abierta a revisión.
- **V**aliosa: aporta valor al usuario.
- **E**stimable: puede medirse en esfuerzo.
- **S**mall: de tamaño manejable.
- **T**estable: con criterios de aceptación verificables.

El resto de los detalles de historias de usuario y épicas se detallan en el capítulo 5. Gestión del proyecto.

6.5. Validación

La **validación** consiste en comprobar que los requerimientos especificados reflejan fielmente las necesidades del negocio y de los usuarios, y que son comprensibles y verificables. Incluye revisiones, prototipos y pruebas de concepto para asegurar que los requerimientos sean completos, consistentes y correctos.

En Fitit Returns, la fase de *Prototipar* y la de *Probar* pertenecientes a *Design Thinking* se aplicó mediante prototipos de la aplicación que se centra en el trámite de devolución o cambio del usuario, probados con usuarios finales, administradores de tienda y el equipo de Fitit. Esto permitió validar los casos de uso principales como el flujo de devolución y hallar mejoras antes de la implementación, asegurando que la solución fuera efectiva y fácil de usar.

6.5.1. Prototipar

Basados en los resultados de la fase anterior, el equipo procedió a prototipar en **baja fidelidad** para poder entender mejor los flujos que se iban a intentar construir más adelante. A continuación, y en el Anexo 12.9.9. Prototipo de baja fidelidad, Prototipo de baja fidelidad, se puede observar un ejemplo utilizando Escalidraw en donde se modeló la vista del usuario, quien puede ver sus devoluciones y estado e incluso entrar al detalle para ver más en profundidad y darle seguimiento a su solicitud:

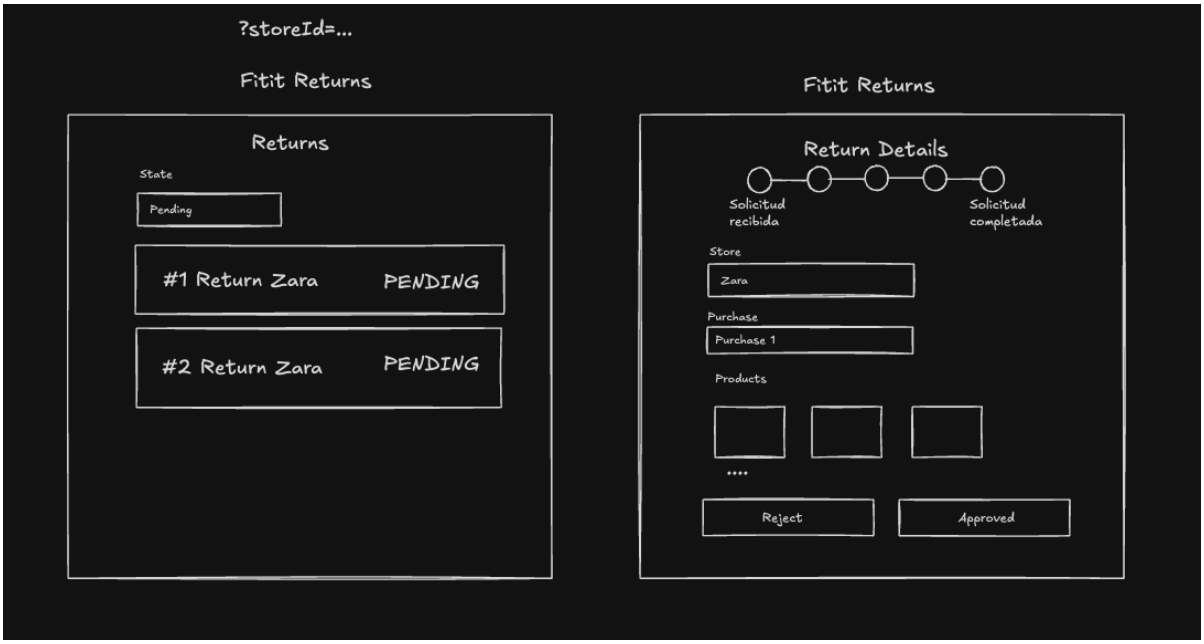


Ilustración 42 - Prototipos de baja fidelidad

También se utilizaron prototipos de **alta fidelidad,** que se pueden ver completos en el Anexo 12.9.10. Prototipo de alta fidelidad, para desarrollar una primera versión de la aplicación del lado del usuario. Para realizar estos prototipos se utilizó Figma, en la siguiente imagen se puede ver un ejemplo de lo que sería el inicio del flujo:

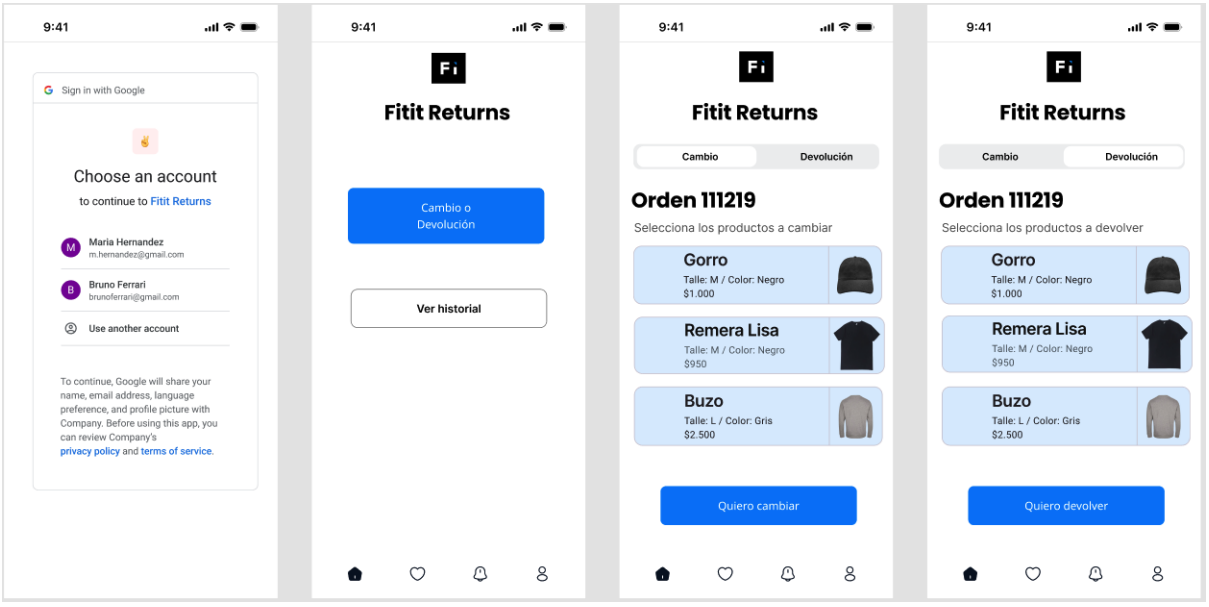


Ilustración 43 - Prototipos de alta fidelidad

6.5.2. Probar

En esta última fase iterativa, se realizaron sesiones de validación de prototipos con los actores principales de cada aplicación a construir. Se verificaron los hallazgos relevados con representantes de **usuarios, tiendas y Fitit**. El objetivo fue comprobar la viabilidad, usabilidad y deseabilidad de las propuestas antes de pasar al desarrollo.

Se llevaron a cabo pruebas de usabilidad en escenarios simulados de devolución y cambio, midiendo tiempos de ejecución, claridad del flujo y nivel de satisfacción percibida. A partir de estas validaciones surgieron aprendizajes clave:

- **Usuarios**: valoraron positivamente la posibilidad de realizar el proceso de devolución 100% online y destacaron como fundamental el tracking en tiempo real del estado de la solicitud. También remarcaron la importancia de recibir notificaciones claras.

- **Tiendas**: confirmaron la necesidad de un dashboard con métricas y de poder configurar políticas diferenciadas de devolución, validando así requerimientos priorizados. Además, se observó que la simplicidad en la integración por linking fue percibida como una ventaja competitiva.

- **Fitit**: validó que la solución propuesta se alineaba con su estrategia de producto y destacó la importancia de capturar datos estructurados sobre motivos de devolución para enriquecer el modelo de recomendación de talles.

Ilustración 44 - Validación de prototipos

El *feedback* obtenido se tradujo en ajustes a los requerimientos funcionales y no funcionales que van a ser detallados en las próximas secciones.

El proceso de *Design Thinking* aportó un marco estructurado y **centrado en las personas** que permitió transformar la comprensión profunda de las **necesidades de los usuarios** y actores involucrados en requerimientos claros, accionables y alineados con los objetivos del negocio. Se facilitó la identificación de problemas reales, la generación de soluciones innovadoras y la priorización de funcionalidades de alto impacto para el *MVP* de Fitit Returns. Este enfoque promovió la colaboración interdisciplinaria y el pensamiento iterativo, permitiendo validar hipótesis y ajustar el diseño de forma continua en función de la retroalimentación.

Tal como afirma **Larry Leifer** en *The Design Thinking Playbook*, *"Básicamente, hay un planteamiento del problema al principio y una solución al final, y la solución se alcanza mediante un procedimiento iterativo"* [22]. En el contexto de este proyecto,

esa naturaleza iterativa fue clave para construir una solución centrada en el usuario, técnicamente viable y con un camino claro de evolución futura.

6.6. Gestión del cambio

La **gestión del cambio** asegura que cualquier modificación en los requerimientos durante el desarrollo del proyecto se registre, evalúe y controle adecuadamente. Su objetivo es mantener la coherencia y trazabilidad de los requerimientos a lo largo del tiempo.

Los **requerimientos no son estáticos**: pueden surgir nuevas necesidades, cambios en el negocio o ajustes durante el desarrollo. La gestión del cambio establece procesos para registrar, evaluar, aprobar y comunicar modificaciones, garantizando que el sistema evolucione de manera controlada y alineada con los objetivos del proyecto.

En Fitit Returns, se aplicaron mecanismos formales de registro y control de versiones, que permitieron evaluar el impacto de cada modificación antes de su implementación. Cada nuevo requerimiento o ajuste fue documentado, analizado junto al equipo de desarrollo y validado con el cliente, asegurando que las decisiones mantuvieran la alineación con los objetivos del proyecto. Se evidencia el proceso de gestión del cambio en el Anexo 12.12. Proceso de gestión de cambios.

6.6.1. Nuevos requerimientos

Créditos de tienda:
Durante la fase de desarrollo surgió la necesidad de priorizar la implementación de los créditos de tienda por sobre otras funcionalidades del *backlog*. Esta decisión se basó en validaciones continuas realizadas con Fitit, las tiendas y el equipo, que permitieron identificar esta funcionalidad como un elemento clave para mejorar la experiencia del usuario. Los créditos de tienda representan una alternativa ágil e innovadora frente a la complejidad de integrar sistemas de pago en una etapa temprana del proyecto. Además, aportan valor tanto a las tiendas —que pueden fidelizar clientes y mantener el flujo económico dentro de su ecosistema— como a los usuarios, que conservan su poder de compra sin verse perjudicados durante el proceso de devolución o cambio.

Migración de librería de UI:
En los primeros *releases* se utilizó una UI básica que priorizó la funcionalidad sobre la estética. Sin embargo, en Julio se agregó el requerimiento a pedido del cliente de realizar la migración a una nueva librería, Radix UI, que se implementó en el *release* 4. Esta decisión implicó rehacer parte de la interfaz, pero permitió obtener un diseño más atractivo, consistente y fácil de mantener. Se puede ver en la Ilustración 45, la tabla de devoluciones del lado de la tienda previo a la migración, y el resultado final mostrado previamente en la sección 3.1.2. Vista de la tienda.

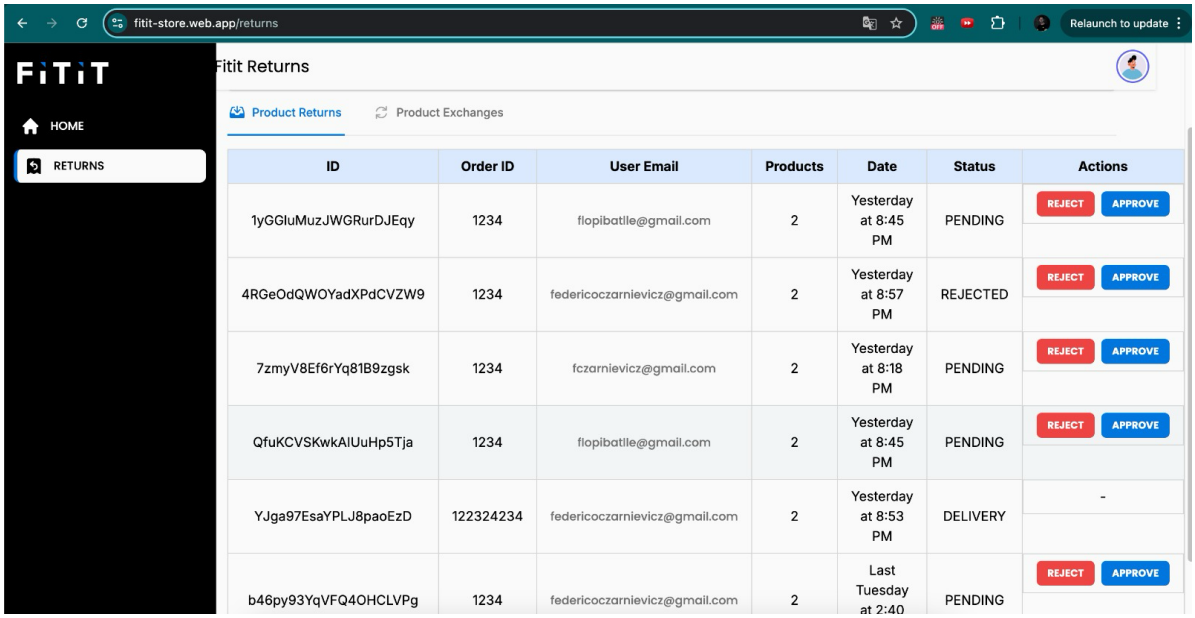


Ilustración 45 - Primera versión de tablero de devoluciones (Tienda)

Si bien supuso un esfuerzo adicional, este cambio fue clave para garantizar la usabilidad y la adopción del sistema por parte de los usuarios. La evolución de las pantallas principales se pueden ver en el Anexo 12.5. Evolución en la UI de pantallas principales.

Ambos casos reflejan cómo la gestión del cambio permitió incorporar nuevas necesidades sin comprometer la estabilidad del proyecto, manteniendo trazabilidad, control y coherencia con los objetivos definidos.

6.7. Requerimientos funcionales

Los requerimientos funcionales definen las capacidades específicas que el sistema debe ejecutar para cumplir con los objetivos del negocio:

6.7.1. Usuario

Requerimiento Funcional	**Descripción**
RFU1: Flujo de devolución	Permitir al usuario iniciar una devolución de producto.
RFU2: Flujo de cambio	Ofrecer al usuario la opción de cambio por otra talle o modelo, incentivando la retención y recompra.
RFU3: Visualización del histórico de devoluciones y sus detalles	Permitir al usuario consultar todas las devoluciones realizadas, junto con el estado actual y los detalles de cada caso.
RFU4: Autenticación con Google	Permitir que los usuarios se autentiquen en el sistema utilizando inicio de sesión único con Google, simplificando el acceso y aumentando la seguridad.
RFU5: Notificaciones al usuario	Comunicar al usuario los eventos más relevantes del proceso.
RFU6: Soporte multi idioma	Proporcionar al usuario la posibilidad de utilizar la aplicación en distintos idiomas, adaptando la interfaz de acuerdo a su preferencia o localización.
RFU7: Gestión de créditos de tienda	Permitir al usuario visualizar, acumular y gestionar los créditos disponibles en su cuenta.
RFU8: Canjear créditos de tienda	Permitir al usuario, hacer uso de sus créditos de tienda al comprar un nuevo producto.

Tabla 1 - Requerimientos funcionales de usuario

6.7.2. Tienda

Requerimiento Funcional	**Descripción**
RFT1: Métricas	Permitir a las tiendas acceder a un panel de control con métricas clave relacionadas con las devoluciones (motivos, porcentaje de incidencias por talle, etc.).
RFT2: Visualización de las devoluciones	Proporcionar a las tiendas un listado detallado de todas las devoluciones iniciadas por los usuarios.
RFT3: Aprobar y rechazar devoluciones/cambios	Permitir a las tiendas gestionar devoluciones iniciadas, tomando decisiones sobre su aprobación o rechazo.
RFT4: Configurar políticas de devolución	Permitir a las tiendas definir y ajustar reglas específicas de devolución según propiedades de productos, tiempos

	o condiciones comerciales.
RFT5: Suscripción a eventos del sistema	Permitir que las tiendas se suscriban a notificaciones o eventos relevantes (ej. devolución iniciada, cambio aprobado).
RFT6: Integrar datos de devoluciones en sistemas de terceros	Proporcionar mecanismos de integración para sincronizar datos de devoluciones con sistemas de gestión internos de la tienda.
RFT7: Creación de usuario y autenticación mediante email y contraseña	Registrar empleados de la tienda en el sistema y permitir acceso con credenciales propias, asegurando control de permisos según roles.
RF8: Gestión de créditos	Permitir a las tiendas administrar créditos emitidos a los usuarios.

Tabla 2 - Requerimientos funcionales de tienda

6.7.3. Fitit

Requerimiento Funcional	**Descripción**
RFF1: *CRUD* básico de usuarios	Permitir la creación, lectura, actualización y eliminación de usuarios dentro del sistema de gestión.
RFF2: *CRUD* básico de tiendas	Permitir la creación, lectura, actualización y eliminación de registros de tiendas en la plataforma.
RFF3: Métricas	Proporcionar a Fitit un panel centralizado con métricas y estadísticas sobre devoluciones, tiendas y usuarios.
RFF4: Visualización de las devoluciones	Permitir a Fitit acceder al listado de devoluciones activas e históricas, con el fin de auditar y monitorear el servicio.
RFF5: Changelog de devoluciones y cambios	Permitir registrar de manera histórica todos los eventos relevantes relacionados con devoluciones y cambios.

Tabla 3 - Requerimientos funcionales de Fitit

6.7.4. Integraciones con terceros

Requerimiento Funcional	**Descripción**
RFI1: Integración con cadeterías	Permitir la comunicación automática con proveedores de delivery para coordinar el transporte de productos y actualizar de manera dinámica el estado de los envíos a lo largo de todo el proceso de devolución o cambio
RFI2: Integración en la web de la tienda	Permitir que las tiendas integren Fitit Returns en sus sitios web.
RFI3: Integración con	Facilitar la conexión con sistemas de venta online para

plataformas de *e-commerce*	acceder a detalles de órdenes, productos manteniendo consistencia y trazabilidad.
RFI4: Integración con el recomendador de talles	Conectar el flujo de devoluciones y cambios con el recomendador de talles de Fitit, permitiendo sugerencias de talle que reduzcan la probabilidad de devoluciones futuras.

Tabla 4 - Requerimientos funcionales de terceros

6.8. Requerimientos no funcionales

Los requerimientos no funcionales (RNF) establecen las propiedades de calidad que el sistema debe cumplir, asegurando que las funcionalidades se entreguen de manera eficiente, segura y con una buena experiencia de usuario. A continuación se detallan los RNF asociados a casos de uso específicos.

RNF	**Información**
RNF-US1	**Descripción:** El tiempo de aprendizaje promedio de los usuarios para generar una devolución no debe superar los 2 minutos. **Atributo de calidad:** Usabilidad - Facilidad de aprendizaje
RNF-US2	**Descripción:** La interfaz del usuario final debe ser *responsive* y permitir completar las acciones desde dispositivos móviles y tablets. **Atributo de calidad:** Usabilidad - Flexibilidad
RNF-US3	**Descripción:** La tienda debe contar con atajos como por ejemplo filtros para procesar devoluciones rápidamente. **Atributo de calidad:** Usabilidad - Flexibilidad y eficiencia de uso (Nielsen)
RNF-US4	**Descripción:** La tienda debe permitir encontrar y procesar devoluciones en menos de 5 clicks. **Atributo de calidad:** Usabilidad - Eficiencia (Nielsen)
RNF-US5	**Descripción:** El sistema debe mostrar *feedback* inmediato (mensajes, loaders, animaciones) ante cada acción del usuario. **Atributo de calidad:** Usabilidad - Visibilidad del estado del sistema (Nielsen)
RNF-US6	**Descripción:** El flujo de devolución y cambio debe usar lenguaje claro, evitando tecnicismos y facilitando la comprensión de los pasos. **Atributo de calidad:** Usabilidad - Correspondencia entre el sistema y el mundo real (Nielsen)
RNF-US7	**Descripción:** Debe permitir deshacer o cancelar acciones como

	devoluciones iniciadas o cambios pendientes, antes de su confirmación final. **Atributo de calidad:** Usabilidad - Control y libertad del usuario (Nielsen)
RNF-US8	**Descripción:** Los elementos de la interfaz (botones, títulos, colores) deben mantener consistencia visual y terminológica con la identidad de Fitit, favoreciendo la familiaridad. **Atributo de calidad:** Usabilidad - Consistencia y estándares (Nielsen)
RNF-US9	**Descripción:** El sistema debe proporcionar mensajes de error claros y acciones sugeridas para que el usuario pueda recuperarse fácilmente ante fallos o campos incompletos. **Atributo de calidad:** Usabilidad - Prevención y recuperación de errores (Nielsen)
RNF-US10	**Descripción:** Las opciones, datos y acciones relevantes deben mostrarse en pantalla para que el usuario no deba recordar información de pantallas previas (por ejemplo: mostrar resumen del pedido, talle seleccionado, política aplicable). **Atributo de calidad:** Usabilidad - Reconocer en lugar de recordar (Nielsen)
RNF-US11	**Descripción:** La interfaz debe priorizar la información relevante, evitando sobrecarga cognitiva y manteniendo un diseño limpio. **Atributo de calidad:** Usabilidad - Diseño estético y minimalista (Nielsen)
RNF-US12	**Descripción:** El sistema debe ofrecer ayuda contextual y guiada, evitando documentaciones extensas (tooltips, *FAQ,* "¿Dónde encontrar tu número de orden?"). **Atributo de calidad:** Usabilidad - Ayuda y documentación accesible (Nielsen)
RNF-SEG1	**Descripción:** Toda información sensible, incluyendo credenciales y datos de usuarios, debe almacenarse cifrada y los accesos controlados según roles. **Atributo de calidad:** Seguridad - Confidencialidad
RNF-SEG2	**Descripción:** Los endpoints de integración con terceros deben requerir autenticación segura y control de permisos. **Atributo de calidad:** Seguridad - Integridad
RNF-SEG3	**Descripción:** El sistema debe garantizar que los créditos de tienda sean gestionados de forma segura, evitando usos indebidos o duplicados. Solo el usuario propietario de los créditos puede canjearlos, transferirlos o ver su saldo, y todas las transacciones

	deben registrarse de manera segura y auditable. **Atributo de calidad:** Seguridad - Autenticación y trazabilidad
RNF-PERF1	**Descripción:** Al menos el 95% de las operaciones críticas deben completarse en menos de 500 ms. **Atributo de calidad:** *Performance* - Tiempo de respuesta
RNF-PERF2	**Descripción:** Los dashboards deben cargar en menos de 2000 ms para el 99% de los casos. **Atributo de calidad:** *Performance* - Tiempo de respuesta
RNF-PERF3	**Descripción:** Las imágenes y recursos estáticos deben servirse optimizados para minimizar el tiempo de carga inicial. **Atributo de calidad:** *Performance* - Eficiencia de recursos
RNF-OBS1	**Descripción:** El sistema debe registrar eventos de usuario en tiempo real para poder auditar y resolver incidentes rápidamente **Atributo de calidad:** Observabilidad - Trazabilidad
RNF-OBS2	**Descripción:** El *backoffice* debe registrar todas las acciones de las tiendas sobre devoluciones y cambios. **Atributo de calidad:** Observabilidad - Auditoría
RNF-OBS3	**Descripción:** El sistema debe registrar y monitorear eventos críticos de la aplicación, de manera que se puedan identificar, diagnosticar y resolver problemas de forma proactiva. **Atributo de calidad:** Observabilidad - Monitoreo y diagnóstico

Tabla 5 - Requerimientos no funcionales priorizados

Además de los requerimientos no funcionales priorizados, durante la fase de diseño se identificaron otros RNF que, si bien no fueron el foco principal, se cumplieron de manera natural gracias a las buenas prácticas y la arquitectura adoptada. Entre ellos se destacan:

RNF	**Información**
RNF-MAN1	**Descripción:** El código fuente debe estar estructurado modularmente, con responsabilidades bien definidas y bajo acoplamiento, de modo que las modificaciones o nuevas funcionalidades solicitadas por el cliente puedan realizarse sin afectar otras partes del sistema. **Atributo de calidad:** Mantenibilidad - Facilidad de cambio
RNF-MAN2	**Descripción:** El sistema debe contar con una cobertura de tests

	automatizados mínima del 80% en módulos críticos, para garantizar estabilidad ante actualizaciones o nuevas integraciones. **Atributo de calidad:** Mantenibilidad - Confiabilidad ante cambios
RNF-INT1	**Descripción:** El sistema debe poder integrarse fácilmente con distintas plataformas de *e-commerce* (por ejemplo: Shopify, Tiendanube, VTEX, Magento). **Atributo de calidad:** Integrabilidad

Tabla 6 - Requerimientos no funcionales sin prioridad

6.9. Restricciones

Las siguientes restricciones establecen los límites y requisitos tecnológicos impuestos por el cliente desde el día cero, asumiendo con total conciencia que las solución iba a ser condicionada por ello:

6.9.1. Tecnologías

- ***Backend***: Debe implementarse con Firebase *Cloud Functions* y Google Cloud Platform (GCP) para alinearse con la infraestructura existente del cliente y asegurar la escalabilidad.
- ***Frontend***: Debe desarrollarse en React.js y Next.js para garantizar el renderizado del lado del servidor y un rendimiento optimizado.
- **Base de datos:** Se utilizará Firestore para el manejo de datos en tiempo real y facilitar la escalabilidad.
- **Librerías/Estilos:** Tailwind es la herramienta principal y obligatoria de estilización para mantener la consistencia visual y agilizar el diseño. Durante el proyecto, se incorporó además Radix UI para mejorar la experiencia de componentes interactivos

6.9.2. Integración con la Herramienta Actual

El cliente establece como restricción que la solución debe utilizar el recomendador de talles, al menos dentro del flujo de cambio, para favorecer la selección de la talla correcta y reducir devoluciones. Adicionalmente, se debe posibilitar la correlación de los datos de devoluciones y cambios registrados en Fitit Returns con los datos de uso del recomendador en la misma tienda, con el fin de evaluar y demostrar el impacto del recomendador de talles sobre la reducción de devoluciones.

6.9.3. Usabilidad y Diseño

Se debe respetar el diseño, los estilos, la tipografía, los colores y las prácticas de usabilidad del producto actual de la empresa. Ver Anexo 12.28. Diseño y estilos de Fitit.

7. Diseño arquitectónico

En este capítulo se describe la arquitectura diseñada por el equipo para cumplir con los requerimientos funcionales y no funcionales del sistema. Desde las primeras etapas del proyecto se comprendió la importancia de definir una buena arquitectura, ya que las decisiones iniciales impactarían directamente en el desarrollo y mantenimiento de la aplicación.

Para realizar el diseño de arquitectura se tomaron como referencia los principios presentados en *Software Architecture in Practice* [31], sumado a la experiencia laboral y académica de los integrantes del equipo. Para documentar correctamente se adoptó un enfoque basado en *Views & Beyond*, estándar propuesto en *Documenting Software Architectures: Views and Beyond* [32]. Este enfoque permite representar las principales decisiones de diseño mediante diferentes vistas y diagramas *UML*.

7.1. Arquitectura a Alto Nivel

Esta sección ofrece una representación general de la arquitectura del sistema, con el objetivo de brindar al lector una comprensión inicial. Muestra los componentes más relevantes y las tecnologías involucradas, presentados de manera esquemática, para facilitar una introducción clara antes de entrar en detalles más técnicos.

7.1.1. Representación Primaria

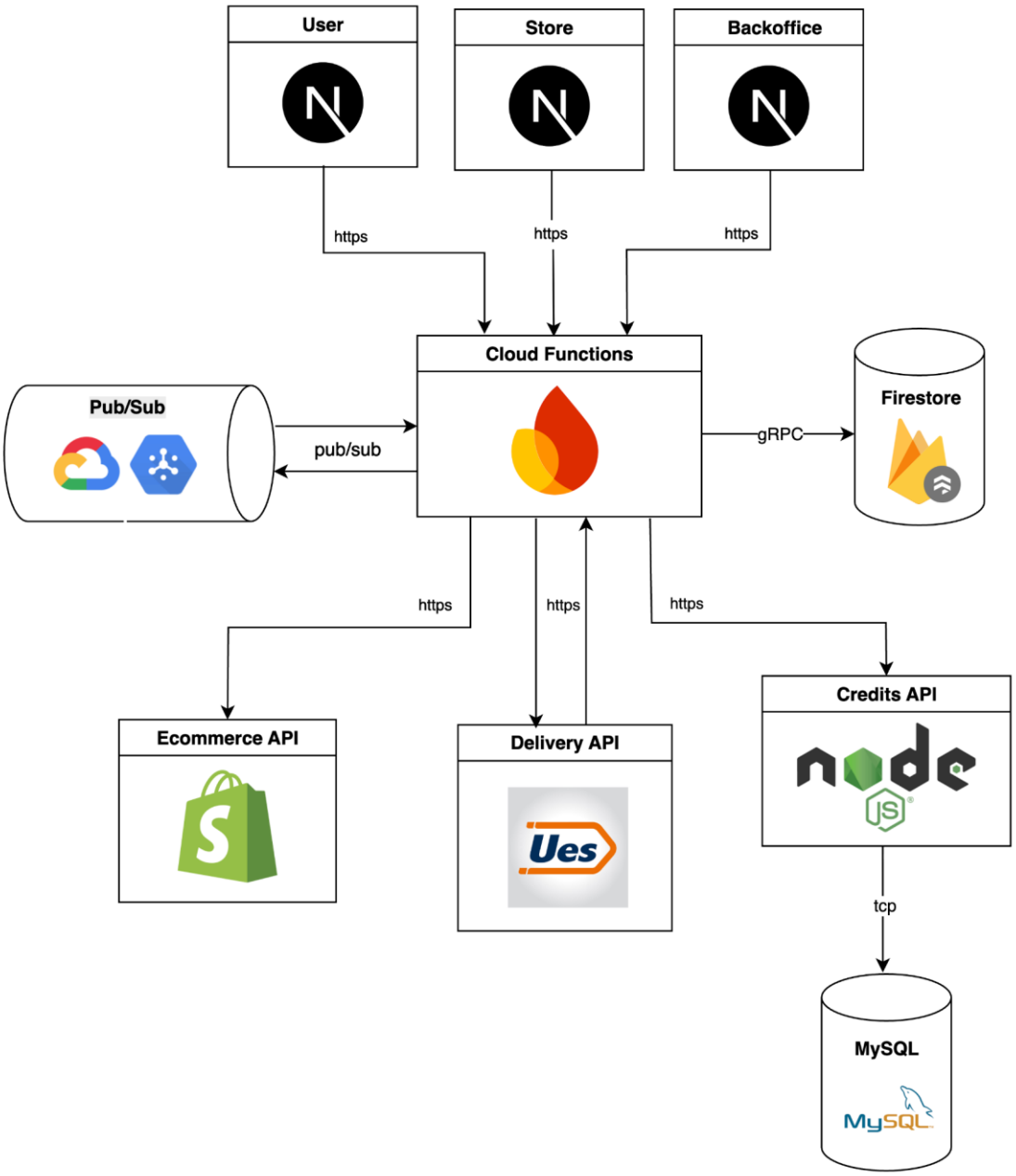


Ilustración 46 - Diagrama de Arquitectura a Alto Nivel

7.1.2. Catálogo de Elementos

A continuación se describen los componentes principales que conforman la arquitectura del sistema, detallando su rol, la tecnología empleada y su propósito dentro del flujo de la información.

- ***Frontends*:** Tres interfaces de usuario separadas que atienden a los distintos actores: usuarios finales, operadores de tiendas y personal interno de Fitit. Todas las instancias inician la comunicación con la lógica de negocio central a través de *HTTPS*.

- ***Cloud Functions*:** El motor central de la lógica de negocio, implementado con una arquitectura *serverless* [33] y actuando como el principal orquestador del sistema. Recibe peticiones de los *Frontends*, procesa eventos asíncronos, y coordina las llamadas a los servicios internos y externos.

- **Google Cloud Pub/Sub:** El servicio de mensajería que permite la comunicación asíncrona y el desacoplamiento entre componentes. Las *Cloud Functions* publican eventos para luego ser procesados por otras funciones o servicios, facilitando la escalabilidad y el manejo de tareas en segundo plano.

- **Credits API:** Un servicio interno que encapsula la lógica de negocio relacionada con la gestión de créditos. Desarrollado en Node.js, este componente es el único autorizado a interactuar directamente con la base de datos MySQL, favoreciendo el cumplimiento del **RNF-SEG Seguridad**.

- **Firestore:** Base de datos *NoSQL* documental que ofrece alta disponibilidad y escalabilidad. La interacción con *Cloud Functions* se realiza mediante *gRPC*.

- **MySQL:** Base de datos relacional utilizada para el almacenamiento de datos estructurados sobre la gestión de créditos. Se accede únicamente de forma directa desde Credits API a través de *TCP*.

- ***E-commerce* API:** Un proveedor externo esencial para el *core* del negocio, que facilita la obtención de órdenes de compra, catálogo de productos y otra información relevante para el flujo comercial. La integración se realiza mediante *HTTPS*.

- **Delivery API:** Un proveedor externo dedicado a la gestión logística. Se utiliza para publicar envíos y suscribirse a *webhooks* o eventos que permiten actualizar internamente los estados de los envíos conforme estos progresan. La comunicación se realiza mediante *HTTPS*.

7.2. Elección de Tecnologías

El equipo dedicó tiempo a analizar y seleccionar las tecnologías más adecuadas para cada componente del sistema: *frontend*, *backend*, base de datos, colas de mensaje y proveedores en la nube. Algunas de estas decisiones fueron tomadas previo a iniciar el desarrollo, mientras que otras fueron surgiendo a medida que la arquitectura iba escalando con el surgimiento de nuevas necesidades.

7.2.1. *Frontend*

Las tecnologías a evaluar en *frontend* eran Next.js y Nuxt.js, ambas restricciones de Fitit. Ambos *frameworks* comparten varias características como el *Server Side Rendering (SSR)*. A continuación, se detalla un análisis comparativo entre ambas herramientas, focalizado en los criterios donde existen diferencias, tales como comunidad, madurez del ecosistema y experiencia actual del equipo:

Criterio	**Next.js**	**Nuxt.js**	**Comentario**
Descargas semanales en NPM en los últimos 3 años [34]	Se nota un claro crecimiento desde ~3 millones hasta ~15 millones	Se encuentra estancado en ~1 millón de descargas	Queda en evidencia que Next.js tiene un uso muy superior que Nuxt.js, lo cual sugiere que tiene también una comunidad mayor.
Cantidad de estrellas en GitHub a la fecha de hoy [34]	~130 mil estrellas	~56 mil estrellas	La diferencia es significativa, lo que sugiere que Next.js tiene una comunidad más grande y despierta mayor interés general.
Experiencia de los integrantes de equipo con la herramienta	Todos los integrantes del equipo suman ~7 años de experiencia en Next.js	Solamente Federico cuenta con 1 año de experiencia usando Nuxt.js	Nuxt.js supondría una curva de aprendizaje dolorosa para el equipo.

Tabla 7 - Comparación entre Next.js y Nuxt.js

Se evidencia en el Anexo 12.13. Gráfica de descargas npm next vs. nuxt la comparación entre las descargas semanales de Next.js y Nuxt.js.

El equipo decidió que Next.js era la herramienta adecuada para el desarrollo de la aplicación. No solo por la comunidad y soporte actual, la cual es muy superior a Nuxt.js, sino también por la experiencia actual del equipo en la tecnología, lo que supondrá una mayor velocidad, agilidad y mejor toma de decisiones a lo largo del proyecto.

7.2.2. *Backend*

7.2.2.1. *Cloud Functions*

La elección tecnológica del *backend* estuvo condicionada inicialmente por una restricción impuesta por Fitit: todo el *backend* debía estar desplegado como *Cloud Functions* utilizando Firebase como plataforma principal. Esta decisión se debe a que el cliente ya contaba con gran parte de su infraestructura implementada de esta forma, lo que aseguraba compatibilidad, facilidad de integración y reducción de costos operativos.

7.2.2.1. Credits API

Cuando surgió la necesidad de implementar la gestión de créditos, el equipo decidió que es sumamente importante manejar esta información en un sistema aislado. Esto se debe a que se iba a manejar dinero de tiendas y usuarios. La gestión de dinero debía ser segura (**RNF-SEG Seguridad**) y robusta, aparte de brindar trazabilidad total de cada crédito otorgado.

El equipo realizó un análisis comparativo de diferentes alternativas para la construcción de dicha *API*. La tabla comparativa de las alternativas se encuentra en Anexo 12.14. Tabla comparativa tecnologías Credits API. Cabe destacar que la comparación se acotó al entorno de Javascript ya que todas las aplicaciones hasta el momento utilizan un *framework* o librería de este ecosistema. Se decidió que no valía la pena considerar opciones que difieran tanto del *stack* tecnológico propuesto inicialmente, al menos que hubiera una limitante o dolor muy fuerte, pero no era el caso.

Considerando el análisis, el equipo decidió utilizar Express.js como *framework* base para la *API*. Esta decisión se fundamentó por su madurez y estabilidad, asegurando soporte a largo plazo, y también por la curva de aprendizaje de bajo nivel que la herramienta presenta.

Cabe destacar que la *API* fue desarrollada utilizando TypeScript, con el objetivo de aprovechar tipado estático, mejorar la detección temprana de errores y garantizar un mantenimiento más robusto a largo plazo. Esta elección permitió sumar un nivel extra de seguridad y control en un entorno donde la integridad de datos es crítica.

7.2.3. Base de Datos

7.2.3.1. *Cloud Functions*

La elección de Firestore como base de datos principal para la arquitectura *serverless* respondió a varios factores: integración nativa con Firebase *Cloud Functions*, escalabilidad automática y baja latencia en lectura y escritura (**RNF-PERF *Performance***). Firestore es ideal para sistemas que requieren sincronización en tiempo real y una estructura flexible de documentos.

7.2.3.2. Credits API

Para Credits API se evaluaron MySQL, PostgreSQL y SQL Server como posibles opciones de base de datos relacional.

- **MySQL:** Amplio soporte, madurez, estabilidad y facilidad de uso. Alta disponibilidad de herramientas y soporte en múltiples entornos cloud. Bajo coste operativo y gran experiencia previa del equipo.

- **PostgreSQL:** Más robusto en operaciones complejas y avanzado en soporte a consultas *SQL*, pero con mayor complejidad de configuración.

- **SQL Server:** Potente, con características empresariales sólidas, pero con mayores costes y dependencia de licenciamiento.

Se eligió **MySQL** por su simplicidad, robustez, velocidad y la experiencia previa del equipo con la herramienta.

7.2.4. Cola de Mensajes

Para la arquitectura de mensajería se evaluó principalmente Pub/Sub de Google Cloud Platform (GCP) frente a Amazon SQS.

- **Pub/Sub GCP:** Integración nativa con Firebase *Cloud Functions,* escalabilidad automática, soporte robusto de mensajes en tiempo real y administración simplificada.

- **SQS AWS:** Muy confiable, ampliamente utilizado y provee integración nativa con otros servicios de AWS.

Dado que la mayoría de los servicios del *backend* están desplegados en GCP y que Pub/Sub ofrece integración directa y optimizada dentro de este ecosistema, se eligió **Pub/Sub de GCP** como sistema de cola de mensajes. Esta decisión reduce la complejidad arquitectónica y minimiza los puntos de fallo.

7.2.5. Proveedor en la Nube

La arquitectura principal se desarrolló sobre **Google Cloud Platform** (GCP), siguiendo la restricción impuesta por el cliente y aprovechando la integración nativa con Firebase. GCP ofrecía un ecosistema cohesivo y escalable que satisfacía la mayoría de los requisitos técnicos y funcionales del proyecto. Fue fundamental para satisfacer el cumplimiento del **RNF-OBS Observabilidad**, ya que trae incorporado por defecto el *monitoring* de los servicios utilizados.

Sin embargo, durante el análisis de costos, el equipo detectó que mantener la base de datos MySQL en GCP tendría un costo elevado para el presupuesto evaluado con el cliente. Dado este escenario, y tras evaluar opciones de optimización, se decidió incorporar **Amazon Web Services** (AWS) exclusivamente para alojar la base de datos MySQL de Credits API.

Esta decisión respondió a dos factores principales:

- Optimización de costos: AWS ofrecía una solución más económica para MySQL, especialmente considerando los recursos gratuitos (150 créditos) otorgados por la facultad ORT.

- Compatibilidad técnica: AWS RDS ofrecía alta disponibilidad y escalabilidad para el tipo de carga esperada en Credits API.

La estrategia resultante fue utilizar GCP como proveedor principal para los servicios core y AWS exclusivamente para la base de datos MySQL, logrando así una arquitectura híbrida que optimiza costes y rendimiento.

7.3. Vista de Módulos

7.3.1. Vista de Capas

7.3.1.1. Representación Primaria

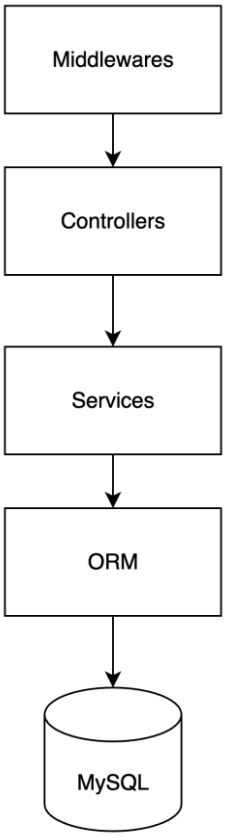


Ilustración 47 - Diagrama de Capas de Credits API

7.3.1.2. Catálogo de Elementos

- **Middlewares:** Valida y normaliza los datos entrantes antes de llegar a la lógica de negocio. Aquí se incluyeron validaciones de *body*, *params* y *query params*.

- **Controllers:** Reciben las solicitudes *HTTPS* para redirigirlas hacia los servicios correspondientes. No contenían lógica de negocio, manteniendo la separación de responsabilidades.

- **Services:** Maneja la lógica de créditos central. Gestiona operaciones como validaciones de saldo, transacciones de entrada y salida, etc.

- **ORM (Prisma):** Facilita el acceso a la base de datos MySQL con tipado estático y control de transacciones. Evita la necesidad de un patrón adicional como Repository, simplificando la arquitectura.

- **Base de Datos (MySQL):** Almacena datos estructurados y soporta operaciones transaccionales seguras.

7.3.1.3. Decisiones de Diseño

Una de las principales decisiones arquitectónicas fue no aplicar una estructura en capas dentro del módulo de *Cloud Functions.* Este enfoque se adoptó de manera consciente, reconociendo la naturaleza granular, aislada y efímera de dichas funciones. Cada *Cloud Function* debía ser autónoma y responder rápidamente a un evento específico, sin depender de una jerarquía interna de capas. Si bien existían utilidades y módulos compartidos, el equipo consideró que imponer un esquema *layered* rígido iría en contra de los principios de simplicidad y escalabilidad propios del modelo *serverless* [35] [36].

Por el contrario, en Credits API sí se aplicó una arquitectura en capas tradicional, lo que permitió una mejor organización del código, separación de responsabilidades y una mayor facilidad para realizar pruebas unitarias. Esto favoreció mucho el desarrollo en *TDD* [37] de la capa de servicios de la aplicación.

Asimismo, se decidió no implementar el *Patrón Repository* sobre el *ORM* utilizado. Dado que Prisma ya proporciona una abstracción completa para las operaciones de acceso a datos —incluyendo validaciones, tipado fuerte, relaciones y transacciones—, agregar un *wrapper* adicional habría introducido complejidad innecesaria sin aportar beneficios reales.

7.3.2. Vista de Descomposición

7.3.2.1. Representación Primaria

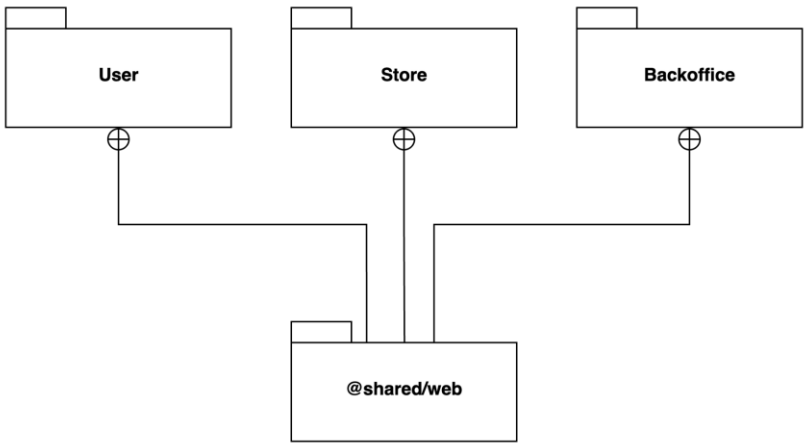


Ilustración 48 - Diagrama de descomposición de los *Frontends*

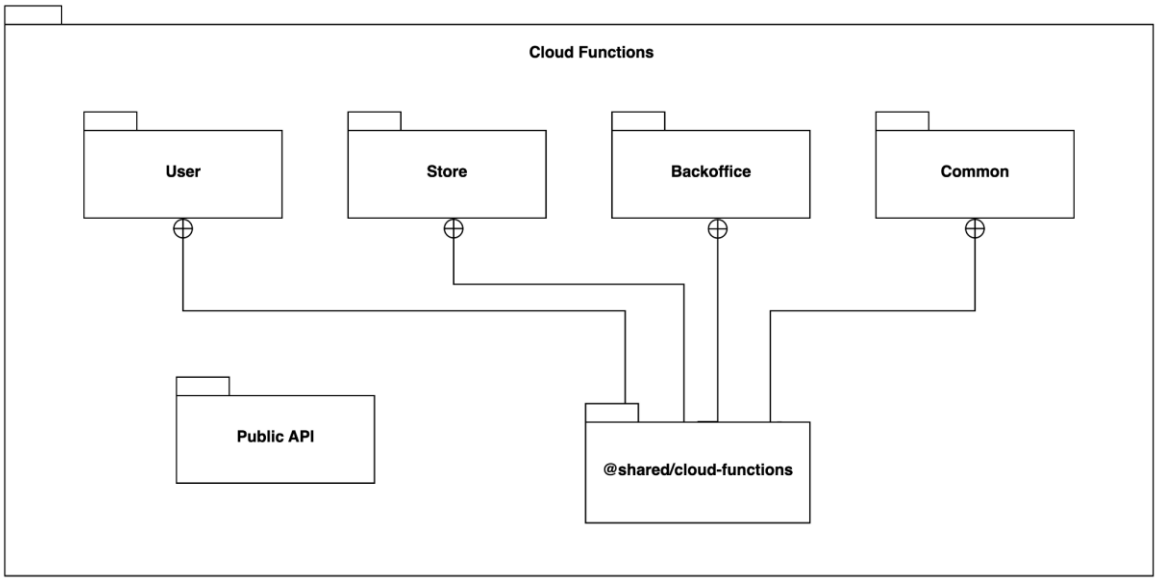


Ilustración 49 - Diagrama de descomposición del módulo *Cloud Functions*

7.3.2.2. Catálogo de Elementos

- **@shared/web:** Corresponde a una librería compartida entre los distintos *Frontends* (User, Store y *Backoffice*). Su propósito fue centralizar todos los componentes y utilidades comunes, garantizando coherencia visual y reutilización de código.

- ***Frontends* (User, Store y *Backoffice*):** Cada uno de estos módulos representa una interfaz web independiente desplegada en Firebase Hosting. User está orientado al usuario final, permitiéndole gestionar devoluciones,

visualizar el estado y acceder a su historial. Store está diseñado para las tiendas, brindando funcionalidades de gestión de solicitudes, coordinación logística y resolución de devoluciones. Finalmente, *Backoffice* concentra las herramientas administrativas internas de Fitit.

- **@shared/cloud-functions:** Librería compartida por los distintos módulos del *backend* desplegados en Firebase (User, Store, *Backoffice* y Common). Su objetivo principal fue proveer un punto centralizado para la creación y configuración de funciones *serverless* mediante una *Factory* de *Cloud Functions*, la cual encapsula toda la lógica repetitiva y estandariza la inicialización de las funciones.

- ***Cloud Functions*:**

    - **User, Store, *Backoffice* y Common:** Cada submódulo de *Cloud Functions* agrupa las funciones *serverless* específicas para su dominio. Cada uno atiende a los casos de uso de un actor en particular, que se corresponde con los tres *Frontends* ya explicados. El submódulo *Common* encapsula funciones reutilizables por más de un actor.

    - **Public API:** Expone *endpoints* versionados para que las tiendas puedan accederlos mediante *API Keys* (validados mediante un middleware llamado *apiKeyAuth*).

7.3.2.3. Decisiones de Diseño

El equipo optó por crear ambas librerías compartidas para fomentar la modularidad y la reutilización de código, buscando reducir duplicaciones y facilitar la evolución del sistema. Ambas librerías fueron publicadas en **NPM**, lo cual permite gestionar de forma prolija las versiones, asegurando un ciclo de actualizaciones ordenado y minimizando la dependencia directa entre proyectos.

En **@shared/web**, se destaca la creación de un *Design System* basado en los principios de *Atomic Design* [38], lo que permitió mantener una identidad visual coherente en todas las interfaces de usuario ya que se utilizaba un mismo tema base (colores, espaciados, márgenes, etc.) y esto aceleraba el desarrollo de nuevas pantallas. Esto es una evidencia fuerte del cumplimiento del **RNF-US Usabilidad**.

Por su parte, en **@shared/cloud-functions** se implementó una *Factory* para la creación de *Cloud Functions* pudiendo así estandarizar la inicialización de las mismas en los diferentes submódulos. El patrón *Factory* permitió abstraer la configuración repetitiva, definir estructuras uniformes aparte de centralizar la autenticación y manejo de errores. Aquí es donde se define un nodo muy importante llamado *auth*, responsable de validar la autenticación según los roles permitidos (USER, STORE y/o INTERNAL).

7.3.3. Vista de Clases

7.3.3.1. Representación Primaria

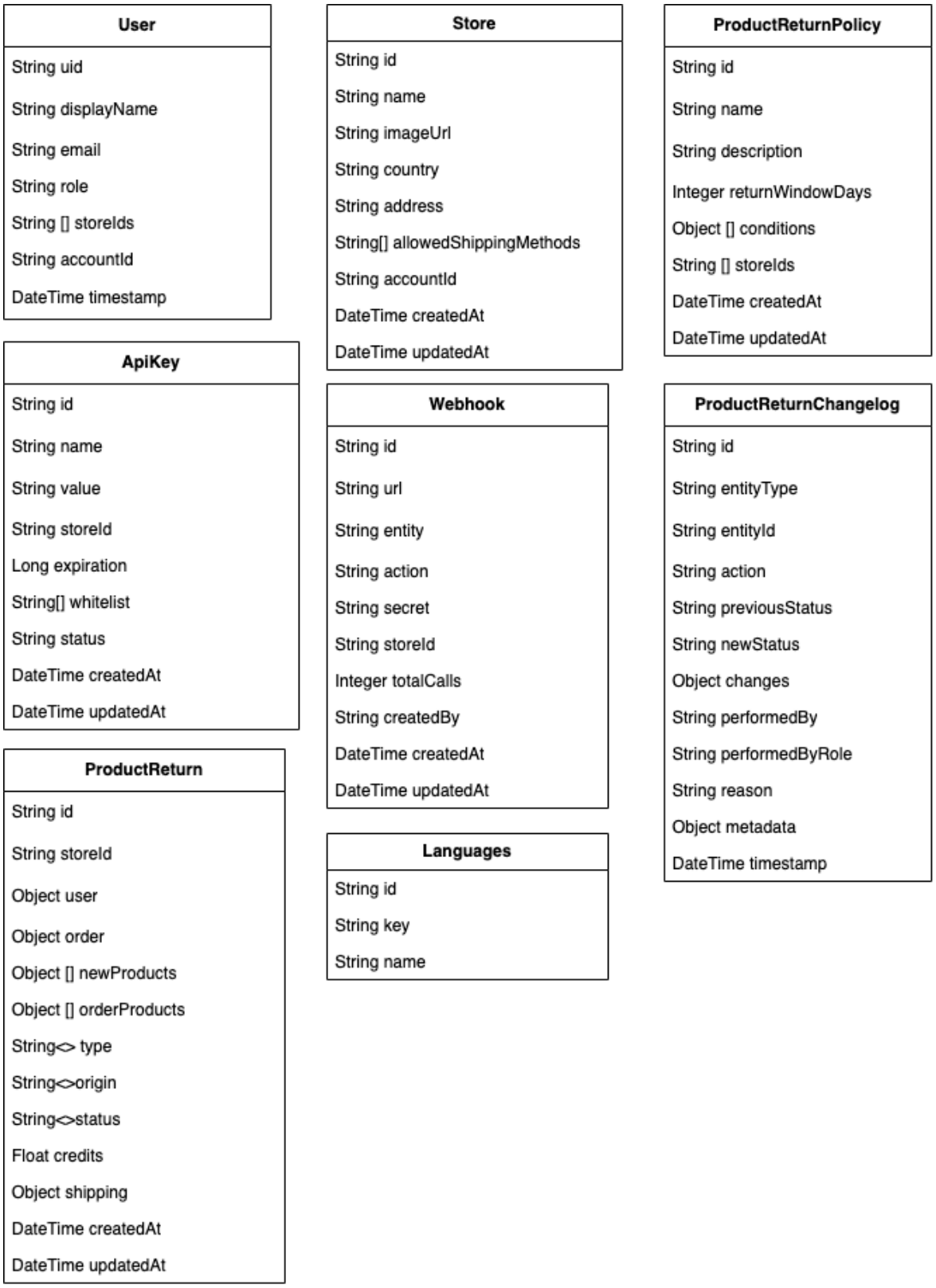


Ilustración 50 - Diagrama de Clases de *Cloud Functions*

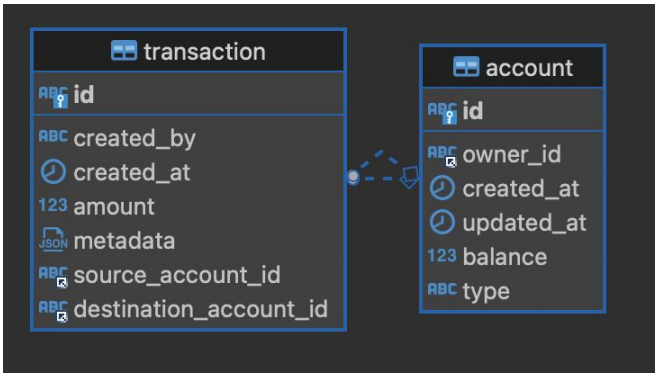


Ilustración 51 - Diagrama ER de Credits API

7.3.3.2. Catálogo de Elementos

A continuación, se detallan las clases y entidades identificadas en los diferentes subsistemas del *backend*, clasificadas según el tipo de almacenamiento que utilizan: *NoSQL* (Firestore) o *SQL* (Credits API).

NoSQL — Cloud Functions

- ***Webhook*:** representa los eventos externos recibidos desde integraciones o servicios de terceros. Contiene la información sobre la entidad afectada, el tipo de acción y los datos necesarios para su trazabilidad.

- **ApiKey:** gestiona las claves de acceso generadas por las tiendas o integraciones externas, garantizando la autenticación y control de acceso a los servicios internos.

- **ProductReturnPolicy:** define las políticas de devolución asociadas a cada tienda, incluyendo su descripción, la cantidad de días de ventana de devolución y las condiciones aplicables.

- **ProductReturnChangelog:** almacena de forma trazable los cambios realizados sobre entidades de devolución o políticas, permitiendo auditar cada modificación a lo largo del tiempo.

- **Store:** representa una tienda dentro del ecosistema. Contiene información como nombre, país, dirección, métodos de envío disponibles y una referencia a su cuenta de créditos dentro del sistema financiero.

- **User:** modela los usuarios registrados en la plataforma, guardando información de identidad, rol, correo electrónico y su cuenta de créditos asociada.

- **ProductReturn:** representa las solicitudes de devolución de productos realizadas por los usuarios. Incluye la referencia a la tienda, el estado de la devolución, los productos involucrados y la información de créditos utilizada o reembolsada.

- **Account:** representa una cuenta de crédito asociada a un usuario o tienda. Mantiene el saldo disponible, el tipo de cuenta y los metadatos necesarios para su trazabilidad.

- **Transaction:** registra cada movimiento financiero entre cuentas dentro del sistema. Cada transacción almacena el origen, el destino, el monto, el usuario que la creó y los metadatos necesarios para garantizar su auditabilidad.

7.3.3.3. Decisiones de Diseño

El diseño del sistema se estructuró bajo un enfoque híbrido que combina bases de datos *NoSQL* y *SQL* para equilibrar flexibilidad operativa y consistencia transaccional. Los módulos de negocio que requieren alta velocidad de desarrollo, escalabilidad y adaptabilidad —como los procesos de devolución, políticas y gestión de usuarios— se persisten con Firestore (*NoSQL*), mientras que las operaciones financieras críticas se guardan en MySQL (*SQL*). Esta separación permite mantener la simplicidad en la capa operativa y, al mismo tiempo, asegurar integridad y trazabilidad en el flujo de créditos.

Credits API funciona como el sistema de verdad financiera, centralizando toda la gestión de créditos y transacciones entre usuarios y tiendas. Cada tienda y cada usuario poseen una cuenta propia en este sistema. Cuando una tienda acredita saldo a un usuario, se genera una transacción de tipo Store → User; cuando el usuario utiliza esos créditos para realizar compras o devoluciones, se registra una transacción User → Store. Este modelo bidireccional permite representar con precisión el flujo de valor y construir un historial financiero totalmente auditable.

7.4. Vistas de Infraestructura

7.4.1. Representación Primaria

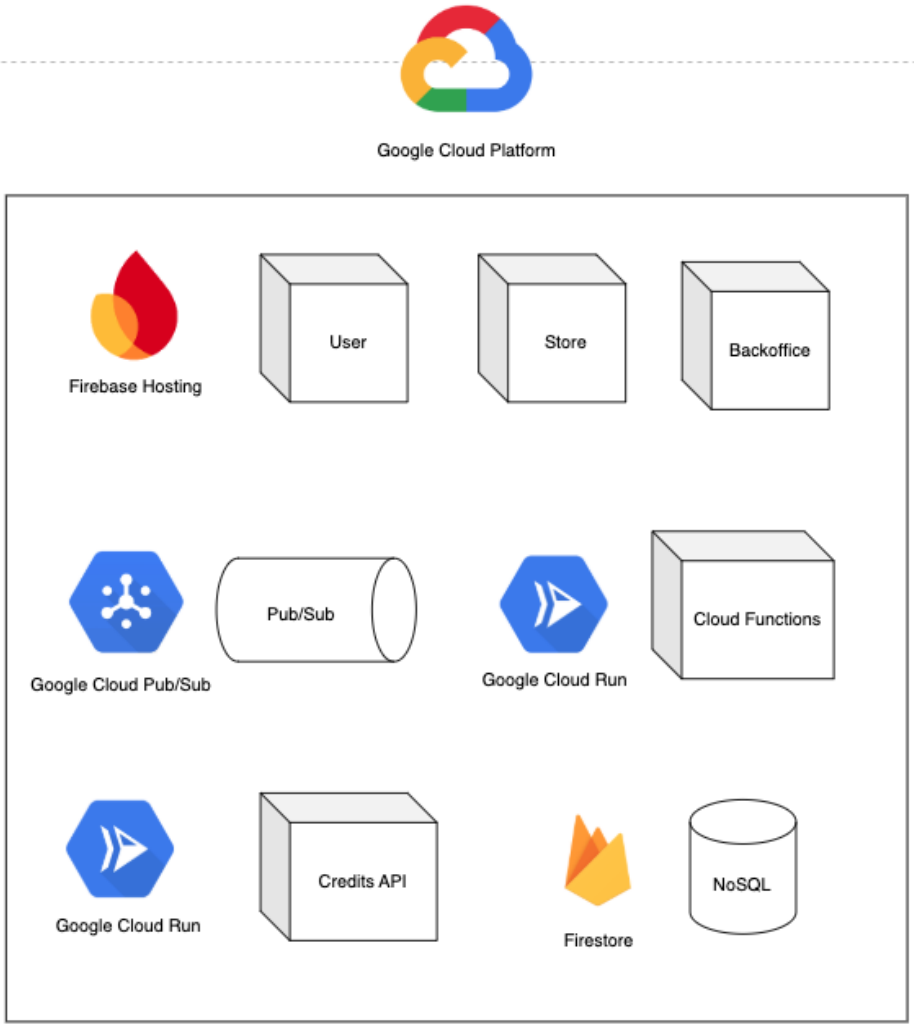


Ilustración 52 - Diagrama de Despliegue de GCP

Ilustración 53 - Diagrama de Despliegue de AWS

7.4.2. Catálogo de Elementos

A continuación, se presenta un listado conciso de los servicios de infraestructura utilizados para el despliegue de la solución.

7.4.2.1. Google Cloud Platform (GCP)

- **Firebase Hosting:** Servicio para el alojamiento de los *Frontends* de las aplicaciones web (User, Store, *Backoffice*).
- **Google Cloud Pub/Sub:** Servicio de mensajería asíncrona para la comunicación desacoplada.
- **Google Cloud Run:** Es la plataforma de contenedores como servicio completamente gestionada y *serverless* de GCP. Permite el despliegue de imágenes de contenedor (Docker) para ejecutar microservicios y *APIs* (como Credits API), ofreciendo escalado automático a cero y sirviendo como *backend* para las *Cloud Functions* de segunda generación.
- **Firestore:** Base de datos *NoSQL* escalable y gestionada.

7.4.2.2. Amazon Web Services (AWS)

- **Amazon RDS:** Servicio de base de datos relacional gestionado, utilizado para correr el motor MySQL.

7.5. Patrones de Diseño

En esta sección se detallan los patrones de diseño que el equipo considera más importantes a destacar, junto con un diagramas ilustrativos para entender mejor el caso de uso.

7.5.1. Patrón Strategy

El patrón *Strategy* [39] se empleó principalmente en los módulos vinculados al *delivery* y al *e-commerce*. Su utilización permitió definir y seleccionar estrategias de comportamiento en tiempo de ejecución según la configuración del entorno o del cliente. De esta forma, fue posible mockear comportamientos durante el desarrollo o las pruebas, y definir estrategias específicas en producción sin modificar la lógica central. Este enfoque facilitó la extensión futura del sistema ante la incorporación de nuevos canales o proveedores (**RNF-INT Integrabilidad**), manteniendo un bajo acoplamiento entre componentes.

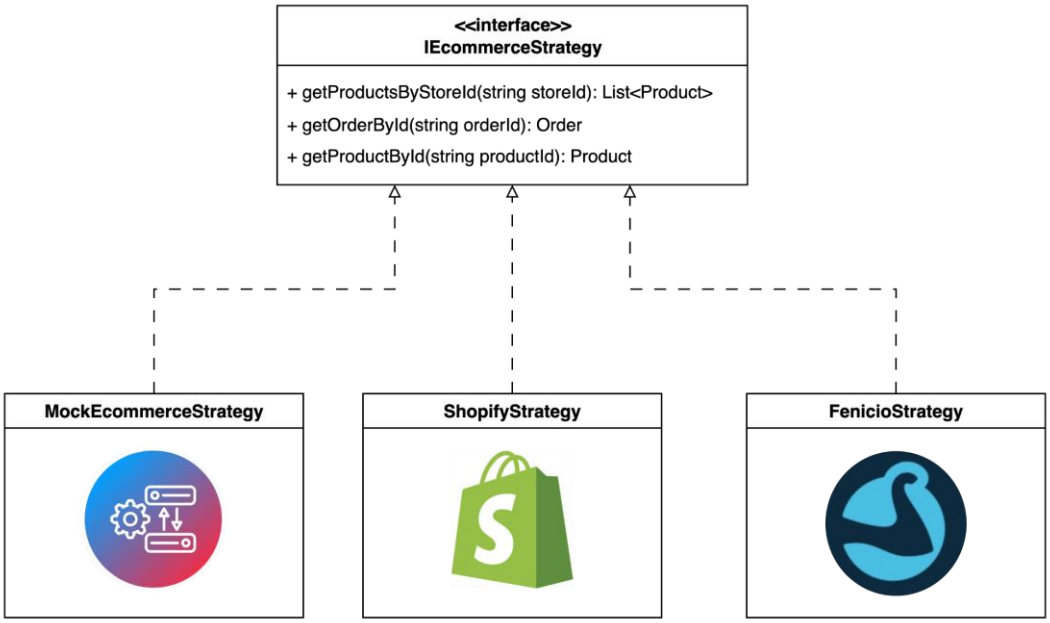


Ilustración 54 - Evidencia del Patrón *Strategy* para los *e-commerce*

7.5.2. Publish/Subscribe

El sistema implementó un modelo de comunicación asincrónica basado en los principios de una *Event-Driven Architecture* [40], con el objetivo de desacoplar procesos, mejorar la escalabilidad y garantizar la trazabilidad de las operaciones. A partir de eventos clave —como la creación o modificación del estado de una devolución— se disparaban acciones internas mediante un mecanismo de publicación y suscripción (Pub/Sub) [41].

Cada evento generaba múltiples efectos secundarios procesados de manera independiente, entre ellos:

- El envío de notificaciones por correo electrónico a los distintos actores involucrados (usuario, tienda e internos).
- La ejecución de *webhooks* hacia las tiendas que contaban con integraciones externas configuradas.
- El registro del evento en un *changelog* histórico que mantenía la trazabilidad completa de cada devolución.

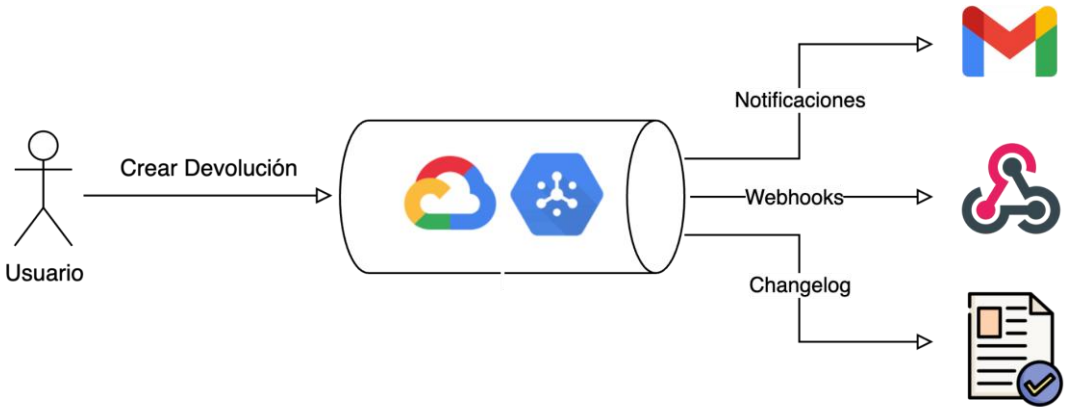


Ilustración 55 - Ejemplo del evento Crear Devolución usando Google Pub/Sub

Estas acciones se ejecutaban de forma asincrónica, evitando bloqueos en el flujo principal del sistema y garantizando la consistencia eventual de los procesos. Esta decisión de diseño permitió alcanzar un equilibrio adecuado entre rendimiento, modularidad y capacidad de expansión futura.

7.5.3. Federated Identity Pattern

Al adoptar Google Firebase Auth como proveedor de identidades externo, se aplica el patrón de identidad federada [42]. En este esquema, la aplicación no gestiona directamente las credenciales del usuario (usuario y contraseña), sino que confía en un proveedor de identidad externo para autenticar a los usuarios.

Esto permite delegar la responsabilidad de validación de credenciales en un sistema de confianza, mientras la aplicación consume los tokens resultantes para otorgar acceso a recursos protegidos. El uso de este patrón es una de las justificaciones sobre el cumplimiento del **RNF-SEG Seguridad**.

7.6. Evolución de la arquitectura a lo largo del tiempo

En esta sección se detallan las distintas versiones de la arquitectura en el transcurso del proyecto.

7.6.1. Versión Inicial

La arquitectura planteada en la Ilustración 56 fue la primera versión definida por el equipo dado los requerimientos funcionales y no funcionales iniciales:

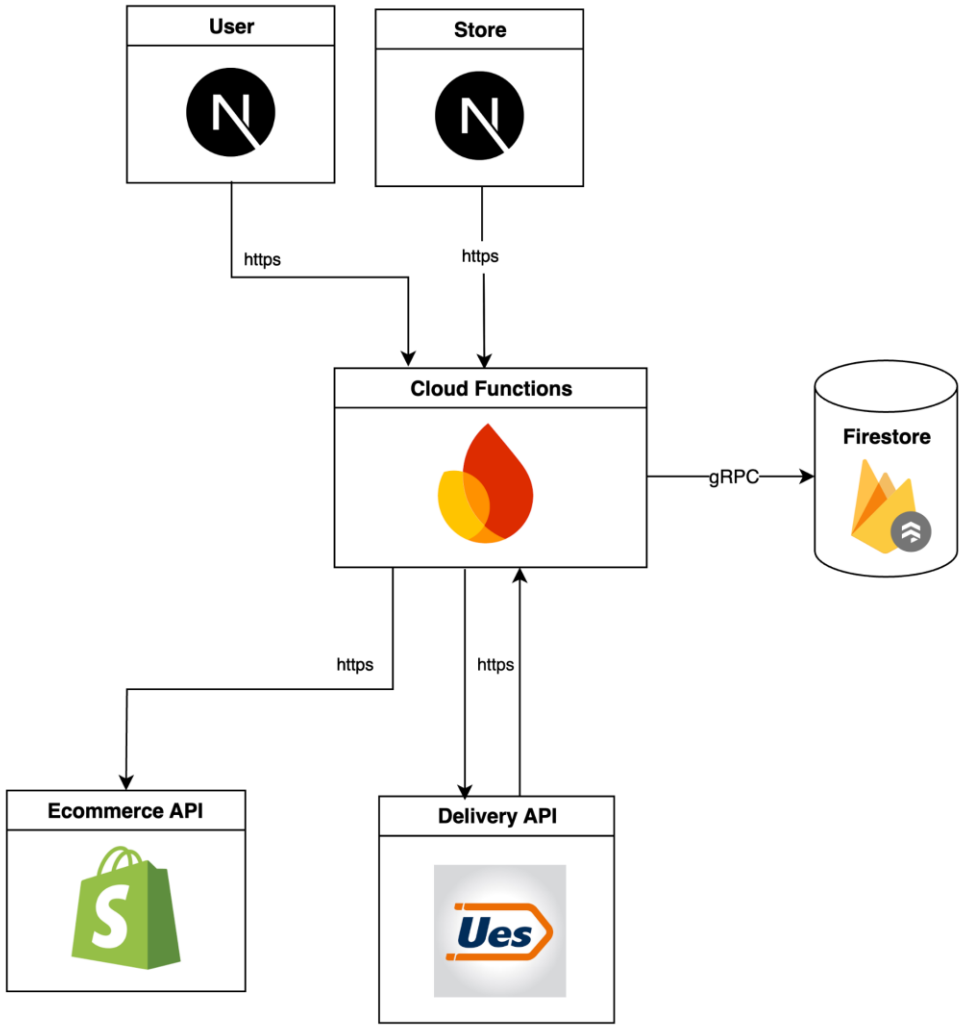


Ilustración 56 - Versión inicial de Arquitectura

Esta arquitectura fue validada por **Dan Bzurovski** (Founder de Fitit) y **Diego Lauz** (Engineer Manager de Strike Security).

7.6.2. Versión Final

Esta versión es la ilustrada en la sección 7.1 Arquitectura Alto Nivel, donde se sumó Credits API que se comunica con una base de datos MySQL y el servicio de mensajería asíncrono Pub/Sub.

Ambos elementos responden a una necesidad clara del negocio. La razón de ser de Credits API ya fue justificada en la sección 7.2.2.1. Credits API. Por otro lado, el surgimiento del esquema Pub/Sub surge por la necesidad de manejar acciones asíncronas asociados a eventos clave de negocio (*ej*. creación de una devolución). El uso de esta herramienta favorece el desacoplamiento entre las distintas partes del sistema, permitiendo que la aplicación escale de mejor manera en el corto y mediano plazo.

8. Aseguramiento de la calidad

A continuación se detallan los objetivos de calidad definidos tanto a nivel de producto como de proceso. Luego se presenta el plan de calidad que se utilizó de guía durante todo el ciclo de vida, las prácticas de aseguramiento (estándares y pruebas), la gestión de incidentes y las métricas seleccionadas para monitorear y mejorar continuamente.

8.1. Objetivos de calidad

El aseguramiento de la calidad se planteó como un aspecto transversal a todo el ciclo de vida del proyecto. El objetivo principal fue garantizar que cada entrega cumpliera con los requerimientos funcionales definidos y además, con los atributos de calidad asociados a los requerimientos no funcionales ya detallados en el capítulo 6. Ingeniería de Requerimientos. Estos aspectos resultaron fundamentales para asegurar que el sistema ofreciera una experiencia fluida, confiable y segura, tanto para las tiendas como para los usuarios finales, aportando valor real al producto y manteniendo la coherencia con los estándares académicos de desarrollo de *software*.

De acuerdo con el IEEE Computer Society, la calidad en *software* se entiende como *"el grado en que el software se ajusta a sus requisitos y satisface las necesidades de sus usuarios"* [43]. Esta definición fue tomada como referencia para orientar las metas de calidad.

Con base en ello, los objetivos de calidad del proyecto se definieron en tres ejes principales:

1. **Calidad del producto**
    - Asegurar que el sistema gestione devoluciones y cambios sin defectos críticos en nuestro ambiente pre-productivo.
    - Brindar una experiencia de usuario clara, intuitiva y confiable, aumentando la confianza en el proceso de postventa online.
    - Diseñar un producto extensible a distintas plataformas de *e-commerce* y servicios de cadetería externos.
2. **Calidad del proceso**
    - Mantener un proceso de desarrollo ágil, con entregas incrementales y *feedback* temprano en cada *sprint*.
    - Minimizar el retrabajo causado por errores de implementación o cambios de alcance, manteniéndolo en niveles controlados.
    - Garantizar la trazabilidad de decisiones y tareas, mediante el uso de herramientas de gestión colaborativas y repositorios centralizados.

3. **Calidad académica**
    - Documentar de forma clara, coherente y completa los aspectos técnicos y de gestión del proyecto.
    - Resaltar los aprendizajes en aseguramiento de calidad, metodologías ágiles y diseño de *software*.
    - Cumplir con los requisitos formales de la institución para la presentación de un trabajo final de carrera.

8.2. Plan de calidad

Para Fitit Returns se elaboró un plan de calidad con el fin de asegurar que tanto el proceso como los entregables se mantuvieran alineados con los objetivos planteados. La idea central fue que la calidad no se revisará únicamente al final, sino que estuviera presente en cada fase del proyecto y en cada *sprint*.

Este plan funcionó como una hoja de ruta donde quedaron definidas las actividades clave, qué herramientas se usarían y quiénes serían los responsables. De esa forma, el equipo pudo mantener un monitoreo constante y corregir desvíos de manera temprana.

El plan se estructuró en distintas fases:

- **Descubrimiento y análisis inicial**: se validó el problema con el cliente, se construyeron los primeros prototipos en Figma (ver Anexo 12.9.9 Prototipos de alta fidelidad), se definió la arquitectura inicial y finalmente se relevaron los requerimientos. Estas actividades permitieron acordar un alcance inicial realista y validar supuestos antes de comenzar el desarrollo.
- **Desarrollo iterativo**: durante los *sprints* se aplicaron prácticas de aseguramiento de calidad como revisiones de código en GitHub, pruebas unitarias de *Cloud Functions* con Jest, pruebas de integración automatizadas con Cypress y pruebas de rendimiento con K6. Además, se utilizó el emulador de Firebase para validar integraciones y flujos de datos. En paralelo, se realizaron *demos* con el cliente para obtener *feedback* temprano, lo que permitió ajustar funcionalidades antes de cada *release*.
- **Documentación y revisiones**: la documentación se construyó en paralelo al desarrollo, con redacciones incrementales revisadas en conjunto. Se priorizó la coherencia entre lo técnico y lo académico, con revisiones cruzadas entre los integrantes y correcciones a partir de los comentarios del tutor.
- **Cierre y entrega final**: se ejecutaron *checklist* de calidad y validación de los criterios de aceptación. Además, se desplegó el sistema y se preparó el material de presentación para la defensa de la tesis.

El detalle del plan de calidad con cada actividad, sus entradas y salidas, responsables y herramientas empleadas se presenta en el Anexo 12.15. Plan de Calidad, lo que

permitió mantener un registro claro y verificable de cómo se controló la calidad a lo largo del ciclo de vida del proyecto.

8.3. Aseguramiento de calidad

8.3.1. Estándares

La definición de estándares de documentación y codificación fue esencial para mantener la consistencia del proyecto **Fitit Returns**, especialmente dado que se desarrolló en un entorno ágil. Los mismos no surgieron sólo de referencias teóricas, sino que se consolidaron a partir de la práctica del equipo y del análisis de buenas prácticas en proyectos similares y del cliente.

8.3.1.1. Estándares de documentación

Se siguieron los lineamientos de formato provistos por la Facultad de Ingeniería para los trabajos finales, incluyendo los documentos:

- **Documento 302** – Contenido y formato general del trabajo final de carrera.
- **Documento 303 y 304** – Aspectos gráficos y de estructura del trabajo a evaluar por los tribunales.
- **Documento 306** – Guías para títulos, resúmenes y abstracts.

Estos lineamientos aseguraron la uniformidad en la redacción, formato, estructura visual y citación bibliográfica de este documento.

8.3.1.2. Estándares de codificación

A continuación y en el Anexo 12.16. Definición de estándares de codificación, se presentan los estándares definidos para las distintas tecnologías utilizadas en el desarrollo de Fitit Returns, junto con las referencias y herramientas aplicadas.

Estándares de código

Tecnología	Estándar adoptado	Referencia
JavaScript / TypeScript	Guía de estilo de Google JS Style Guide. Uso de camelCase para variables y funciones, PascalCase para clases y componentes.	https://google.github.io/styleguide/jsguide.html

React.js / Next.js	Componentes funcionales, *hooks*, y separación clara entre *pages*, *screens*, *widgets* y lógica.	https://react.dev/, https://nextjs.org/docs
Firebase Functions (*Backend*)	Arquitectura modular, CommonJS para compatibilidad, uso de HttpsError para manejo de errores y Yup [44] para validaciones.	https://firebase.google.com/docs/functions
CI/CD y Control de versiones	GitHub Flow con ramas con vida corta y revisiones obligatorias vía *Pull Requests*	https://githubflow.github.io
Testing	*TDD* en módulos críticos y cobertura mínima del 90%. Uso de Jest, *mocks* y *coverage reports*.	https://jestjs.io/

Tabla 8 - Estándares de código

Estándares de formato y estilo

Estándar	Descripción	Herramienta	Justificación
Formateo automático	Aplicación de *Prettier* en todo el repositorio para mantener coherencia en espacios, comillas e indentación.	Prettier	Evita inconsistencias entre desarrolladores.
Linting	*ESLint* configurado con reglas en .eslintrc.json y ejecución automática en cada *commit*.	ESLint	Asegura calidad sintáctica y mejores prácticas.

Commitlint	Verifica que los mensajes de *commit* sigan el formato semántico convencional.	Commitlint + Husky	Facilita el versionado semántico y el *changelog* automático.

Tabla 9 - Estándares de formato y estilo

En el Anexo 12.17. Configuración y aplicación de estándares están las pruebas sobre cómo se aplicaron los estándares.

8.3.2. Pruebas

La prueba de *software* es el proceso de evaluar y verificar un producto para comprobar que hace lo que se supone que debe de hacer. Sus beneficios incluyen la prevención de errores, la reducción de costos y la mejora del rendimiento. A continuación se detallan los distintos tipos de pruebas realizadas para el control de calidad [45].

8.3.2.1. Pruebas automáticas

Las pruebas automáticas cumplieron un papel esencial en la calidad y estabilidad del sistema Fitit Returns, permitiendo detectar errores antes de que los cambios fueran incorporados al ambiente pre-productivo y asegurando que las nuevas funcionalidades no afecten el comportamiento de las ya existentes.

El equipo decidió concentrar los esfuerzos de testing exclusivamente en el *backend*, dado que en esta capa se encontraba la mayor parte de la lógica de negocio. En particular, se priorizó testear las funcionalidades clave:

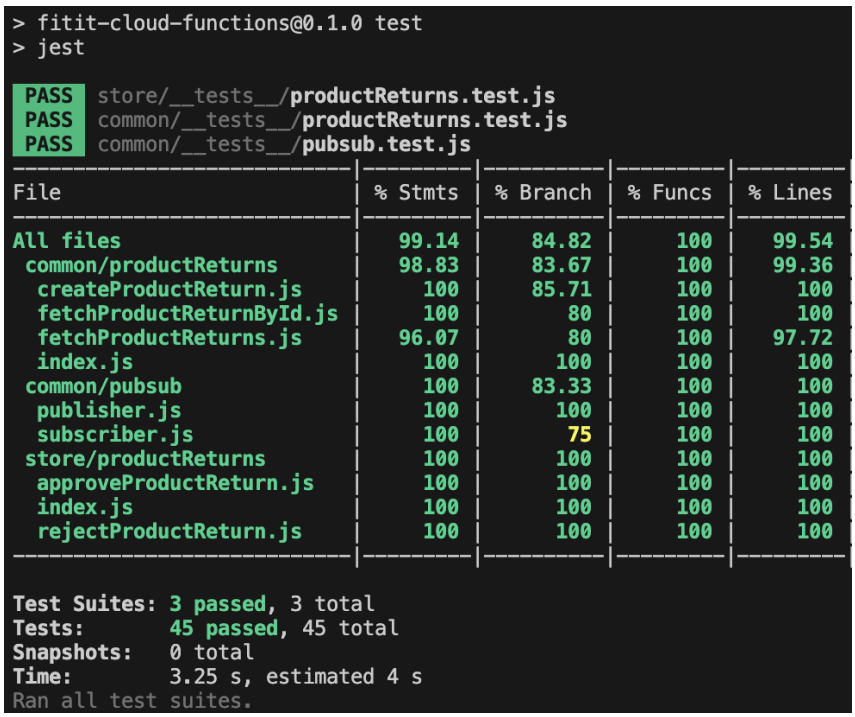
- La creación de una devolución y posterior aprobación o rechazo.
- El sistema de gestión de créditos.

Se consideró que el *frontend* no requería una cobertura inmediata, ya que su funcionalidad dependía directamente de los servicios *backend*, los cuales, al estar validados exhaustivamente, ya garantizaban la correcta ejecución de los flujos principales.

Las pruebas se implementaron utilizando Jest [46], marco de pruebas ampliamente adoptado en entornos JavaScript/TypeScript por su integración con Node.js y su soporte nativo para *mocks*, *coverage reports* y ejecución en paralelo. Cabe destacar que en Credits API se aplicó un enfoque *TDD*. Este enfoque resultó especialmente adecuado, ya que la lógica de negocio —basada en cálculos de transacciones, balances y estados financieros— era tan directa que resultaba más eficiente escribir los tests primero que desarrollar sin ellos.

Según *Graphite* [47], mantener una cobertura de código cercana al 90 % constituye una buena práctica de la industria, ya que representa un equilibrio adecuado entre el esfuerzo de desarrollo y la detección temprana de defectos en los componentes más

críticos del sistema. En función de esta recomendación, el equipo estableció un **objetivo mínimo del 90 % de cobertura global**, con el propósito de asegurar que la mayor parte de los flujos y rutas esenciales estuvieran validados mediante pruebas automatizadas.

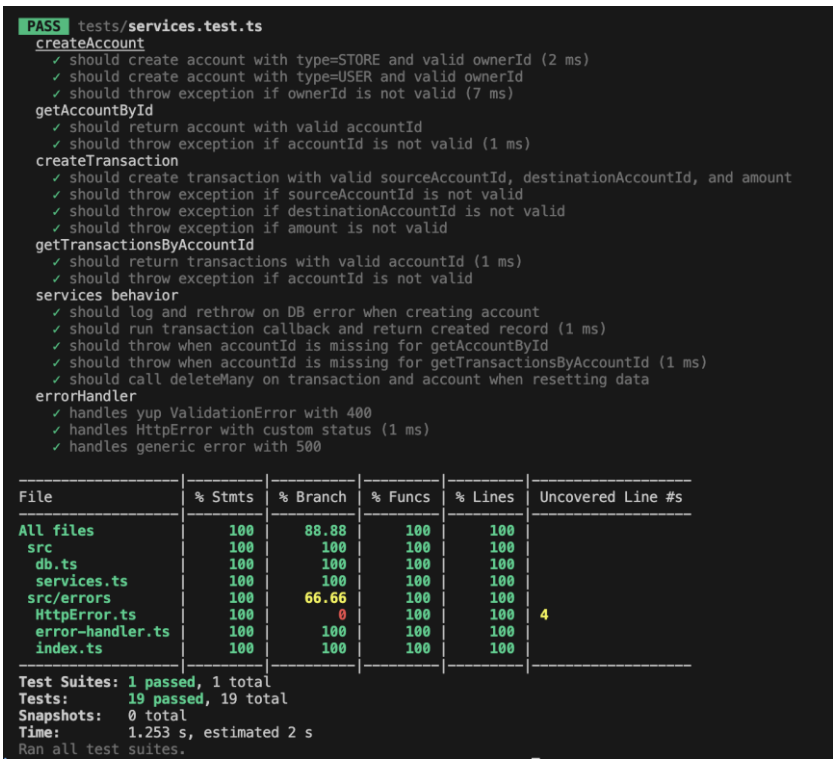


```
> fitit-cloud-functions@0.1.0 test
> jest

 PASS  store/__tests__/productReturns.test.js
 PASS  common/__tests__/productReturns.test.js
 PASS  common/__tests__/pubsub.test.js
-----------------------------|---------|----------|---------|---------
File                         | % Stmts | % Branch | % Funcs | % Lines
-----------------------------|---------|----------|---------|---------
All files                    |   99.14 |    84.82 |     100 |   99.54
 common/productReturns       |   98.83 |    83.67 |     100 |   99.36
  createProductReturn.js     |     100 |    85.71 |     100 |     100
  fetchProductReturnById.js  |     100 |       80 |     100 |     100
  fetchProductReturns.js     |   96.07 |       80 |     100 |   97.72
  index.js                   |     100 |      100 |     100 |     100
 common/pubsub               |     100 |    83.33 |     100 |     100
  publisher.js               |     100 |      100 |     100 |     100
  subscriber.js              |     100 |       75 |     100 |     100
 store/productReturns        |     100 |      100 |     100 |     100
  approveProductReturn.js    |     100 |      100 |     100 |     100
  index.js                   |     100 |      100 |     100 |     100
  rejectProductReturn.js     |     100 |      100 |     100 |     100
-----------------------------|---------|----------|---------|---------

Test Suites: 3 passed, 3 total
Tests:       45 passed, 45 total
Snapshots:   0 total
Time:        3.25 s, estimated 4 s
Ran all test suites.
```

Ilustración 57 - Porcentaje de cobertura de *Cloud Functions*



```
 PASS  tests/services.test.ts
  createAccount
    ✓ should create account with type=STORE and valid ownerId (2 ms)
    ✓ should create account with type=USER and valid ownerId
    ✓ should throw exception if ownerId is not valid (7 ms)
  getAccountById
    ✓ should return account with valid accountId
    ✓ should throw exception if accountId is not valid (1 ms)
  createTransaction
    ✓ should create transaction with valid sourceAccountId, destinationAccountId, and amount
    ✓ should throw exception if sourceAccountId is not valid
    ✓ should throw exception if destinationAccountId is not valid
    ✓ should throw exception if amount is not valid
  getTransactionsByAccountId
    ✓ should return transactions with valid accountId (1 ms)
    ✓ should throw exception if accountId is not valid
  services behavior
    ✓ should log and rethrow on DB error when creating account
    ✓ should run transaction callback and return created record (1 ms)
    ✓ should throw when accountId is missing for getAccountById
    ✓ should throw when accountId is missing for getTransactionsByAccountId (1 ms)
    ✓ should call deleteMany on transaction and account when resetting data
  errorHandler
    ✓ handles yup ValidationError with 400
    ✓ handles HttpError with custom status (1 ms)
    ✓ handles generic error with 500

------------------|---------|----------|---------|---------|-------------------
File              | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
------------------|---------|----------|---------|---------|-------------------
All files         |     100 |    88.88 |     100 |     100 |
 src              |     100 |      100 |     100 |     100 |
  db.ts           |     100 |      100 |     100 |     100 |
  services.ts     |     100 |      100 |     100 |     100 |
 src/errors       |     100 |    66.66 |     100 |     100 |
  HttpError.ts    |     100 |        0 |     100 |     100 | 4
  error-handler.ts|     100 |      100 |     100 |     100 |
  index.ts        |     100 |      100 |     100 |     100 |
------------------|---------|----------|---------|---------|-------------------
Test Suites: 1 passed, 1 total
Tests:       19 passed, 19 total
Snapshots:   0 total
Time:        1.253 s, estimated 2 s
Ran all test suites.
```

Ilustración 58 - Porcentaje de cobertura de Credits API

El resultado final alcanzado fue de **99,5 %** de cobertura global promedio, calculado a partir de los reportes previamente ilustrados, cumpliendo de esta forma con el **RNF-MAN Mantenibilidad**.

Las pruebas automáticas fueron integradas en el flujo de CI/CD mediante GitHub Actions, lo cual se detalla en profundidad en la sección 9.3.5. Integración continua.

8.3.2.2. Pruebas funcionales

El objetivo de las pruebas funcionales fue comprobar que el sistema cumpliera con los requerimientos definidos y que los distintos flujos de uso se comporten conforme a lo esperado. Este tipo de pruebas se realizaron siguiendo un enfoque de caja negra, donde no se evaluó el código ni la lógica interna del sistema, sino el comportamiento observable a partir de los datos de entrada y los resultados obtenidos [48].

Estas pruebas se centraron en los dos perfiles más relevantes dentro del flujo del sistema: **usuarios finales** y **tiendas**. Los casos de prueba para estos actores están especificados en el Anexo 12.18. Casos de pruebas funcionales, donde se detallan los pasos, el contexto y los resultados esperados para cada uno de los perfiles evaluados.

En cuanto a *backoffice*, el equipo decidió no incluirlo dentro del alcance de las pruebas funcionales, dado que se trata de una interfaz de uso restringido al equipo técnico de Fitit. Al no estar expuesto a usuarios externos, se consideró que el riesgo asociado era bajo y que los esfuerzos de prueba debían concentrarse en las áreas de mayor impacto funcional.

Todas las pruebas funcionales se realizaron en el entorno de *staging* (pre-producción) luego de cada *release*, verificando las nuevas funcionalidades incorporadas. Los resultados fueron documentados y, en cada iteración, permitieron confirmar que los principales flujos operativos del sistema se mantenían estables, detectando y corrigiendo defectos menores antes de cada entrega.

8.3.2.3. Pruebas de carga con k6

Para validar el cumplimiento del requerimiento no funcional de *performance*, se realizaron pruebas de carga sobre el entorno pre-productivo utilizando **k6**, con el objetivo de evaluar la estabilidad y capacidad de respuesta del sistema ante múltiples solicitudes concurrentes [49]. Las pruebas se centraron en las *Cloud Functions* críticas del flujo de devoluciones, verificando la eficiencia del escalado automático de Firebase y la correcta comunicación con la base de datos.

Los resultados mostraron una **latencia promedio de 18 ms**, con el 95 % de las respuestas resueltas en menos de 52 ms y un índice de error de apenas 0,33 %, lo que evidencia una respuesta rápida y consistente incluso bajo carga simultánea. Estas métricas confirman que el sistema soporta adecuadamente escenarios de uso intensivo sin degradar su rendimiento ni comprometer la disponibilidad de los servicios.

El comportamiento observado demuestra que el sistema cumple con el requerimiento **RNF-PERF *Performance***, garantizando tiempos de respuesta adecuados y un

funcionamiento estable en condiciones de carga realistas. La evidencia detallada de estas pruebas se presenta en el Anexo 12.19. Pruebas de carga con k6.

8.3.2.4. Pruebas E2E

Las pruebas *end-to-end* (*E2E*) y de usabilidad tuvieron como objetivo garantizar que el sistema Fitit Returns funcione correctamente a lo largo de los flujos reales de uso y que la experiencia del usuario sea fluida, comprensible y coherente con las expectativas de cada tipo de actor.

El equipo desarrolló pruebas *E2E* automatizadas utilizando **Cypress**, enfocadas en validar flujos completos tanto en el módulo fitit-user (usuarios finales) como en fitit-store (tiendas). En el primero, se probaron escenarios completos de devolución y cambio: inicio de sesión, selección de productos, carga de motivos, envío de evidencias fotográficas y confirmación del flujo. En el módulo de tienda, se validaron los flujos de aprobación o rechazo de devoluciones funcionaran correctamente dentro del panel de administración.

Si bien estas pruebas se integraron al flujo local de desarrollo, no se configuraron de forma automatizada vía GitHub Actions debido a limitaciones de tiempo y recursos. En su lugar, las pruebas *E2E* se ejecutaron manualmente en el entorno local previo a cada *release*. (Ver Anexo 12.26. Evidencia de pruebas en Cypress)

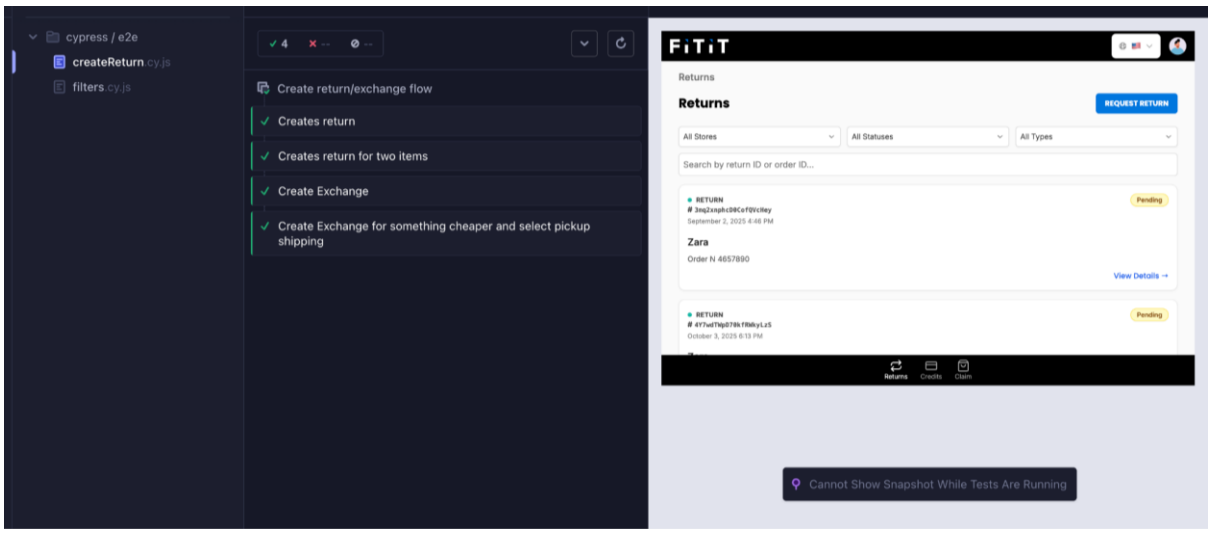


Ilustración 59 - Pruebas del flujo de usuario final en Cypress

8.3.2.5. Pruebas de usabilidad

El equipo realizó pruebas de usabilidad con usuarios reales que representaban los tres perfiles principales del sistema: usuarios finales, administradores de tienda y miembros del equipo interno de Fitit. Estas pruebas se efectuaron tras el primer *release* y en cada iteración posterior, combinando sesiones presenciales controladas con pruebas remotas.

Para evaluar el sistema con personas externas, sin conocimiento previo del producto, se utilizó **UserBrain**, lo que permitió observar comportamientos espontáneos y medir la comprensión de las interfaces sin sesgo de familiaridad. Los registros y resultados completos pueden consultarse en el Anexo 12.20. Pruebas de usabilidad.

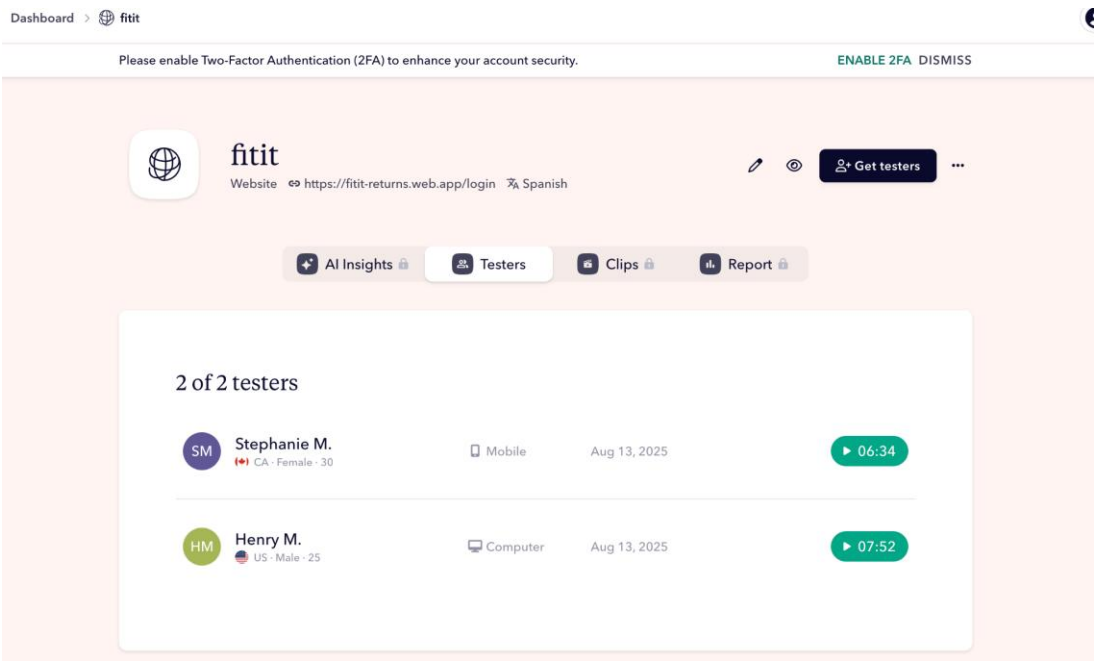


Ilustración 60 - Pruebas en UserBrain

Las sesiones con usuarios finales abarcaron el flujo completo de devolución y cambio en la web: iniciar el proceso, seleccionar ítems elegibles, indicar motivo, subir fotos/comentarios, elegir la resolución (devolución o cambio) y confirmar. Los participantes completaron todo sin asistencia y por debajo de los objetivos definidos:

- **Iniciar devolución: 12 s** (objetivo ≤ 20 s)
- **Completar formulario: 26 s** (objetivo ≤ 30 s)
- **Elegir resolución: 5 s** (objetivo ≤ 10 s)
- **Confirmar/cerrar: rápido y sin incidencias** (validaciones guiaron el paso final)

Los errores observados fueron menores (por ejemplo: intentar avanzar sin motivo o sin método de envío) y fueron interceptados por los mensajes de validación. Con base en estas observaciones se ajustaron textos y jerarquías visuales para reforzar la visibilidad del paso activo y del estado de la solicitud, especialmente en *mobile*.

En el panel de tienda, las pruebas se enfocaron en revisar solicitudes pendientes, aprobar/rechazar y en la gestión de políticas. Las personas participantes operaron sin asistencia y con pocos clics, y tras los ajustes de interfaz se observaron mejoras en tiempo:

Acción	Tiempo antes (s)	Tiempo después (s)	Mejora (%)

Ver y filtrar solicitudes	11	8	27,3 %
Aprobar una devolución válida	27	22	18,5 %
Rechazar una solicitud	24	19	20,8 %

Tabla 10 - Comparación de las pruebas de usabilidad luego de mejoras

Los cambios que explican estas mejoras incluyen filtros rápidos (por usuario/fecha/estado) y confirmaciones más visibles al aprobar con crédito o rechazar.

El *backoffice*, orientado al *CRUD* de entidades del sistema y visualización, no se incluyó en las pruebas de usabilidad por ser una interfaz de uso restringido. En su lugar, se validó funcionalmente por el equipo y Fitit durante cada *release*. Además que fue inspirado en su *backoffice* actual para garantizar compatibilidad y una posible futura integración.

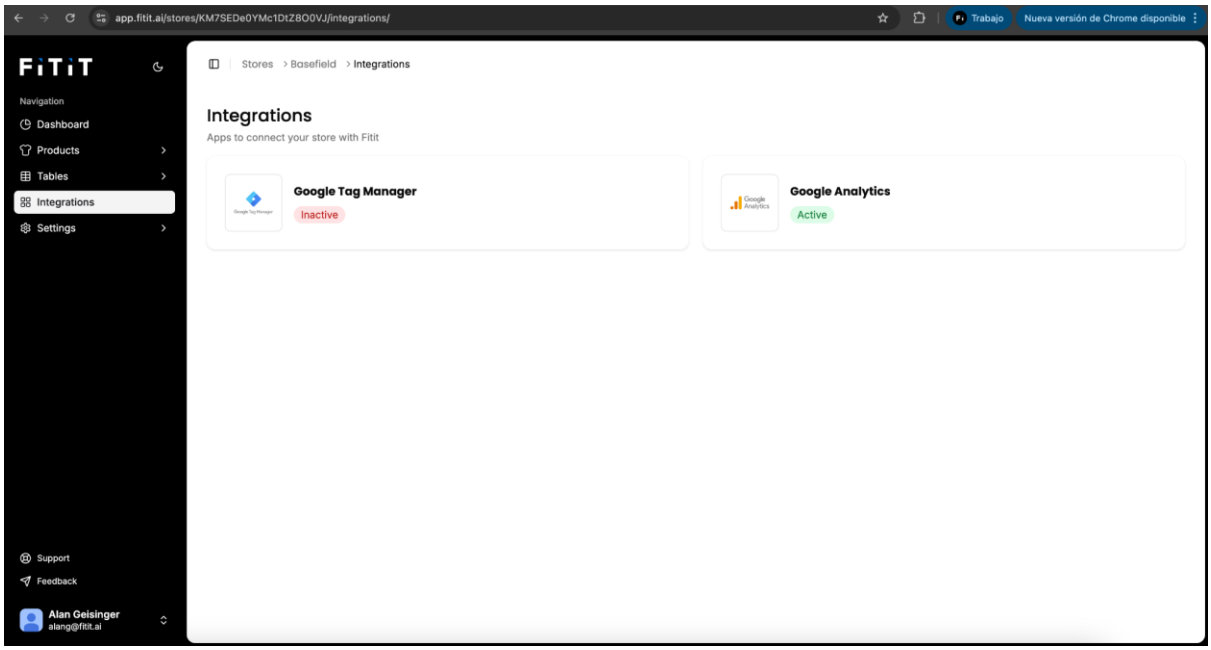


Ilustración 61 - *Backoffice* actual de Fitit

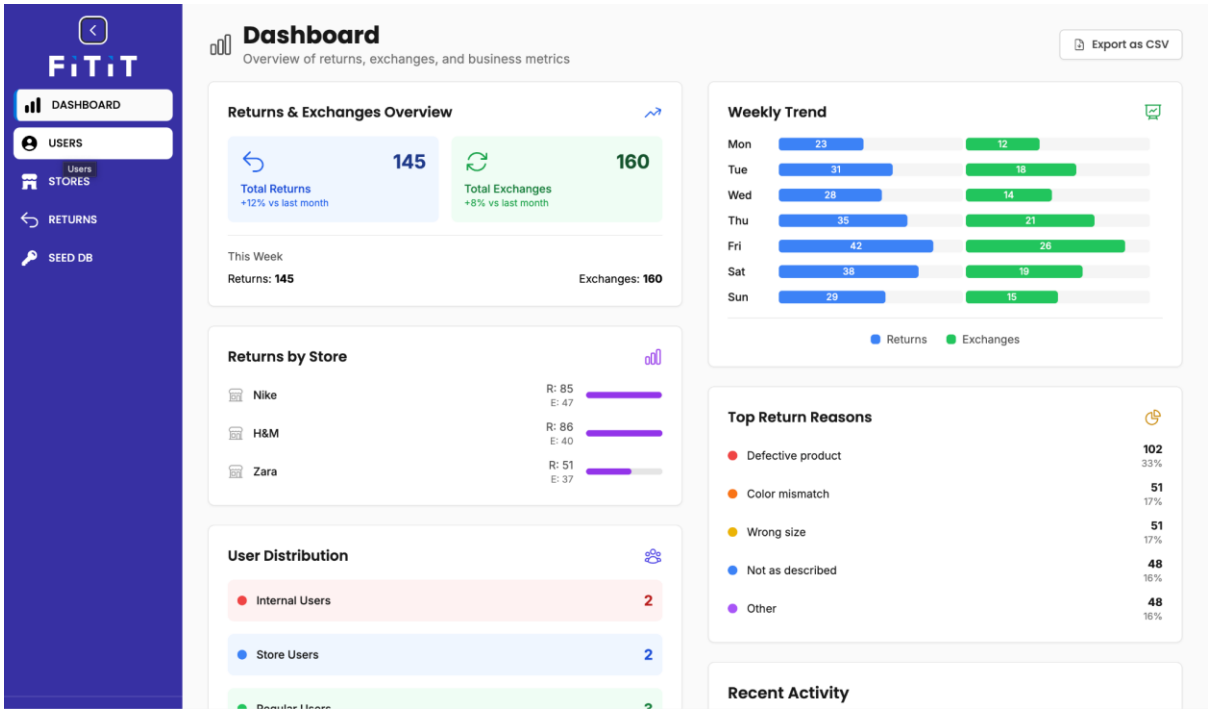


Ilustración 62 - *Backoffice* de nuestra solución

En total participaron 15 usuarios finales y 3 empleados de tienda, de cuyas sesiones se extrajeron hallazgos accionables. Además, las pruebas realizadas mediante UserBrain aportaron evidencia adicional desde usuarios externos, permitiendo contrastar los resultados con personas sin conocimiento previo del sistema, las cuales se ven en el Anexo 12.20.3. Sesiones externas (UserBrain).

Los resultados demostraron el cumplimiento del **RNF-US Usabilidad**. En particular, se confirmó el objetivo de completar el formulario de devolución en ≤ 120s bajo condiciones normales de uso. Las mejoras derivadas de estas pruebas, como visibilidad del estado y ajustes visuales en *mobile,* se implementaron en los siguientes *sprints*.

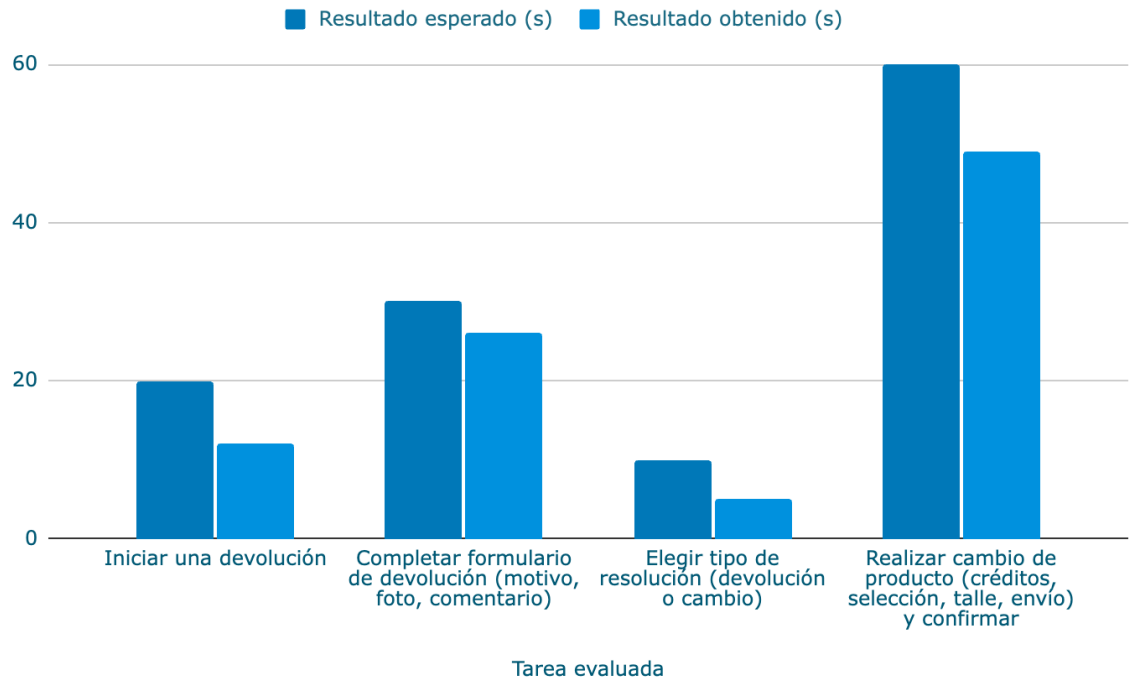


Ilustración 63 - Comparación entre el tiempo esperado y el obtenido (en segundos) para cada tarea.

8.3.3. Gestión de *bugs*

En todo proyecto de *software*, la aparición de defectos o errores, conocidos como ***bugs***, es un aspecto inherente al proceso de desarrollo. En nuestro caso, el equipo definió un proceso formal y estandarizado para la gestión de incidentes, con el objetivo de garantizar una rápida detección, análisis, priorización y resolución de los problemas que afectan la funcionalidad o experiencia de uso del sistema.

Dado que el producto implicaba la interacción simultánea de múltiples módulos se preveía la posibilidad de que los defectos pudieran originarse en distintas capas del sistema. Por esta razón, se estableció un protocolo uniforme para el tratamiento de todos los defectos, independientemente de su origen.

Los errores detectados durante los *code reviews* o antes del *release* no se contabilizaron como defectos, ya que fueron considerados parte del propio proceso de desarrollo de la *feature*. Solo aquellos que surgieron tras el *release* o durante las pruebas de validación fueron registrados como incidentes formales.

8.3.3.1. Registro y categorización de *bugs*

El registro y seguimiento de los defectos se realizó utilizando Jira, seleccionando el tipo de **tarea Bug** dentro del tablero general del proyecto. Cada defecto reportado era documentado con un título descriptivo, una descripción detallada y, cuando correspondía, los pasos para reproducir el error acompañados de capturas de pantalla, prioridad, responsable asignado, etiquetas, y la épica correspondiente.

Los incidentes podían ser reportados por distintos actores:

- **El equipo de desarrollo** a partir de pruebas internas.
- **El cliente** (equipo de Fitit), tras la ejecución de pruebas de aceptación o revisiones funcionales.
- **Usuarios finales y administradores de tiendas**, principalmente tiendas que participaron de la prueba piloto o usuarios que simularon devoluciones reales.

En algunos casos, el cliente o los usuarios reportaron defectos directamente por canales informales (por ejemplo, WhatsApp o email). Luego, el equipo formalizaba estos reportes en Jira, donde se completaban los campos mencionados anteriormente.

Un ejemplo completo del formato de registro de defectos se presenta en el Anexo 12.21. Formato de *bug* en jira.

8.3.3.2. Resolución de *bugs*

Los defectos se trataron como tareas independientes de las historias de usuario. Una vez registrados en el *backlog*, el equipo asignaba la prioridad y responsable, y decidía si el defecto se abordaría en el *sprint* en curso o en el siguiente. En casos urgentes, el equipo incorporaba la corrección directamente al *sprint* activo, incluso si eso implicaba una re-planificación menor.

Cada corrección se desarrollaba en una rama separada del repositorio, con *pull requests* que incluían la referencia a la tarea en Jira y una descripción del cambio. Antes de fusionarse en la rama principal, otro integrante del equipo revisaba el código.

Tras resolverse el defecto, se ejecutaban nuevamente las pruebas unitarias y funcionales para confirmar la corrección y evitar regresiones. Finalmente, se actualizaba el estado de la tarjeta en Jira.

El detalle de los *bugs* corregidos por sprint, junto con su prioridad, módulo afectado y responsable, se encuentra en el Anexo 12.22. Detalle de *bugs* por iteración.

8.3.4. Revisiones

El equipo de Fitit Returns planificó instancias de revisión continua para garantizar la calidad del código y de los requerimientos.

1. **Revisiones internas**:
    - **De requerimientos:** al inicio de cada iteración, se revisaban los requisitos y criterios de aceptación definidos en las tareas para asegurar su claridad y viabilidad.
    - **De código:** todos los *pull requests* fueron revisados por otro integrante del equipo antes de hacer *merge*, verificando estilo, buenas prácticas y cumplimiento de estándares.

2. **Revisiones con Fitit, usuarios y expertos:**

   - **Revisiones con Fitit:** se mantuvieron reuniones luego de cada *release* con representantes de la empresa, quienes aportaron su conocimiento sobre los flujos operativos actuales, el funcionamiento del recomendador de talles y los puntos de mejora identificados en la gestión de devoluciones. Estas sesiones permitieron ajustar las funcionalidades y priorizar aquellas de mayor valor para la operación real de la empresa.

   - **Martín Ganon** (experto en usabilidad y experiencia de usuario): brindó recomendaciones sobre diseño centrado en el usuario, simplificación de flujos y comunicación visual dentro del sistema, contribuyendo a mejorar la accesibilidad y la curva de aprendizaje de la aplicación.

Ilustración 64 - Revisión con Martin Ganon

3. **Revisiones formales académicas:**

   - **Primera revisión:** centrada en la definición del problema, cliente y enfoque de desarrollo.

   - **Segunda revisión:** enfocada en los requerimientos, la gestión de riesgos, las políticas de calidad y la metodología de trabajo, ya con una *demo* funcional del primer *release.*

   - **Tercera revisión:** consistió en una presentación de todo el proceso de descubrimiento y desarrollo de la solución, con *demo* funcional de todo el sistema.

8.3.5. Validaciones

Además de las revisiones, las validaciones constituyeron una etapa clave para confirmar que el sistema desarrollado cumpliera con las necesidades reales del cliente y los usuarios finales. Estas validaciones se realizaron sobre los requerimientos funcionales, no funcionales y la arquitectura del sistema.

En este último caso, se verificó que las decisiones arquitectónicas aseguraran escalabilidad, mantenibilidad y bajo acoplamiento entre componentes (ver detalles en la sección 7.6.1. Versión Inicial).

En todas las sesiones se analizaron métricas cualitativas, como el tiempo promedio para completar una devolución, la comprensión de los mensajes del sistema y la percepción general de confianza. Las observaciones fueron positivas, y el equipo aprovechó estas instancias para ajustar detalles de diseño y microinteracciones (por ejemplo la visualización de estados o el texto de los mensajes).

Las validaciones confirmaron que Fitit Returns cumplía los objetivos establecidos: simplificar el proceso de devolución y cambio tanto para las tiendas como para los usuarios finales, manteniendo una experiencia ágil, clara y segura.

8.3.6. Métricas de calidad

El equipo definió un conjunto de métricas para evaluar tanto el producto como el proceso de desarrollo, con el objetivo de analizar su efectividad, identificar oportunidades de mejora y asegurar la calidad de la solución en cada *release*. Cada métrica se estableció con un objetivo cuantificable, una **frecuencia de medición** y un **indicador de cumplimiento**.

Las mediciones se realizaron antes de cada una de los cuatro *releases* del sistema, lo que permitió observar la evolución del producto y del proceso en función de los ajustes aplicados tras cada iteración.

8.3.6.1. Calidad del producto

Las métricas de calidad del producto permitieron evaluar la estabilidad del sistema, el cumplimiento de los requerimientos funcionales, la satisfacción del cliente y la usabilidad de la interfaz.

Satisfacción del cliente:

Se realizaron encuestas de satisfacción al finalizar cada *release* dirigidas al cliente, preguntando a los tres *founders* su satisfacción del 1 al 10 con el proyecto.

El objetivo fue mantener una puntuación de satisfacción de 8/10 o superior, asegurando una percepción positiva del sistema en términos de utilidad, claridad y diseño.

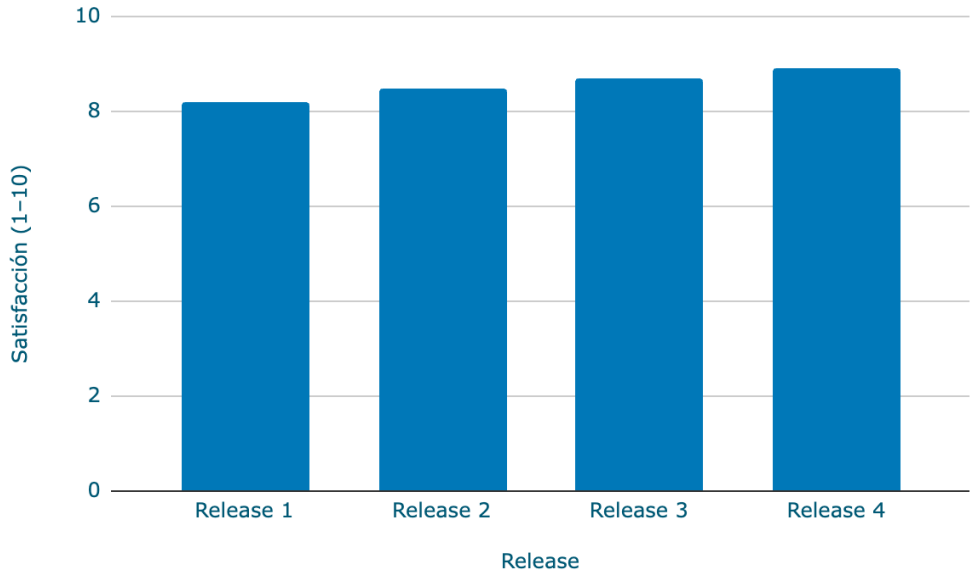


Ilustración 65 - Gráfica de satisfacción del cliente en los *releases*

Los resultados evidencian un aumento sostenido de la satisfacción, alcanzando **8.9/10 puntos en el *release* 4**, y cumpliendo ampliamente con el objetivo. Este crecimiento se atribuye a las mejoras en la interfaz, la reducción de *bugs* y la incorporación de funcionalidades valoradas por el cliente.

Errores críticos en releases:

Durante todo el ciclo de desarrollo, el sistema no presentó errores críticos en los *releases*, cumpliendo con el objetivo de tener **cero errores críticos**.

Este resultado fue posible gracias a la ejecución sistemática de pruebas funcionales, automatizadas y exploratorias antes de cada despliegue, así como al proceso de revisión por pares implementado en cada *pull request*.

La ausencia total de defectos críticos en el entorno pre-productivo refleja la madurez del proceso de validación, la efectividad de las pruebas realizadas y la solidez de la arquitectura del sistema.

***Bugs* encontrados vs. corregidos:**

Para evaluar la capacidad de respuesta del equipo ante incidencias, se analizó la relación entre defectos encontrados y corregidos.

El objetivo fue mantener una **tasa de resolución del 100 %**, garantizando que todos los defectos reportados fueran solucionados antes de avanzar a la siguiente versión.

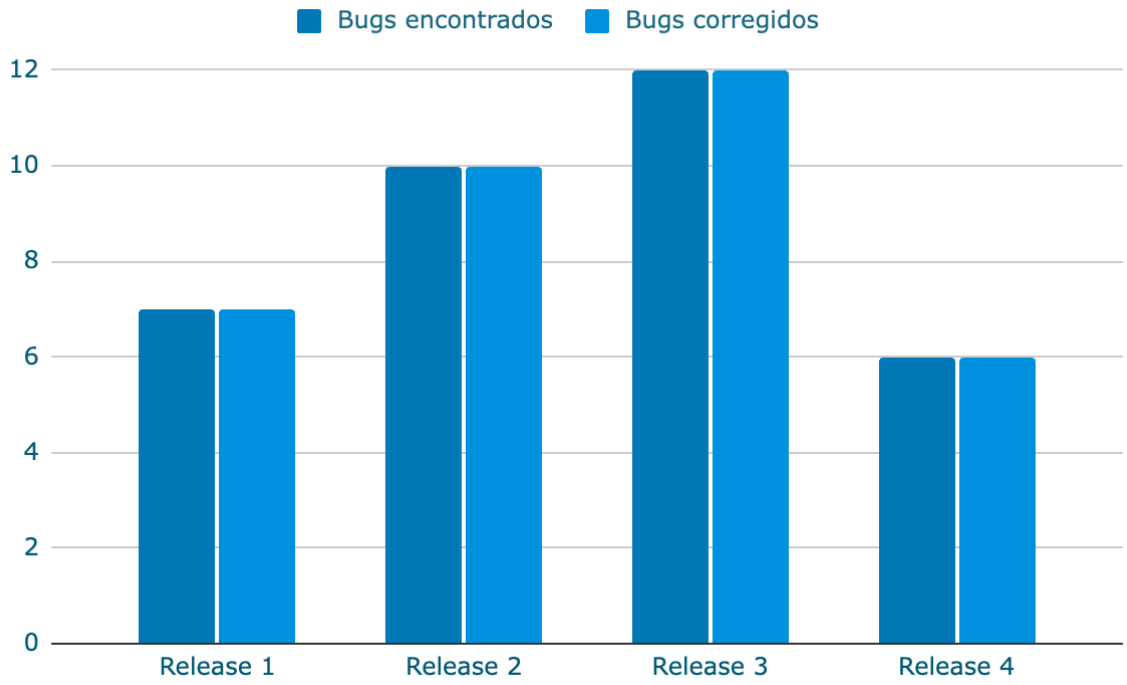


Ilustración 66 - Gráfica de *bugs* encontrados vs. corregidos en cada *release*

En todas las iteraciones se cumplió con el objetivo: 7/7 en el primer *release*, 10/10 en el segundo, 12/12 en el tercero y 6/6 en el cuarto, reflejando una gestión efectiva de incidencias y un control de calidad sostenido y resolviendo un total de 35 *bugs*.

Usabilidad basada en las Heurísticas de Nielsen:

El equipo realizó una verificación de la usabilidad a lo largo del proyecto, tomando como referencia las 10 Heurísticas de Nielsen [50]. El objetivo fue respetar y seguir las heurísticas para cumplir con los **RNF-US Usabilidad**.

En el Anexo 12.23. Aplicación de Heurísticas de Nielsen, se incluye la tabla de aplicación, que documenta cómo se materializa cada heurística en la UI con ejemplos concretos (componentes, mensajes, validaciones y patrones de navegación).

Ilustración 67 - Evidencia de botón para volver atrás

En concordancia con el principio de **Control y libertad del usuario** (Heurística de Nielsen), la Ilustración 67 muestra la implementación de un botón 'Atrás' persistente en los pasos intermedios del flujo, lo que garantiza que el usuario mantenga el control y la capacidad de deshacer acciones o salir del proceso en cualquier momento.

8.3.6.2. Calidad del proceso

Las métricas de proceso tuvieron como objetivo evaluar la eficiencia del trabajo en equipo, la gestión del retrabajo y los tiempos de respuesta frente a defectos e incidencias.

Retrabajo y corrección de incidencias
El retrabajo se midió como porcentaje de esfuerzo dedicado a correcciones y ajustes (no solo *bugs*), por lo que puede variar independientemente del conteo de defectos, permitió evaluar la estabilidad del desarrollo y la calidad de las implementaciones en cada *release*.

A medida que el equipo fue consolidando la estructura del sistema y fortaleciendo el proceso de revisión de código, el porcentaje de retrabajo disminuyó progresivamente, situándose en 11,2 % en el primer *release*, 8 % en el segundo, 7 % en el tercero y 6 % en la cuarto, cumpliendo finalmente con el objetivo planteado.

En paralelo, el tiempo medio de resolución de incidencias también mostró una evolución positiva. En el primer *release*, el promedio fue superior al esperado (alrededor de 2,3 días), principalmente por la necesidad de revisar flujos nuevos y ajustar integraciones con terceros.

No obstante, a partir del segundo *release* el tiempo de respuesta se estabilizó y mejoró de forma sostenida, alcanzando **1,5 días promedio** en el cuarto *release*, lo que evidencia una mayor eficiencia del proceso de corrección y priorización de tareas.

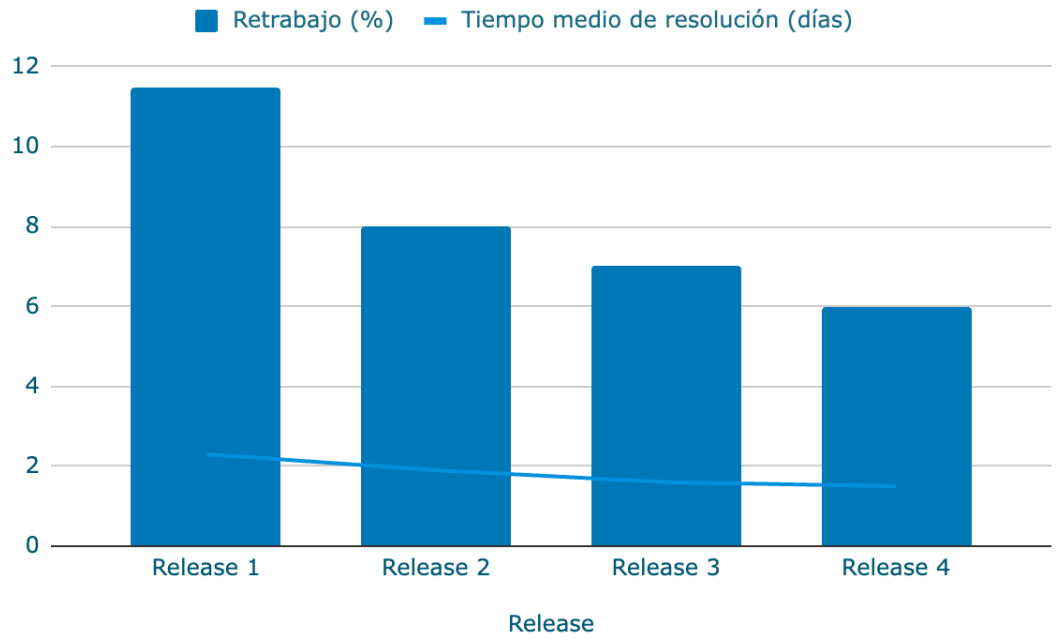


Ilustración 68 - Porcentaje de retrabajo y dias de resolución

Estos resultados demuestran que, aunque los objetivos no se cumplieron completamente en las primeras etapas, la implementación de revisiones por pares, pruebas automatizadas y planificación más precisa permitió alcanzar los niveles de calidad y eficiencia esperados en las etapas finales del proyecto.

8.3.6.3. Calidad del código

Las métricas de calidad del código se centraron en evaluar la mantenibilidad (**RNF-MAN Mantenibilidad**), la cobertura de pruebas y la correcta revisión de los *pull requests*.

Como se mencionó anteriormente en la sección 8.3.2.1. Pruebas automáticas, la cobertura de *tests* del *backend* tuvo como objetivo alcanzar un **mínimo del 90 %**. En el primer *release* fue de 91,57%, mejorando gradualmente hasta llegar a 99,5 % en la cuarta, cumpliendo el objetivo. Este crecimiento refleja la consolidación de una base de código estable, respaldada por un conjunto robusto de pruebas automatizadas.

Por otra parte, se mantuvo una revisión del 100 % de los *pull requests* a lo largo del proyecto, asegurando el cumplimiento de los estándares de codificación, la consistencia en el estilo y la validación de buenas prácticas antes de cada *merge*.

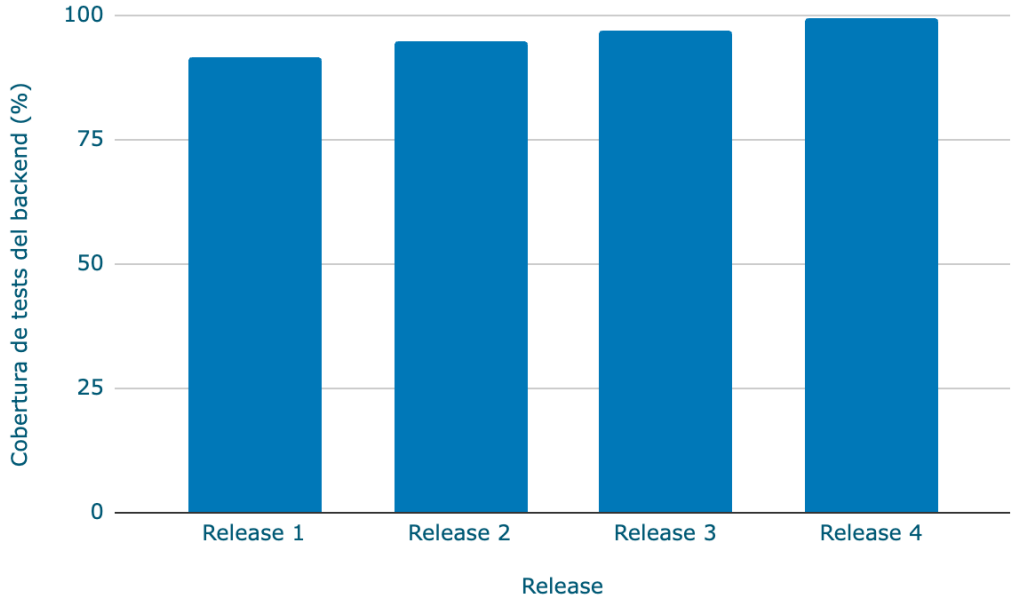


Ilustración 69 - Evolución de la cobertura de tests por *release*

Las métricas de calidad del proyecto Fitit Returns evidenciaron una mejora sostenida a lo largo de los cuatro *releases*. Se observó una reducción progresiva de defectos, un aumento en la cobertura de pruebas automatizadas y una disminución significativa del retrabajo, indicadores que reflejan la madurez alcanzada en la gestión del proceso.

La estrategia de aseguramiento de la calidad no solo garantizó un producto robusto y validado, sino que también fortaleció las prácticas del equipo y consolidó un marco de trabajo replicable para futuros proyectos.

8.4. Distribución del esfuerzo

En esta sección se analiza cómo se distribuyó el esfuerzo entre las actividades realizadas para asegurar la calidad del sistema, con el objetivo de observar qué impacto tuvo cada una en los resultados obtenidos. Las actividades se clasificaron en tres grandes categorías: prevención, evaluación y corrección, descritas a continuación.

En primer lugar, las **acciones de prevención** tuvieron como objetivo evitar la aparición de defectos desde las fases iniciales, incluyendo la definición temprana de estándares de codificación y estructura del repositorio, la validación de arquitectura y la configuración del entorno local. El equipo dedicó tiempo a familiarizarse con las tecnologías clave —Firebase, React.js, Next.js y Google Cloud Platform—, lo que facilitó el desarrollo posterior y redujo errores estructurales. Además, se implementaron revisiones de código en todos los *pull requests*, garantizando la consistencia, la seguridad y la mantenibilidad del código.

Las **actividades de evaluación** se centraron en verificar la calidad del sistema mediante pruebas unitarias y automáticas de *backend*, pruebas funcionales e integraciones entre *Cloud Functions*, y *tests* de interfaz en distintos dispositivos y

navegadores. También se realizaron evaluaciones de usabilidad, tanto internas como con usuarios reales y administradores de tiendas, que permitieron identificar oportunidades de mejora en la navegación, los textos y la comprensión del flujo de devolución.

Finalmente, las **actividades de corrección** abarcaron el retrabajo derivado de los *bugs* detectados en las pruebas o reportes externos. Cada incidencia fue documentada y gestionada en Jira, priorizada según su impacto y validada antes del despliegue. Este enfoque iterativo y continuo de mejora permitió reducir el retrabajo, mantener una alta cobertura de *tests* y asegurar que cada *release* cumpliera con los estándares de calidad establecidos.

8.4.1. Análisis general

Como se observa en la gráfica siguiente, la mayor proporción del esfuerzo se destinó a las **actividades de prevención**, seguida por las de evaluación y, en menor medida, las de corrección.

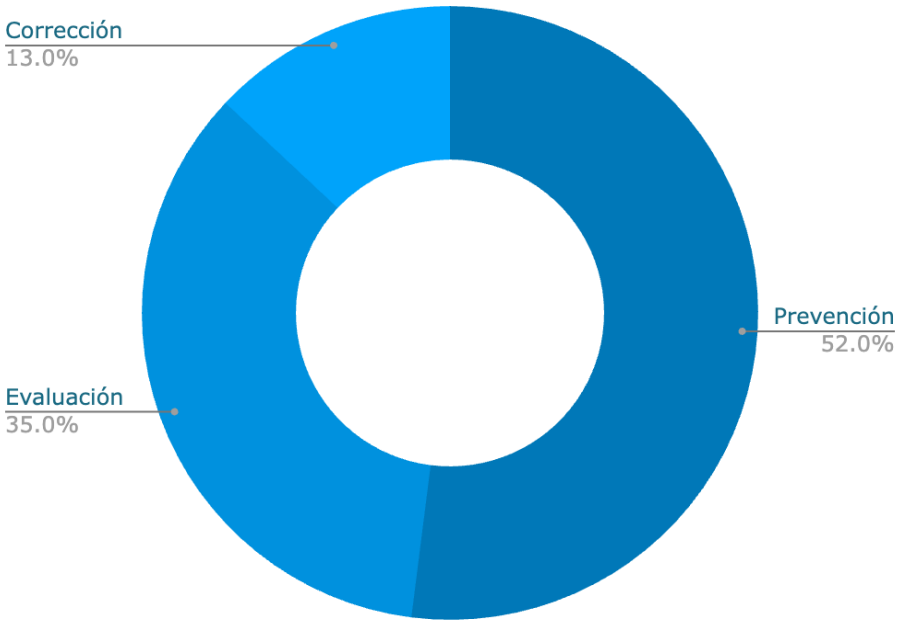


Ilustración 70 - Distribución de esfuerzo en calidad.

Esta distribución fue clave para mantener la calidad general del sistema. La **inversión en prevención**, especialmente en revisiones de código, familiarización técnica y definición de estándares, redujo significativamente la cantidad de defectos críticos.

El proceso de gestión de la calidad en Fitit Returns fue fundamental para consolidar un producto robusto, confiable y alineado con los objetivos del cliente y de la tesis. El uso de métricas objetivas brindó visibilidad sobre el progreso del proyecto, facilitando la toma de decisiones informadas y confirmando una mejora sostenida. La estrategia adoptada no solo garantizó la entrega de un producto libre de errores críticos y validado por los usuarios, sino que también fortaleció la madurez técnica y organizativa del equipo.

9. Gestión de la configuración

Una parte fundamental en cualquier proyecto de *software* es establecer herramientas y procesos que ayuden a gestionarlo de forma adecuada.

9.1. Identificación de elementos de la configuración

Se identificaron los siguientes elementos de la configuración, subdivididos en las siguientes categorías:

- Código
- Ciclo de vida
- Documentación

9.1.1. Código

Elemento	**Herramienta**
Código fuente	Git y GitHub
Pipeline de CI/CD	GitHub Actions

Tabla 11 - Herramientas de código

9.1.2. Metodologías

Elemento	**Herramienta**
Gestión del proyecto	Jira y ClickUp
Registro de esfuerzo	Clockify y ClickUp
Retrospectivas	Miro
Estimaciones	Planning Poker
Establecimiento de reuniones	Google Calendar
Reuniones	Google Meet
Comunicaciones informales	Whatsapp
Comunicaciones formales y con tutor	Slack

Tabla 12 - Herramientas de metodologías

9.1.3. Documentación

Elemento	**Herramienta**
Informes de reuniones y entrevistas (tutor, clientes, usuarios, etc.)	Google Docs, ClickUp y Notion
Cesión de derechos	Google Docs
Anteproyecto	Google Docs
Informes de avance	Google Docs
Documentación final	Google Docs
Design Thinking	Google Docs, Escalidraw y Miro
Gestión de Riesgos	Google Docs
Arquitectura	Draw.io y Escalidraw
Presentaciones	Google Slides
Diseños de baja fidelidad	Escalidraw y Prototipos en papel
Diseños de alta fidelidad	Figma

Tabla 13 - Herramientas de documentación

9.2. Herramientas de gestión

Se eligieron las herramientas que se consideraron más adecuadas para llevar adelante la gestión de los distintos elementos de configuración, garantizando una administración eficiente y alineada con los objetivos del proyecto.

9.2.1. Git y GitHub

La utilización de Git y GitHub resultó fundamental para la gestión del código y la coordinación del trabajo en equipo, ya que permitió mantener un control riguroso de versiones y asegurar la trazabilidad de los cambios. Asimismo, el uso de repositorios centralizados y *pull requests* favoreció la implementación de buenas prácticas de desarrollo, facilitando la revisión colaborativa y mejorando la calidad del *software* entregado.

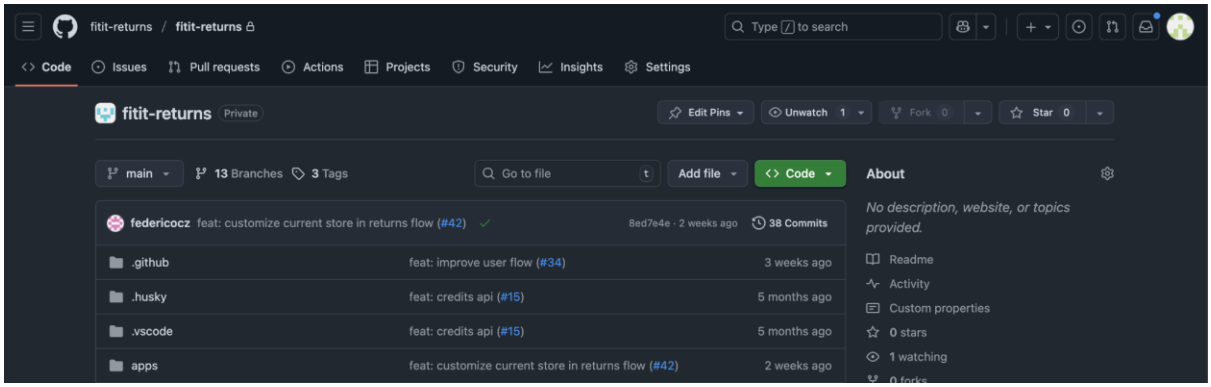


Ilustración 71 - Repositorio de GitHub privado

9.2.2. GitHub Actions

GitHub Actions es la herramienta de automatización de GitHub que permite configurar *pipelines* de integración y despliegue continuo (CI/CD). Se utilizó para ejecutar pruebas para validar y facilitar la entrega continua del *software*.

9.2.3. ClickUp

ClickUp es una plataforma de gestión de proyectos y productividad. Durante la etapa de descubrimiento se empleó para la organización del *backlog*, registro de horas y almacenamiento de documentación. Posteriormente se migró a herramientas más específicas como Jira y Clockify debido a limitaciones de la suscripción gratuita.

9.2.4. Notion

Notion es una plataforma multifuncional que el equipo utilizó para documentar dentro de ClickUp. Permitió generar un entorno colaborativo en el que la información se mantenía siempre actualizada. Gracias a su enfoque modular y su interfaz intuitiva, Notion se convirtió en un espacio que fomentó la transparencia y alineación entre los miembros del equipo a lo largo del proyecto.

9.2.5. Jira

Jira es una herramienta de Atlassian enfocada en la gestión ágil de proyectos. Se utilizó para planificar, priorizar y dar seguimiento a tareas, *bugs* y nuevas funcionalidades, adoptando los flujos de trabajo ya mencionados en el capítulo 5. Gestión de proyecto.

9.2.6. Clockify

Clockify es una herramienta de registro de tiempo que permite llevar un control detallado de las horas dedicadas por el equipo en cada tarea o proyecto. Se utilizó para medir esfuerzo y darle seguimiento a la carga de trabajo.

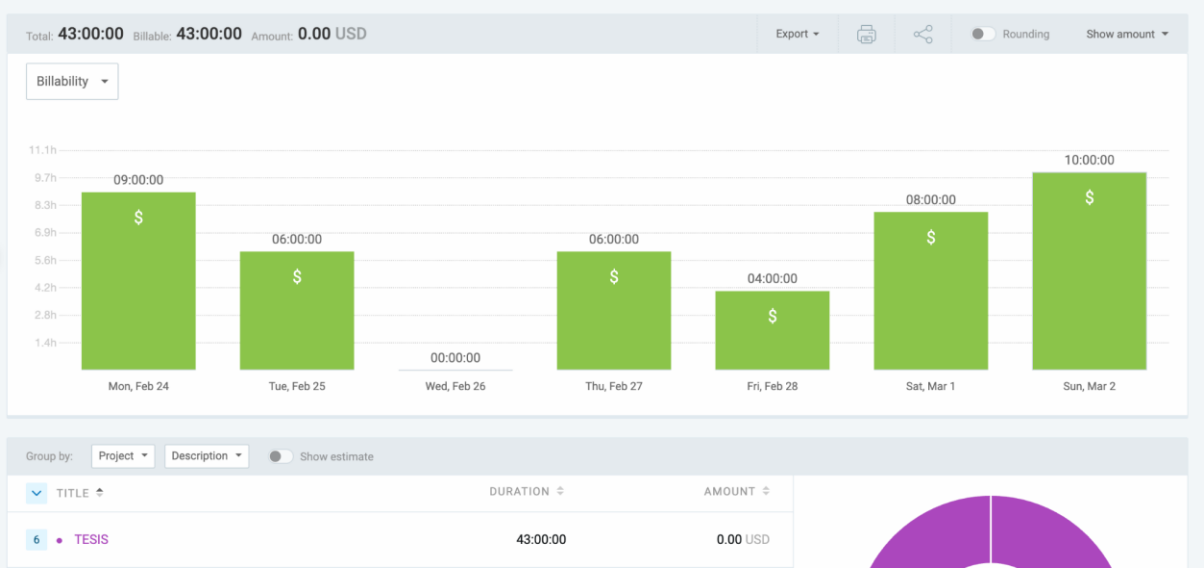


Ilustración 72 - Carga de horas en una semana en clockify

9.2.7. Google Docs

Google Docs es una herramienta de edición colaborativa de documentos en la nube. Se empleó para la redacción de informes, documentación técnica y entregables del proyecto, permitiendo la edición simultánea por múltiples integrantes del equipo, lo que fue muy beneficioso para garantizar la coherencia de la información, agilizar el trabajo colaborativo y reducir tiempos de entrega.

9.2.8. Google Slides

Google Slides se utilizó como herramienta de presentaciones colaborativas para preparar y exponer las revisiones de tesis en la facultad ORT.

9.2.9. Escalidraw

Escalidraw es una aplicación colaborativa de diagramación y creación de prototipos de baja fidelidad. Fue utilizada para distintas etapas de *Design Thinking*, para representar visualmente procesos y mockups de la aplicación y también para realizar diagramas de arquitectura en las revisiones oficiales en ORT (ver Anexo 12.27. Excalidraw).

9.2.10. Draw.io

Draw.io es una herramienta de diagramación orientada a representar arquitecturas de *software*, diagramas de flujo y modelos estructurados. Se utilizó para documentar la arquitectura del sistema y otros diagramas técnicos.

9.2.11. Figma

Figma es una herramienta de diseño colaborativo enfocada en interfaces gráficas. Se empleó para la creación de prototipos de alta fidelidad, facilitando la definición de la

experiencia de usuario y la validación temprana del diseño con Fitit. Estos se pueden ver en el Anexo 12.9.10. Prototipos de alta fidelidad.

9.2.12. WhatsApp

Desde el inicio del proyecto el equipo consideró fundamental tener un grupo exclusivo en WhatsApp para las comunicaciones rápidas entre los integrantes. Allí se compartieron avances diarios de forma breve, dudas puntuales y recordatorios. Este espacio permitió mantener la fluidez sin necesidad de agendar reuniones formales, además de ser útil para acordar horarios y resolver bloqueos en el momento.

9.2.13. Slack

Slack fue el canal de mensajería elegido para centralizar la comunicación vinculada a la gestión del proyecto y comunicaciones con el tutor. Se utilizaron distintos canales temáticos lo que permitió organizar mejor las conversaciones.

9.2.14. Google Meet

Google Meet fue la plataforma utilizada para las reuniones virtuales. Se llevaron a cabo por este medio las bi-weekly meetings internas, las *sprint plannings*, las *retrospectives* y las *reviews*. Además, resultó clave para las reuniones con el tutor y para los encuentros con el cliente cuando no podían hacerse de manera presencial.

Ilustración 73 - Reunión con tutor

9.3. Gestión del repositorio

El repositorio del proyecto se encuentra alojado en la organización privada de Fitit.

9.3.1. *Monorepo*

Se decidió utilizar un enfoque de *monorepo* mediante NPM Workspaces, en contraste con la gestión más común basada en múltiples repositorios. Esta decisión respondió a la necesidad de aumentar la velocidad en el desarrollo y facilitar la gestión de dependencias. A continuación se detalla la comparación que se realizó a la hora de tomar esta decisión:

Criterio	***Monorepo***	**Multirepo**
1. Reutilización de código	Alta porque librerías compartidas se referencian directo	Baja porque requiere publicación/versionado en *registries*
2. Gestión de dependencias	Centralizada, pero puede crecer en complejidad	Independiente en cada repo, puede generar duplicaciones
3. Cambios transversales	Sencillos, un único *PR* modifica todos los módulos afectados	Complejos, requieren *PRs* coordinados en múltiples repos
4. CI/CD	Requiere *pipelines* inteligentes para ejecutar solo lo necesario	*Pipelines* simples, cada repo solo *build*/*testea* su propio código
5. Consistencia técnica	Fácil mantener mismas reglas de *lint*, *test* y CI/CD	Difícil mantener homogeneidad entre repos
6. Escalabilidad técnica	Puede volverse pesado (ejemplo: clones lentos, *pipelines* costosos, etc.)	Ligero por repo, más simple clonar y navegar
7. Autonomía de equipos	Menor porque todos comparten reglas y *releases*	Mayor porque cada equipo define su propio flujo y versionado
8. Control de acceso	Granularidad baja (todos tienen acceso al mismo repo)	Granularidad alta (permisos gestionados repo por repo)

Tabla 14 - Comparación de *Monorepo* vs. Multirepo

Se resolvió la problemática de *pipelines* inteligentes (punto 4) usando los *workflow action paths,* una serie de pasos automatizados para la ejecución de tareas:

```yaml
name: 'Build - User'      Nicolás Andreoli Sandler, 9 months

on:
  push:
    branches:
      - main
    paths:
      - 'shared/web/**'
      - 'apps/fitit-user/**'
  pull_request:
    types: [opened, synchronize, reopened, ready_for_review]
    paths:
      - 'shared/web/**'
      - 'apps/fitit-user/**'
```

Ilustración 74 - Extracto de la action *build-user.yaml*

En este ejemplo se ve como el *Build* de la aplicación se ejecuta únicamente si hay cambios en la librería @shared/web o en el código fuente de la app fitit-user. Si bien hay que configurarlo manualmente, la complejidad de hacerlo es relativamente baja.

Como la aplicación es un *MVP* básico, el tamaño del repo y sus dependencias no va a crecer al punto de que no escale técnicamente en entornos locales (punto 6). Las últimas dos puntos 7 y 8, no aplican porque el equipo es de 3 personas. Estos serían factores interesantes a considerar en una empresa de mayor escala.

Todos los demás puntos que favorecen el uso de *Monorepo* (puntos 1, 2, 3 y 5). Por todas estas razones, el *tradeoff* resultó a favor de esta última estrategia de organización de código [51].

9.3.2. Gestión de dependencias

La gestión de dependencias se simplifica significativamente al utilizar un *monorepo*, ya que todas las dependencias se administran desde un solo *package-lock.json* (ver extracto de Anexo 12.24.1. Extracto de package.json), que actúa como la única fuente de la verdad. Este enfoque garantiza que todas las aplicaciones utilicen versiones idénticas de las librerías, minimizando conflictos y facilitando futuras actualizaciones.

9.3.3. Estrategia de *branching*

Para la gestión de ramas se utilizó *GitHub Flow*:

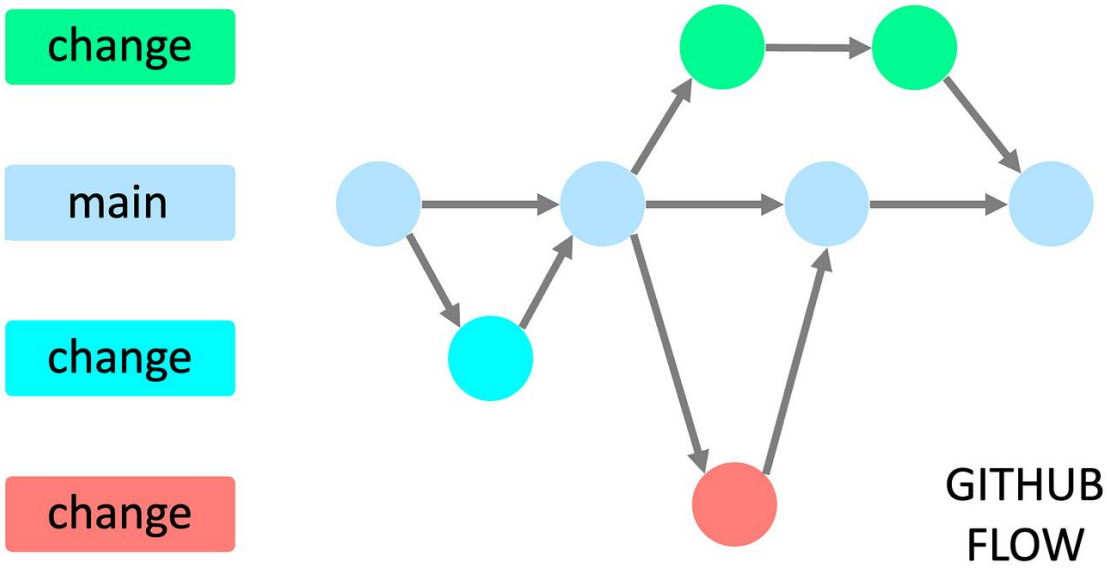


Ilustración 75 - Diagrama explicativo de GitHub Flow

Esta estrategia consta de una rama principal *main* de la cual derivan todas las demás ramas de desarrollo. Para garantizar consistencia en el nombre de las *branches*, se implementó un sistema de linteo mediante la herramienta *branchlint*, con el objetivo de estandarizar la nomenclatura y mantener claridad en la gestión del código. La configuración permite únicamente los prefijos *feature*, *fix*, *hotfix* y *release*, con un máximo de dos secciones separadas por "/" (ver Anexo 12.17. Configuración y aplicación de estándares).

Se evaluó la alternativa de *Git Flow*, pero se priorizó *GitHub Flow* porque se ajustaba mejor a las necesidades de entrega ágil del proyecto. Esta estrategia de *branching* se complementa con un flujo de integración continua mediante GitHub Actions, que asegura la calidad y robustez del código incorporado, con el fin de tomar todas las medidas posibles para que el código que llegue a producción sea lo más fiable posible [52].

9.3.4. *Branch protection rules*

El repositorio cuenta con *Branch protection rules* que refuerzan las buenas prácticas de desarrollo:

- Se permite únicamente el *merge* mediante *Squash.*
- Se requiere al menos una aprobación por *Pull Request* de una persona distinta a la que la generó.
- Se requiere que todas las GitHub Actions se completen exitosamente
- Se prohíbe *force* al pushear.

Esta configuración garantiza que cualquier cambio incorporado a la rama principal cumpla con los estándares de calidad definidos.

9.3.5. Integración continua

La integración continua se estructura en tres grandes grupos de acciones:

- *Builders*
- *Linters*
- *Testing*

Estas acciones se ejecutan automáticamente ante cada *push* o *pull request* abierto. El objetivo principal de este flujo es mantener la integridad del código y minimizar la posibilidad de introducir errores en la rama principal, promoviendo un desarrollo más eficiente y confiable.

Para correr las acciones de *Build*, fue necesario utilizar otra *feature* de GitHub conocida como *GitHub Action Secrets*. Esta herramienta permite setear secretos que solamente las acciones tienen que interpretar, inclusive se puede setear distintos secretos según el ambiente.

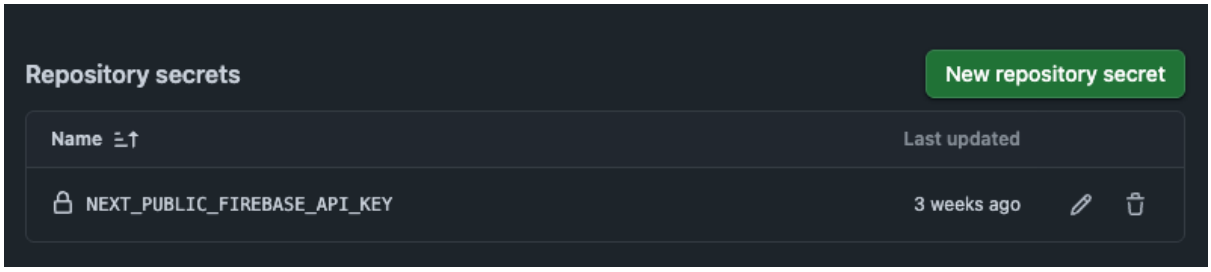


Ilustración 76 - Evidencia del uso de GitHub Secrets

La evidencia de la ejecución y el detalle de todas las GitHub Actions se encuentra en el Anexo 12.24.2. Github Actions.

9.4. Gestión de versiones

Para la gestión de versiones se adoptó una nomenclatura simple basada en *releases* numeradas:

- *release-1*
- *release-2*
- *release-3*
- *release-4*

Aunque se consideró el uso del estándar semántico de versionado *major.minor.patch* [53], se decidió optar por un esquema más directo y simple.

Cada *release* cuenta con un *changelog* que resume los *pull requests* incorporados, dando visibilidad y claridad sobre los cambios que tiene incluído. La utilización de la acción *pr-name-linter* permitió estandarizar los nombres de los *PRs*, aplicando la

convención universal de prefijos *feat*, *fix*, *chore*, etc. Esto contribuye a mantener una documentación clara y uniforme, además de mejorar la legibilidad del historial de versiones.

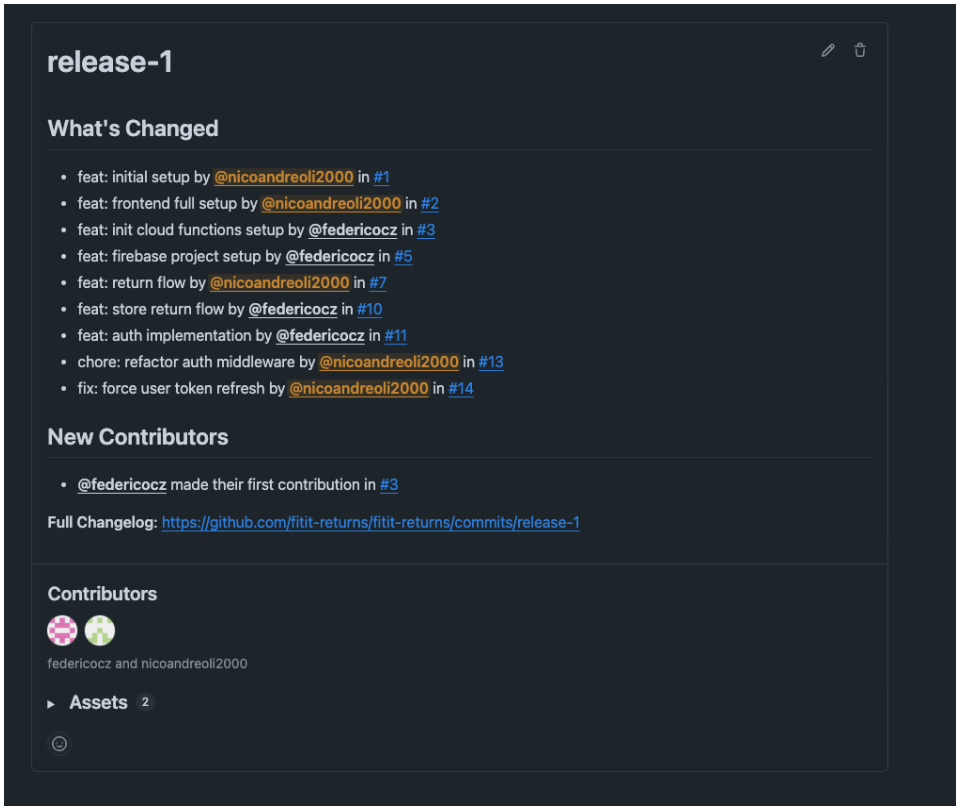


Ilustración 77 - Evidencia de los *releases*

10. Conclusiones

10.1. Estado actual

Al 16 de octubre de 2025, Fitit Returns se encuentra completamente desarrollado, documentado y desplegado en un entorno de pre-productivo (*staging*) de GCP y AWS. El sistema opera como una solución modular sustentada en un *monorepo* que organiza múltiples paquetes y aplicaciones especializadas.

En el frente funcional, la plataforma se articula en tres interfaces web:

- **Aplicación de usuarios finales (fitit-user):** permite iniciar solicitudes de devolución o cambio, adjuntar evidencia (texto e imágenes) y consultar el estado en tiempo real a lo largo del proceso.

- **Aplicación de gestión de tiendas (fitit-store):** posibilita a los administradores revisar, aprobar o rechazar solicitudes, aplicar políticas de devolución y emitir créditos según reglas definidas.

- **Aplicación de backoffice interno (fitit-backoffice):** ofrece visibilidad y control centralizado para el equipo de Fitit, incluyendo supervisión de operaciones, revisión de registros y gestión de usuarios y tiendas.

A nivel de entrega y operación, el código fuente reside en un **repositorio** de GitHub, al que Fitit ya tiene acceso, con instrucciones de despliegue y configuraciones diferenciadas por ambiente (local, *staging* y eventualmente producción). La documentación técnica —incluyendo guía de instalación, configuración de entornos y estructura del *monorepo*— acompaña la entrega y permite la puesta en producción de manera trazable y reproducible.

En calidad, el sistema se consolidó con una batería de **pruebas** que combina unitarias, integración, *E2E* y carga. Entre los resultados relevantes, las pruebas de *performance* con k6 mostraron latencia promedio baja dentro de los umbrales definidos, las *E2E* con Cypress validaron flujos completos de usuario y tienda sobre los módulos críticos, y las pruebas de usabilidad con perfiles representativos contribuyeron a mejorar la claridad del flujo de uso y la comprensión de las interfaces.

10.2. Sobre los objetivos

En esta sección se analiza el cumplimiento de los objetivos planteados al inicio del proyecto, según lo establecido en la sección 1.3 Objetivos.

10.2.1. Objetivos académicos

Objetivo 1 – Actualizar y fortalecer competencias técnicas – Cumplido

Se construyó una solución web moderna aplicando una infraestructura *serverless*, que involucra herramientas desconocidas la mayoría del equipo como por ejemplo Google Cloud Platform y Firebase *Cloud Functions*. El uso en conjunto de servicios como Google Cloud Pub/Sub, Google Cloud Run y AWS RDS desafió a todos los integrantes a nivel técnico y generó una cultura de aprendizaje continua a lo largo del proyecto.

Objetivo 2 – Diseño y gestión de *software* – Cumplido

Se aplicaron prácticas de arquitectura, gestión ágil y optimización a lo largo del proyecto, con un proceso que abarcó desde requerimientos y diseño hasta aseguramiento de la calidad y cierre, tal como fue planteado en los objetivos.

En la gestión, se sostuvo un ciclo **incremental–iterativo** con *releases* trazables, *changelog* y estandarización de *PRs*, lo que aportó visibilidad y control del cambio. En calidad, se combinaron pruebas automáticas (Jest, con cobertura > 90 %) y pruebas *E2E* sobre los flujos críticos (usuario y tienda), además de validaciones de usabilidad con actores representativos, fortaleciendo la evidencia de funcionamiento y la toma de decisiones.

10.2.2. Objetivos del producto

Objetivo 1 – Gestionar y centralizar devoluciones en *e-commerce* – Cumplido

Se implementó un flujo trazable y consistente de punta a punta para usuarios finales y tiendas, con tres interfaces web coordinadas. El módulo de tienda permite revisar, aprobar/rechazar y administrar políticas. El de usuario guía el inicio de devolución/cambio con carga de evidencia, y los dos paneles (tienda y administradores de Fitit) ofrecen visibilidad y capacidad de maniobra operativa.

Objetivo 2 – Mejorar la experiencia del usuario final – Cumplido

Se ejecutaron pruebas *E2E* sobre flujos reales y pruebas de usabilidad con perfiles representativos (usuarios finales y empleados de tienda) y los resultados confirmaron navegación clara, mensajes comprensibles y estados visibles, reduciendo fricción y errores.

Objetivo 3 – Generar métricas útiles para el negocio – Cumplido

El sistema registra motivos de devolución, tiempos de resolución y uso de créditos, dejando trazas que habilitan tableros y análisis posteriores. La estandarización del proceso y la documentación de resultados por *release* sostienen la confiabilidad de la información para la toma de decisiones.

Objetivo 4 – Aportar a Fitit un marco de validación – Cumlido

Al centralizar la postventa y estandarizar estados, el sistema permite vincular el comportamiento de las devoluciones con hipótesis del negocio (por ejemplo, impacto del recomendador de talles), sobre evidencia objetiva recogida en el propio flujo (razones, tiempos, reincidencia).

Objetivo 5 – Preparar integraciones externas – Cumplido

El uso del patrón *Strategy* deja una base extensible para conectar con plataformas de *e-commerce* y cadeterías sin afectar el núcleo del dominio.

10.2.3. Objetivos del proyecto

Objetivo 1 – Construir una plataforma eficiente y automatizada – Cumplido

Se llevó a cabo un proceso de *testing* completo (desde pruebas manuales documentadas hasta *testing* automatizado) y se verificó desempeño bajo concurrencia con k6 en *staging*, alcanzando latencia promedio de 18 ms y tasa de error de 0,33 %. A su vez, la estructuración en módulos y la automatización de despliegues agilizan el ciclo de entrega y reducen errores operativos.

Objetivo 2 – Asegurar transparencia y control extremo a extremo – Cumplido

Se mantuvo trazabilidad de decisiones y evidencias por *release*. Las pruebas funcionales en *staging* tras cada entrega confirmaron estabilidad de flujos, documentando resultados e incidencias menores. El *backoffice* quedó fuera del alcance de funciones por bajo riesgo y uso restringido, decisión registrada y justificada en el documento, manteniendo el foco en áreas de mayor impacto.

10.3. Cierre del proyecto con el cliente

El cierre del proyecto se realizó en conjunto con Fitit, estableciendo dos entregables principales y un plan de transición.

Entrega del código fuente
El código fue transferido al cliente a través del repositorio privado en GitHub, incluyendo las tres aplicaciones web, la configuración de entornos y las instrucciones de despliegue. Se documentaron las dependencias, reglas de seguridad y estructura de carpetas, garantizando la continuidad técnica del producto.

Entrega de documentación técnica
En lugar de un paquete documental independiente, se entregó un *README* único

dentro del repositorio con todo lo necesario para instalar, configurar y ejecutar el sistema, junto con una vista de la arquitectura y la organización del *monorepo*. Dado que Fitit ya contaba con acceso al repositorio privado y al proyecto de Firebase, no fue necesario duplicar información en manuales separados ni replicar accesos o credenciales: el *README* centraliza las instrucciones y referencias que el equipo de Fitit necesita para trabajar de forma autónoma.

El *README* incluye:

- **Prerrequisitos** como Node.js, NPM, Docker, permisos sobre el proyecto de Firebase, etc.
- **Configuración de entornos** con archivos *.env.example* por paquete y notas sobre variables sensibles.
- **Estructura del *monorepo*** (aplicaciones de usuario, tienda y *backoffice*, paquetes compartidos, funciones por dominio) y convenciones básicas de carpetas.
- **Pasos de instalación y ejecución local** mediante la instalación de dependencias, el uso de Firebase Emulator Suite y Docker para emular una base de datos MySQL localmente.
- **Guía breve de despliegue** a *staging* que incluyen detalles de cómo hacer: *build, deploy* y verificación rápida *post-deploy.*
- Una **descripción de la arquitectura** a alto nivel (responsabilidades de cada módulo, flujo de una devolución, puntos de integración y reglas de seguridad aplicadas).

Con este esquema, Fitit dispone de una referencia única, versionada y cercana al código, suficiente para levantar el entorno, realizar despliegues controlados y comprender las decisiones estructurales del sistema sin recurrir a documentación externa adicional. Con la entrega final, se estableció el cierre formal del proyecto, quedando Fitit con todos los elementos necesarios para continuar su desarrollo o integrarlo con el resto de su ecosistema digital.

10.4. Lecciones aprendidas

Las lecciones que dejó Fitit Returns abarcan producto, proceso, colaboración, y decisiones técnicas. Más que una lista aislada, el equipo describe qué aprendió, por qué, y recomendaciones. Cuando corresponde, se incluyen ejemplos concretos surgidos durante el proyecto.

10.4.1. Gestión del alcance y estrategia de entregas

El equipo comprobó que comenzar enfocado en el *MVP* y cerrar enfocado en fechas fue efectivo. En los primeros *sprints* priorizó el flujo base (iniciar solicitud, validar contra políticas, resolución por tienda y comunicación al cliente), lo que redujo incertidumbre al forzar decisiones concretas y habilitar validaciones con Fitit y tiendas

piloto. Cuando surgió el módulo de créditos, que impactaba políticas, paneles, *backoffice* y comunicación, el equipo rearmó los *releases* con objetivos y dependencias explícitas. Con el alcance estabilizado, el tramo final se orientó a estabilidad, migración de la librería de UI y documentación.

10.4.2. Producto y descubrimiento continuo

El equipo constató que el *feedback* temprano de tiendas y del equipo de Fitit evitó retrabajos costosos. Al validar políticas reales (plazos, excepciones, categorías excluidas), ajustó la UI con mensajes que explican el porqué, bloqueos con sentido y estados visibles. Cuando el sistema explica y no solo prohíbe, baja la frustración y sube la tasa de finalización.

10.4.3. Arquitectura y decisiones técnicas

El equipo confirmó que el enfoque *serverless* simplifica, pero exige disciplina en modelado de datos y observabilidad. Separar créditos en un servicio propio con base transaccional permitió razonar saldos/movimientos, evitar duplicados y mantener el resto simple. Diseñar integraciones (*e-commerce* y cadeterías) adaptables al cambio evitó reescrituras ante nuevos conectores.

10.4.4. Calidad, pruebas y herramientas

El equipo verificó que invertir en pruebas automáticas y en emulación cercana al real se paga solo. *Tests* que ejercitan reglas, decisiones y casos borde facilitaron *refactors*. Los *code reviews* con verificación de estilo/formato sostuvieron la consistencia y alineamiento del equipo.

10.4.5. Experiencia de usuario y accesibilidad

El equipo observó que menos pasos y mejores decisiones mejoran la experiencia. Pre-cargar datos cuando se ingresa desde la tienda (orden, correo, tienda) redujo dudas y favoreció la experiencia del usuario final. Detalles como validaciones oportunas, mensajes claros y estados vacíos orientadores cambiaron la percepción del sistema.

10.4.6. Proceso y colaboración

El equipo comprobó que Scrum a medida funcionó mejor que “Scrum de manual”. Con un equipo chico rindieron más reuniones de pie dos veces por semana que reuniones diarias, y retrospectivas cada cuatro *sprints* con pocas mejoras, pero con accionares claros y llevados a cabo con precisión. Kanban sirvió en investigación/documentación; *sprints* dieron foco en construcción. Documentar decisiones breves (qué, por qué, alternativas) evitó discusiones repetidas.

10.4.7. Gestión del tiempo y foco

El equipo trató el tiempo como requisito. Ajustar estimaciones a disponibilidad real (exámenes, feriados, viajes) y proteger bloques de trabajo profundo fue decisivo. La migración de UI tuvo costo de corto plazo, pero dejó una base más estable.

10.4.8. Documentación viva

El equipo validó que documentar mientras construye reduce la deuda final. Notas cortas en el tablero, commits informativos y un README central valen más que cien páginas al cierre.

10.5. Reflexiones personales

Nicolás Andreoli

“Este proyecto fue una oportunidad excelente para trabajar con un cliente real en un problema de alto impacto: ordenar el flujo de devolución y brindar un seguimiento claro que involucre a los distintos actores. Desde el comienzo me propuse que la solución fuera sencilla de entender y confiable de usar, tanto para quien gestiona el proceso en la tienda como para el usuario que inicia la solicitud.

Me resultó especialmente motivante diseñar el manejo de créditos como una parte separada del resto, con reglas simples y transparentes. Eso nos permitió evitar errores típicos, como aplicar un crédito dos veces o perder el rastro de por qué se otorgó, y, sobre todo, dar mejor visibilidad.

Otro aspecto que disfruté fue la disciplina para mantener la estabilidad: automatizamos pruebas, armamos un entorno de pruebas idéntico al real y registramos lo suficiente como para detectar problemas temprano. No es lo más vistoso, pero es lo que marca la diferencia cuando el sistema crece.

En el trabajo cotidiano apostamos a cambios pequeños y revisiones frecuentes, cuando algo se trababa, nos juntábamos para definir como seguir. Aprendí a priorizar lo que realmente agrega valor y despriorizar lo que no justificaba.

Me quedo muy conforme con el resultado obtenido y lo que aprendimos en el proceso. El equipo tuvo mucha sinergía desde el día cero y eso hizo que todo fuera más fácil.”

Florencia Batlle

“Lo más valioso de Fitit Returns fue sostener un ritmo de trabajo estable y convertirlo en resultados concretos en cada sprint. Desde la coordinación buscamos que el avance no dependiera del día ni del ánimo, sino de acuerdos claros por iteración, criterios de aceptación bien escritos, demostraciones con avances reales y devolución aplicada en el ciclo siguiente. Ajustamos la forma de trabajar según la etapa, con mayor flexibilidad cuando investigábamos o escribíamos y con sprints de dos semanas en desarrollo para mantener el foco. Mantuvimos tableros visibles y tareas claras para todos, lo que ayudó a anticipar bloqueos y a ordenar mejor el esfuerzo del equipo.

El mayor desafío fue la incorporación del sistema de créditos a mitad del proyecto. No era un cambio menor y la salida fácil era apurar, pero elegimos ordenar.

Replanificamos entregas, cuidamos la calidad y explicamos con calma qué iba primero y por qué. Las listas de verificación y los estándares redujeron la carga del equipo y evitaron depender de la memoria de una sola persona. Este orden permitió avanzar sin perder trazabilidad y facilitó la comunicación.

Me queda la satisfacción de ver un producto coherente, con métricas y documentación, y un equipo que creció en disciplina y autonomía. Aprendí a priorizar con datos, a no sobrerreaccionar ante los cambios y a cuidar lo básico de preparar, revisar y cerrar. Sostuvimos revisiones entre pares, un entorno de pruebas equivalente al real y registros simples que ayudaron a detectar problemas temprano. El resultado es una solución clara, estable y lista para seguir mejorando sin fricciones innecesarias."

Federico Czarnievicz
"Desde el comienzo, mi foco fue acercar el negocio al producto. Como enlace con el cliente y responsable de requerimientos, me dediqué a entender cómo vive cada tienda el proceso de devolución: qué esperan, qué les preocupa, dónde se tranca la gente y qué los hace perder tiempo. De esas conversaciones salieron políticas claras, como plazos, condiciones, excepciones, y tareas concretas con criterios fáciles de comprobar. Las entrevistas, los prototipos y las pruebas con usuarios nos enseñaron algo que parece obvio pero no siempre se cumple: la mejor funcionalidad es la que reduce fricción. Eso se tradujo en mensajes que explican el porqué, formularios más simples, estados visibles y una usabilidad amigable, sin que el usuario tenga que adivinar.

Disfruté mucho trabajar esos detalles de comunicación: cambiar una frase, ordenar un campo o mostrar un aviso antes puede evitar un reclamo y mejorar la percepción de todo el sistema. Medimos y lo vimos: menos dudas, menos pasos y tiempos más cortos para completar los flujos. También cuidé que el backlog se mantuviera vivo y ordenado: priorizamos lo que más impacto tenía para tiendas y clientes, y acordamos con Fitit qué entraba en cada entrega para no dispersarnos. En ese ida y vuelta, aprendí a que no, sin que suene a rechazo, y a defender con datos las decisiones que simplifican.

Me voy con el aprendizaje de priorizar valor antes que cantidad de funciones, y con la tranquilidad de haber dejado una solución usable, medible y lista para escalar dentro del ecosistema de Fitit. Si empezara otra vez, arrancaría con un glosario compartido para que todos hablemos el mismo idioma y un canal único para reportar problemas con un formato simple. Más allá de eso, repetiría el camino: escuchar, sintetizar, probar y ajustar hasta que las cosas sean claras para todos."

10.6. Cierre final

Fitit Returns es más que una tesis: es una solución operable que ordena devoluciones y cambios en *e-commerce* con reglas claras, trazabilidad y tiempos razonables. Nació de meses de trabajo sostenido, investigación aplicada y colaboración con un cliente real, y hoy se apoya en una base técnica simple y escalable que puede seguir creciendo.

En lo técnico, el sistema quedó maduro para continuar su evolución: módulos separados por responsabilidades (usuarios, tienda y *backoffice*), un módulo de créditos con reglas claras y auditables, e infraestructura *serverless* con despliegues reproducibles. La presencia de un *README* único (instalación, entornos, despliegue) y *workflows* de verificación automática reduce la dependencia del equipo original y facilita el traspaso al cliente.

En el proceso, el enfoque híbrido (Scrum para construir, Kanban para investigar y documentar) permitió sostener cadencia y absorber cambios —en especial la incorporación de créditos a mitad del proyecto— sin perder calidad. Se eligió iterar con incrementos verificables, criterios de aceptación medibles y documentación viva (decisiones, listas de verificación) que acompañó el desarrollo.

Desde el producto, el valor se ve en menos fricción y más visibilidad. Para el usuario final, hay menos pasos y mejores mensajes, para la tienda, políticas aplicadas de forma consistente y una línea de tiempo que registra decisiones, para Fitit, una plataforma integrable que habilita adopción en distintos contextos comerciales. La validación con tiendas piloto y las mejoras de usabilidad por iteraciones son evidencia directa de ese impacto.

El camino deja aprendizajes duraderos: validar temprano y con evidencia, planificar con márgenes realistas, automatizar lo repetible, escribir lo esencial mientras se construye y sostener una comunicación clara con el cliente.

En síntesis, Fitit Returns muestra cómo un proyecto académico puede convertirse en una solución real, útil y escalable, lista para integrarse al ecosistema de *e-commerce*. Combina rigor técnico con empatía por el usuario y un proceso de trabajo sano. Con esa brújula, simpleza bien diseñada, integrabilidad desde el inicio, trazabilidad de punta a punta y comunicación temprana, el producto queda preparado para generar impacto y el equipo listo para llevar estos aprendizajes a los desafíos que vienen.

11. Bibliografía

[1] Shopify. (2024). *Global Ecommerce Statistics: Trends to Guide Your Store in 2025.* [En línea]. Disponible en: https://www.shopify.com/enterprise/blog/global-ecommerce-statistics [Consulta: 25 de Septiembre de 2025].

[2] Shopify. (2025). *Ecommerce Returns: Average Return Rate and How to Reduce It*. Shopify. [En línea]. Disponible en: https://www.shopify.com/enterprise/blog/ecommerce-returns[Consulta: 25 de Septiembre de 2025].

[3] WeSupply Labs. (2024). *Why Returns Can Drive Repeat Purchases*. WeSupply Labs. [En línea]. Disponible en: https://wesupplylabs.com/why-returns-can-drive-repeat-purchases/ [Consulta: 25 de Septiembre de 2025].

[4] Ahmed, W., Huma, S. & Ali, S. U. (2024). *Influence of return convenience on young buyers' repurchase intentions*. Young Consumers. [En línea]. Disponible en: https://www.emerald.com/insight/content/doi/10.1108/YC-02-2023-1691/full/html [Consulta: 25 de Septiembre de 2025].

[5] API University, *Webhooks: Events for RESTful APIs*, Vol. 4. API-University Press, 2018. [En línea]. Disponible en: https://www.api-university.com/books/webhooks-events-for-restful-apis/ [Último acceso: 01 Octubre 2025]

[6] Worcester Polytechnic Institute, *Webhooks-as-a-Service: A Custom API Design*. Digital WPI, 2020. [En línea]. Disponible en: https://digital.wpi.edu/downloads/d504rp58n [Último acceso: 01 Octubre 2025]

[7] Google Cloud, *Best practices for managing API keys*. Google, 2023. [En línea]. Disponible en: https://cloud.google.com/docs/authentication/api-keys-best-practices [Último acceso: 01 Octubre 2025]

[8] Nordic APIs, *Keep API Keys Safe, Because The Repercussions Are Huge*, 2021. [En línea]. Disponible en: https://nordicapis.com/keep-api-keys-safe-because-the-repercussions-are-huge/ [Último acceso: 01 Octubre 2025]

[9] i18next. "i18next Documentation." i18next.com, s.f. [En línea]. Disponible en: https://www.i18next.com/ [Consulta: 20 de Setiembre de 2025].

[10] Beck, K. et al. Manifesto for Agile Software Development, 2001. [En línea]. Disponible en: https://agilemanifesto.org [Consulta: 25 de Septiembre de 2025].

[11] Proyectos Ágiles. "Desarrollo iterativo incremental." *Proyectos Ágiles*, 2018. [En línea]. Disponible en: https://proyectosagiles.org/desarrollo-iterativo-incremental/ [Consulta: 25 de Septiembre de 2025].

[12] Asana. (2025). *Guía de la metodología en cascada*. Recuperado de https://asana.com/es/resources/waterfall-project-management-methodology

[13] Kniberg, H. (2016). *Making sense of MVP*. Crisp Blog. [En línea]. Disponible en: https://blog.crisp.se/2016/01/25/henrikkniberg/making-sense-of-mvp [Consulta: 25 de Septiembre de 2025].

[14] Schwaber, K. y Sutherland, J. *The Scrum Guide*. Scrum.org, noviembre 2020. [En línea]. Disponible en: https://scrumguides.org [Consulta: 25 de Septiembre de 2025].

[15] Anderson, D. *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press, 2010. [En línea]. Disponible en: https://kanban.university [Consulta: 25 de Septiembre de 2025].

[16] Wikipedia. *Planning Poker*. Wikipedia, 2025. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Planning_poker [Consulta: 25 de Septiembre de 2025].

[17] PlanningPokerOnline. *PlanningPokerOnline.com*. s.f. [En línea]. Disponible en: https://planningpokeronline.com/ [Consulta: 20 de Abril de 2025].

[18] TechTarget. "What is Feature-Driven Development (FDD)?" *TechTarget SearchSoftwareQuality*. 2024. [En línea]. Disponible en: https://www.techtarget.com/searchsoftwarequality/definition/feature-driven-development [Consulta: 25 de Septiembre de 2025].

[19] Muñoz, S. (2022). *Deadline Driven Development*. Estrategia de Producto. [En línea]. Disponible en: https://www.estrategiadeproducto.com/p/deadline-driven-development [Consulta: 10 de Octubre de 2025].

[20] Sommerville, I. *Software Engineering*, 10ª edición. Pearson, 2016.

[21] IEEE Computer Society. *ISO/IEC/IEEE 29148-2018 – Systems and Software Engineering — Life Cycle Processes — Requirements Engineering.* IEEE, 2018. [En línea]. Disponible en: https://standards.ieee.org/standard/29148-2018.html [Consulta: 5 de Octubre de 2025].

[22] Lewrick, Michael, Patrick Link, y Larry Leifer. *The Design Thinking Playbook: Mindful Digital Transformation of Teams, Products, Services, Businesses and Ecosystems*. Wiley, 2018.

[23] Interaction Design Foundation. "Design Thinking." Interaction Design Foundation, s.f. [En línea]. Disponible en: https://www.interaction-design.org/literature/topics/design-thinking?srsltid=AfmBOopgCDog-uglwC5n_oUeSePx27-8R-bcz-hE4iWQcUmA2zCQYRBs#the_five_stages_of_design_thinking-6 [Consulta: 2 de Octubre de 2025].

[24] Uruguay. Ley N° 17.250 de 11 de agosto de 2000: Relaciones de Consumo. Artículo 16.[En línea]. Disponible en https://www.impo.com.uy/bases/leyes/17250-2000 [Consulta: 12 de Octubre de 2025].

[25] Gorgias. *"The Rise of Online Sizing Solutions for E-commerce Fashion Brands"*. Gorgias Blog, 22 de septiembre de 2025. [En línea]. Disponible en:

https://www.gorgias.com/blog/online-sizing-solutions [Consulta: 5 de Octubre de 2025]. gorgias.com

[26] Radial. *“Tech takes on e-commerce's $218 billion returns problem”*. Radial Insights. s.f [En línea]. Disponible en: https://www.radial.com/eur/insights/tech-takes-on-e-commerces-218-billion-returns-problem [Consulta: 5 de Octubre de 2025]. Radial

[27] VTEX. “Plataforma de comercio digital y marketplace”. VTEX.com, s.f. [En línea]. Disponible en: https://vtex.com/es-es/ [Consulta: 5 de Octubre de 2025].

[28] Agile Alliance. “Epic.” *Agile Alliance Glossary*, 2022. [En línea]. Disponible en: https://www.agilealliance.org/glossary/epic
[Consulta: 5 de Octubre de 2025].

[29] Cohn, Mike. *User Stories Applied: For Agile Software Development.* Addison-Wesley Professional, 2004.

[30] Wake, Bill. “INVEST in Good Stories, and SMART Tasks.” *XP123*, 2003. [En línea]. Disponible en: https://xp123.com/articles/invest-in-good-stories-and-smart-tasks
[Consulta: 5 de Octubre de 2025].

[31] L. Bass, P. Clements and R. Kazman, Software Architecture in Practice, 4th ed, Boston, MA, USA: Pearson Addison-Wesley, 2021.

[32] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord and J. Stafford, Documenting Software Architectures, Views and Beyond, 2th ed, Boston, MA, USA: Pearson Addison-Wesley, 2010.

[33] Google Cloud. “¿Qué es la arquitectura sin servidor (serverless)?”. Google Cloud Documentation, s.f. [En línea] Disponible en: https://cloud.google.com/discover/what-is-serverless-architecture?hl=es

[34] npm-compare.com, "next vs nuxt | Frameworks for Server-Side Rendering Comparison", s.f. [En línea]. Disponible en: https://npm-compare.com/next,nuxt. [Consulta: 2 de Octubre de 2025].

[35] AWS. *Mistakes to Avoid When Implementing Serverless Architecture with AWS Lambda.* Blog de AWS Architecture, 2020. [En línea] Disponible en: https://aws.amazon.com/es/blogs/architecture/mistakes-to-avoid-when-implementing-serverless-architecture-with-lambda/. [Consulta: 8 de Octubre de 2025].

[36] AWS. *Operating Lambda: Anti-patterns in Event-Driven Architectures – Part 3.* Blog de AWS Compute, 2023. [En línea] Disponible en: https://aws.amazon.com/es/blogs/compute/operating-lambda-anti-patterns-in-event-driven-architectures-part-3/. [Consulta: 8 de Octubre de 2025].

[37] Beck, K. *Test-Driven Development: By Example*. Addison-Wesley Professional, 2003.

[38] B. Frost, Atomic Design, 1st ed., Pittsburgh, PA, USA: CreateSpace Independent Publishing Platform, 2016. [En línea]. Disponible en: https://atomicdesign.bradfrost.com. [Consulta: 2 de Octubre de 2025].

[39] Refactoring.Guru, “Strategy Design Pattern”, *Refactoring.Guru*, s.f. [En línea]. Disponible en: https://refactoring.guru/design-patterns/strategy. [Consulta: 12 de Octubre de 2025].

[40] Google Cloud. “Arquitecturas basadas en eventos”. Google Cloud Documentation, s.f. [En línea] Disponible en: https://cloud.google.com/eventarc/docs/event-driven-architectures?hl=es-419 [Consulta: 12 de Octubre de 2025].

[41] Google Cloud, “Overview of Pub/Sub”, *Google Cloud Documentation*, s.f. [En línea]. Disponible en: https://cloud.google.com/pubsub/docs/overview?hl=es-419. [Consulta: 12 de Octubre de 2025].

[42] Microsoft, “Federated Identity pattern”, *Microsoft Learn – Cloud Design Patterns*, s.f. [En línea]. Disponible en: https://learn.microsoft.com/en-us/azure/architecture/patterns/federated-identity. [Consulta: 12 de Octubre de 2025].

[43] IEEE Computer Society. “What is Software Quality?”, 2023. [En línea]. Disponible en: https://www.computer.org/resources/what-is-software-quality/ . [Consulta: 1 de Octubre de 2025].

[44] Yep. *yup*. GitHub Repository, s.f. [En línea]. Disponible en: https://github.com/jquense/yup.[Consulta: 13 de Agosto de 2025].

[45] IBM. *¿Qué son las pruebas de software?,* 2024. [En línea]. Disponible en: https://www.ibm.com/mx-es/think/topics/software-testing. [Consulta: 4 de Octubre de 2025].

[46] OpenJS Foundation. *Jest*. Jest Documentation, s.f. [En línea]. Disponible en: https://jestjs.io/. [Consulta: 20 de Setiembre de 2025].

[47] Graphite. *“Code Coverage Best Practices”*, 2024. [En línea]. Disponible en: https://graphite.dev/guides/code-coverage-best-practices. [Consulta: 4 de Octubre de 2025].

[48] GeeksforGeeks. *Software Engineering | Black Box Testing*. 2025. [En línea]. Disponible en: https://www.geeksforgeeks.org/software-testing/software-engineering-black-box-testing/. [Consulta: 2 de Octubre de 2025].

[49] QAlified. *Pruebas de carga eficientes con k6: Descubre sus ventajas.* 2024. [En línea]. Disponible en: https://qalified.com/es/blog/k6-pruebas-de-carga/. [Consulta: 2 de Octubre de 2025].

[50] Nielsen, J. *Ten Usability Heuristics*. Nielsen Norman Group. 1994. [En línea]. Disponible en: https://www.nngroup.com/articles/ten-usability-heuristics/. [Consulta: 3 de Octubre de 2025].

[51] Thoughtworks. *Monorepo vs. multi-repo: Different strategies for organizing repositories.* 2023. [En línea]. Disponible en: https://www.thoughtworks.com/insights/blog/agile-engineering-practices/monorepo-vs-multirepo. [Consulta: 2 de Octubre de 2025].

[52] Git Flow vs Github Flow. *GeeksforGeeks*. 2025. [En línea]. Disponible en: https://www.geeksforgeeks.org/git/git-flow-vs-github-flow/. [Consulta: 20 de Setiembre de 2025].

[53] Preston-Werner, T. *Semantic Versioning 2.0.0*. SemVer.org, 2013. [En línea]. Disponible en: https://semver.org/spec/v2.0.0.html. [Consulta: 20 de Agosto de 2025].

12. Anexo

12.1. Pantallas de *backoffice*

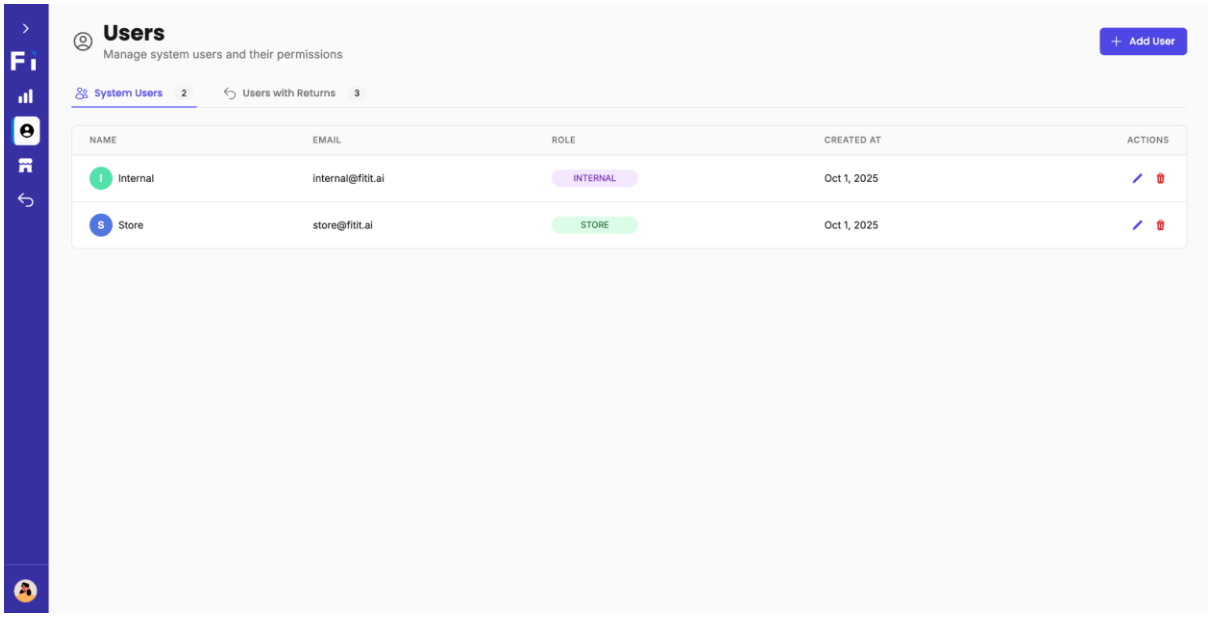


Ilustración 78 - Gestión de usuarios del sistema en fitit-backoffice

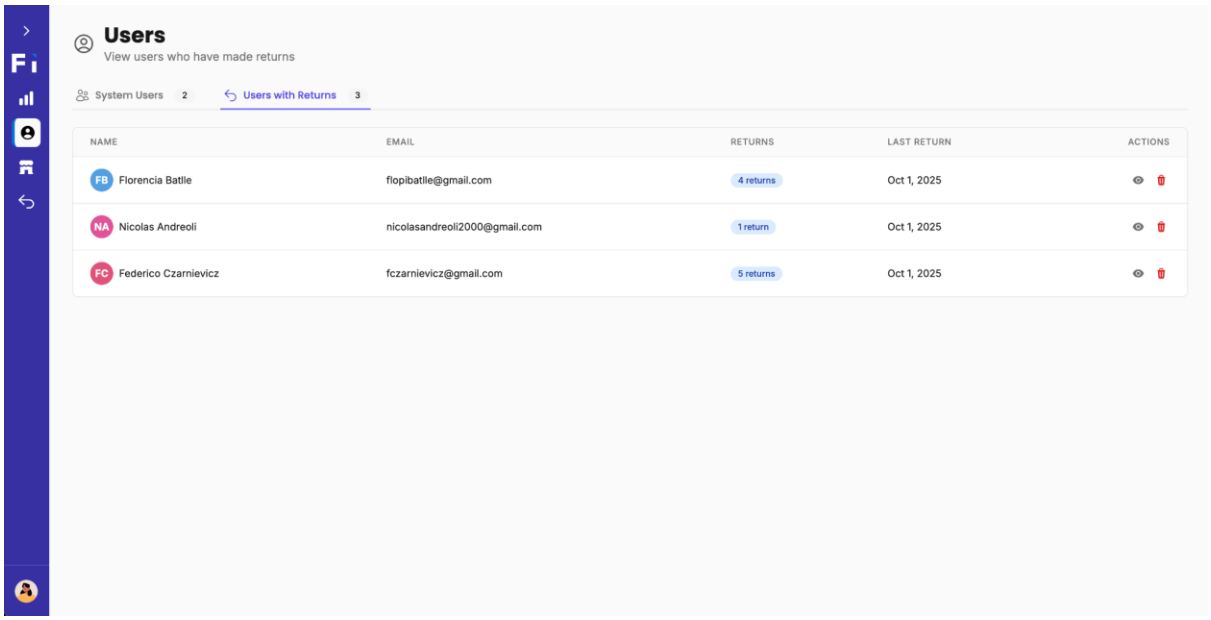


Ilustración 79 - Gestión de usuarios con devoluciones en fitit-backoffice

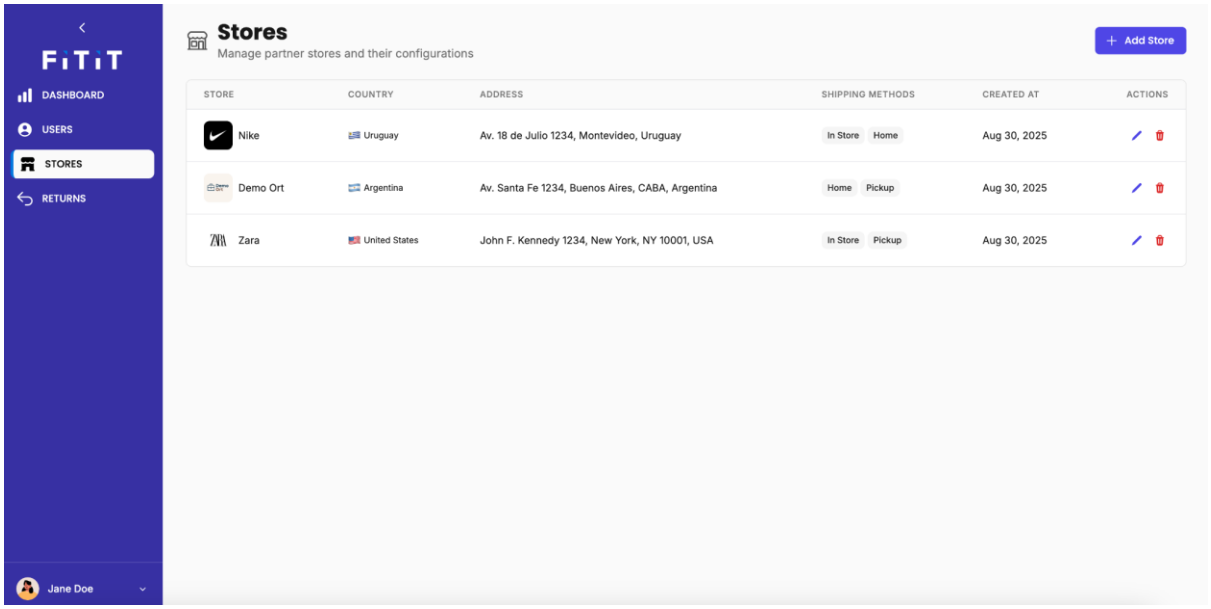


Ilustración 80 - Gestión de tiendas en fitit-backoffice

12.2. Pantallas en *mobile*

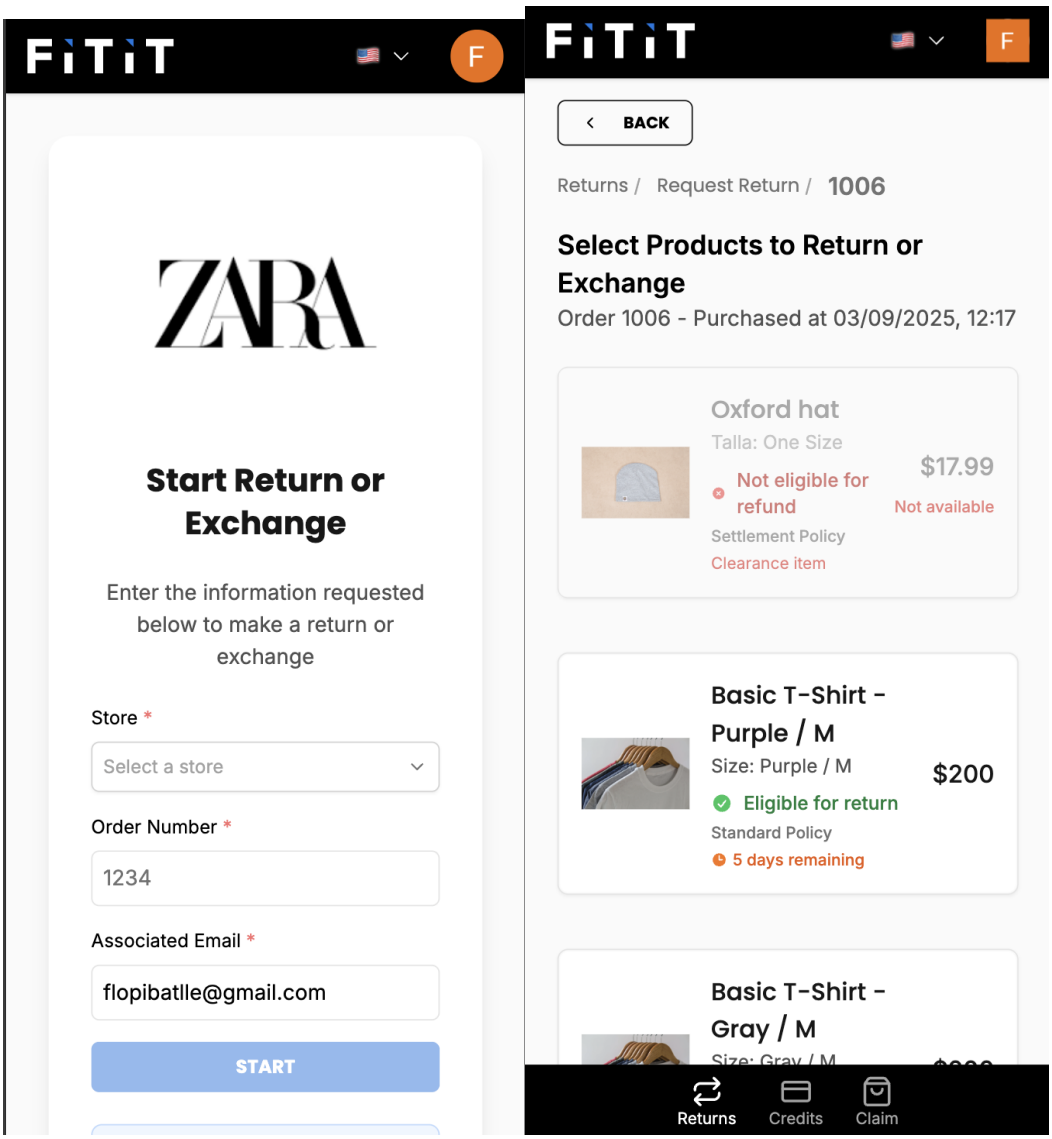


Ilustración 81 - Pantallas de inicio del proceso de devolución

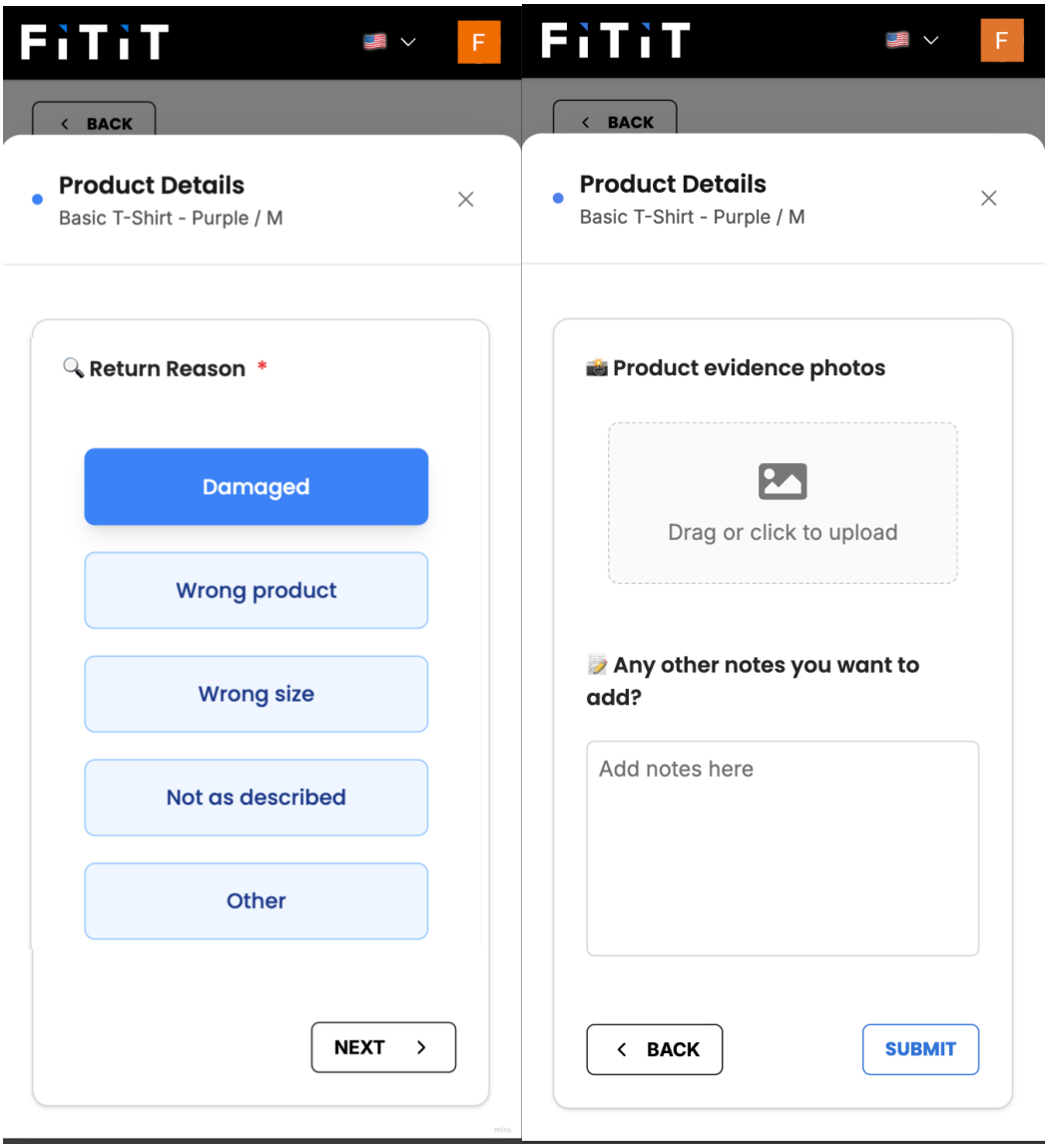


Ilustración 82 - Pantallas de inicio del proceso de devolución (continuación)

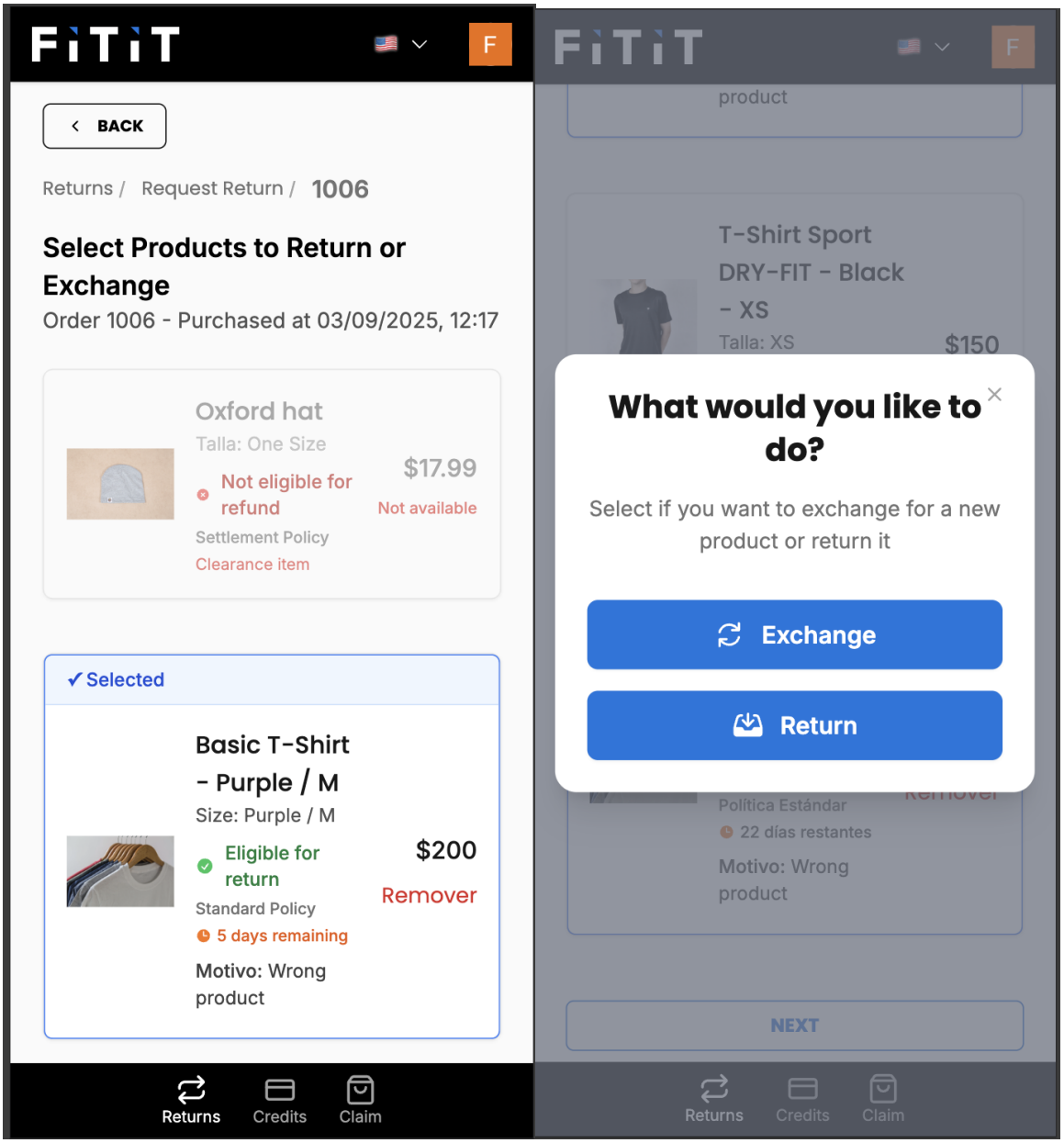


Ilustración 83 - Pantallas de inicio del proceso de devolución (continuación)

12.3. Product Backlog

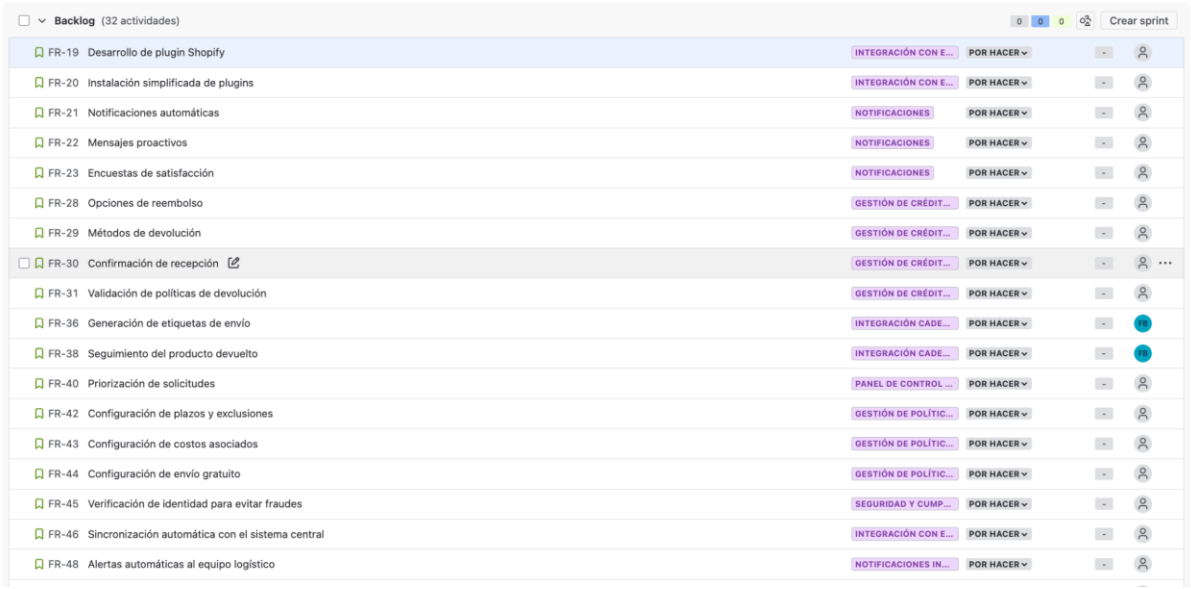


Ilustración 84 - Product Backlog

12.4. Funcionalidades por *release* y descartadas

Funcionalidad	Release 1 – 08/03/2025 (*MVP*)	Release 2 – 03/05/2025	Release 3 – 12/07/2025	Release 4 – 06/09/2025
Gestión de devoluciones	Flujo mínimo: cliente inicia devolución y tienda aprueba/rechaza	Mejoras de gestión y validaciones	Validaciones más avanzadas y reglas de negocio	Estabilización y refinamiento final
Políticas de devolución	No	No	Inicio: configuración por tienda (plazos, excepciones básicas)	Ampliaciones y consolidación
Notificaciones	No	No	Email por eventos básicos	Consolidación del canal de email
Créditos de tienda	No	Se incorpora el módulo base (alta/uso/consulta)	Reglas de vencimiento y ajustes operativos	Consolidación del módulo (consistencia y controles)
Reportes y dashboards	No	Dashboard de tienda	Primeros reportes de devoluciones	Optimización/mejoras de usabilidad

Flujos de login/registro	Login y signup inicial	Refinamiento de validaciones	Ajustes de seguridad y usabilidad	Consolidación junto con nueva UI
Librería de UI / Diseño	UI inicial	UI mantenida	UI mantenida	Migración a nueva librería de UI
Integraciones con *e-commerce* y couriers	Conectores básicos habilitados	Extensión y ajustes	Nuevos conectores/validaciones	Consolidación de integraciones
Backoffice	No	No	Dashboard y listados iniciales	*Backoffice* completo.
***Webhooks* & Public API**	No	No	No	Implementación de *webhooks* + *API* pública

Tabla 15 - Funcionalidades principales implementadas por *release*

Funcionalidades descartadas

En paralelo, también se analizaron ciertas funcionalidades que finalmente no fueron incluidas en esta primera versión, ya que el equipo decidió priorizar los recursos disponibles en el núcleo del sistema. Estas decisiones no significan que hayan quedado descartadas de forma definitiva, sino que fueron pospuestas para futuras fases donde tengan más sentido y puedan implementarse con la madurez adecuada del producto.

- **Plugins para integración con *e-commerce*:** se discutió la posibilidad de ofrecer plugins listos para plataformas de *e-commerce* como Shopify o VTEX,

con el fin de facilitar la adopción del sistema. Sin embargo, esta opción fue descartada ya que hubiera implicado dedicar un esfuerzo considerable a mantener compatibilidad con múltiples versiones de cada plataforma, lo cual excedía el alcance de este proyecto. El equipo entendió que, en esta etapa inicial, era más eficiente concentrar los recursos en la plataforma central, garantizando estabilidad y robustez, antes de abrir la integración mediante soluciones empaquetadas.

- ***SDK* propio**: también se evaluó la creación de un kit de desarrollo (*SDK*) que permitiera a terceros integrar el sistema de forma personalizada. A pesar de ser una idea con gran potencial, se descartó en esta primera versión porque hubiera requerido definir y mantener librerías específicas, documentación técnica extensa y soporte para desarrolladores externos, lo que suponía una sobrecarga de mantenimiento difícil de sostener con un equipo reducido.

  No obstante, como alternativa se avanzó en la apertura de una Public API, que si bien no constituye un *SDK* completo, sí ofrece una vía de integración más flexible y ligera para terceros. Esta decisión permitió habilitar casos de uso externos sin comprometer la capacidad del equipo, dejando abierta la posibilidad de evaluar un *SDK* en el futuro.

- **Notificaciones SMS:** durante el análisis inicial se consideró el envío de SMS como canal complementario de comunicación. Sin embargo, el *feedback* de las tiendas indicó que el email era suficiente para notificar a los clientes en este tipo de procesos.

En todos los casos, la decisión de descartar estas funcionalidades se tomó en conjunto con el tutor y el cliente, buscando un equilibrio entre alcance, tiempo y calidad. De esta manera, el sistema logró entregar un producto funcional, estable y alineado con los objetivos principales, sin desviar esfuerzos hacia características que podían posponerse sin comprometer el éxito del proyecto.

12.5. Evolución en la UI de pantallas principales

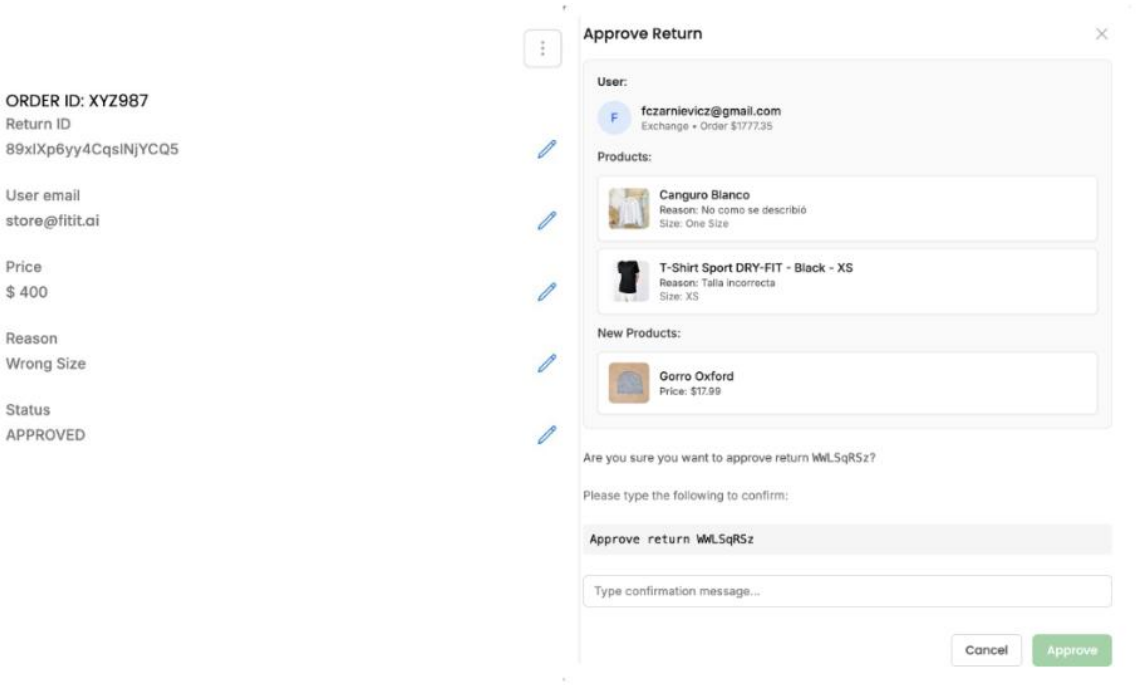


Ilustración 85 - Evolución de detalle de devolución en tienda (Primer *release* a la izquierda y cuarto *release* a la derecha)

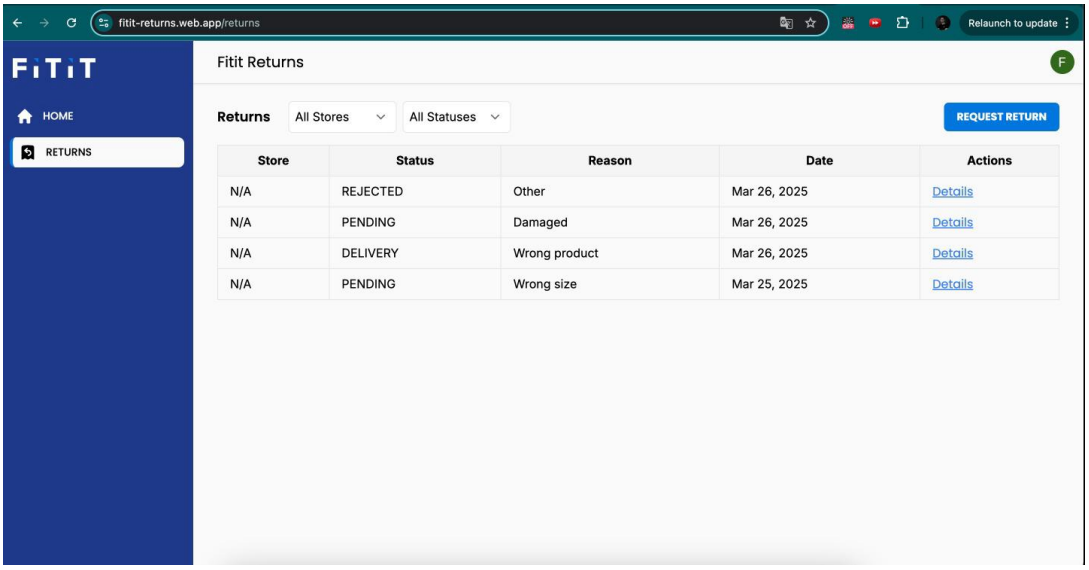


Ilustración 86 - Pantalla del listado de devoluciones (usuario final) en el primer *release*

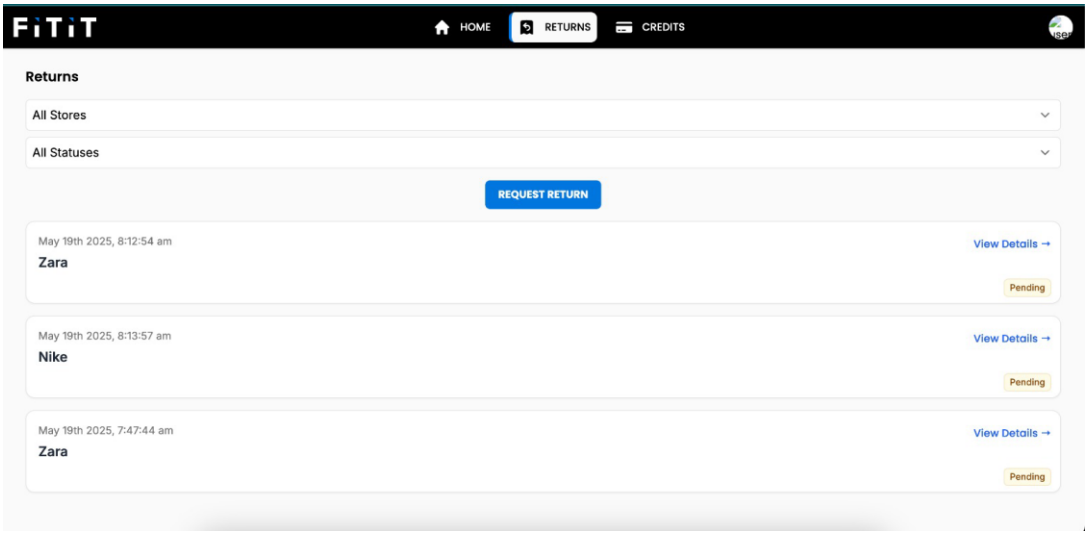


Ilustración 87 - Pantalla del listado de devoluciones (usuario final) en el segundo y tercer *release*

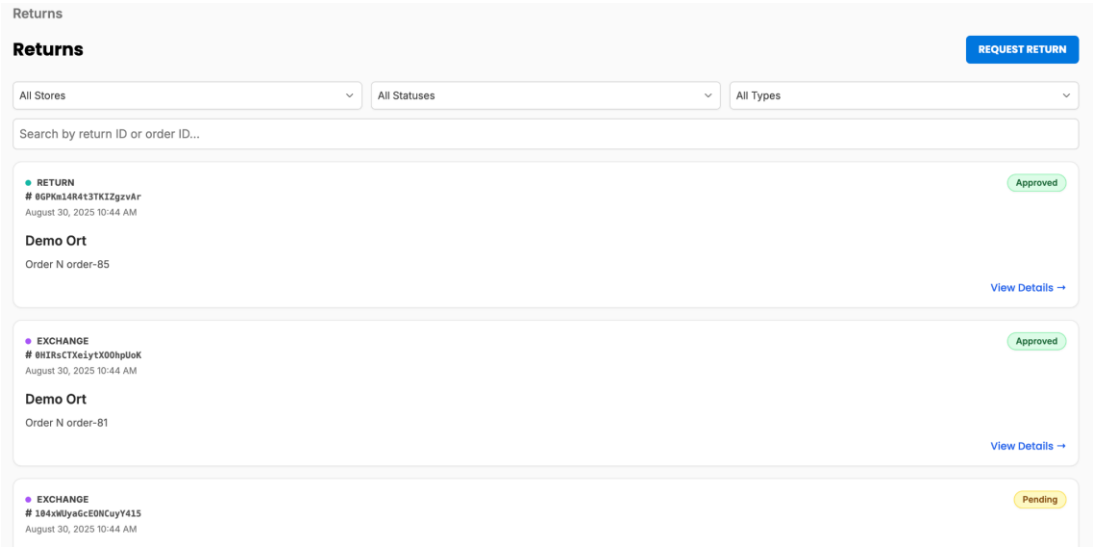


Ilustración 88 - Pantalla final del listado de devoluciones (usuario final) en el cuarto *release*

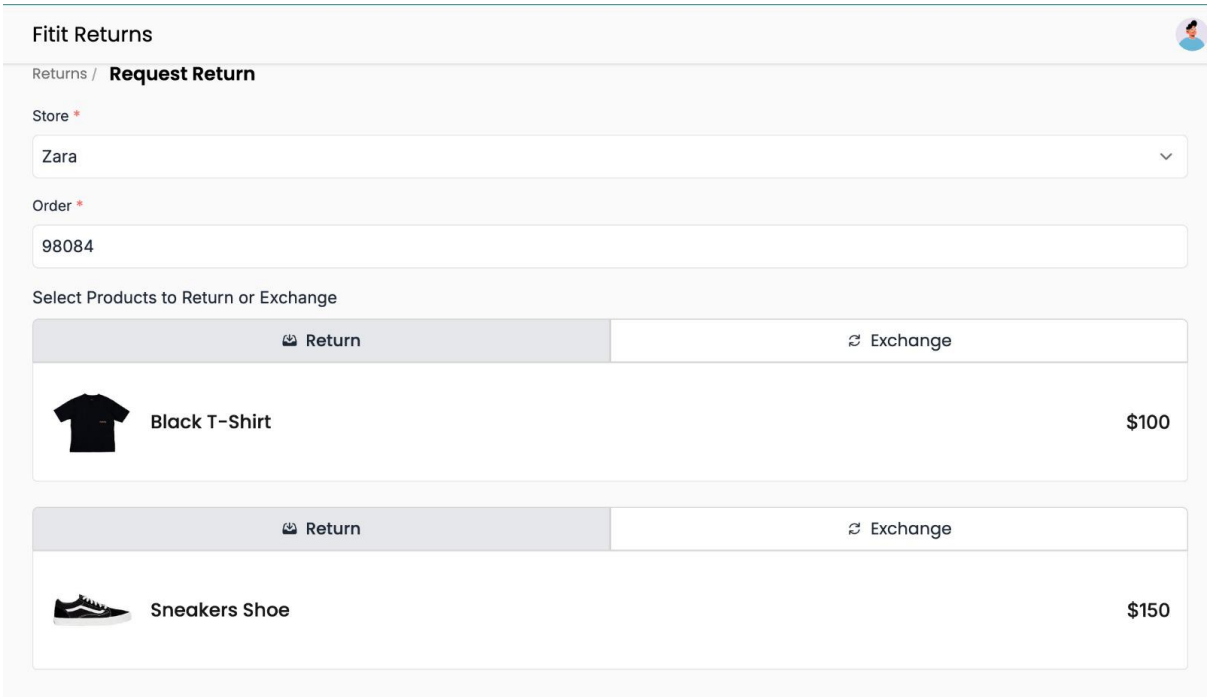


Ilustración 89 - Pantalla de inicio de flujo de devolución (usuario final) en el primer *release*

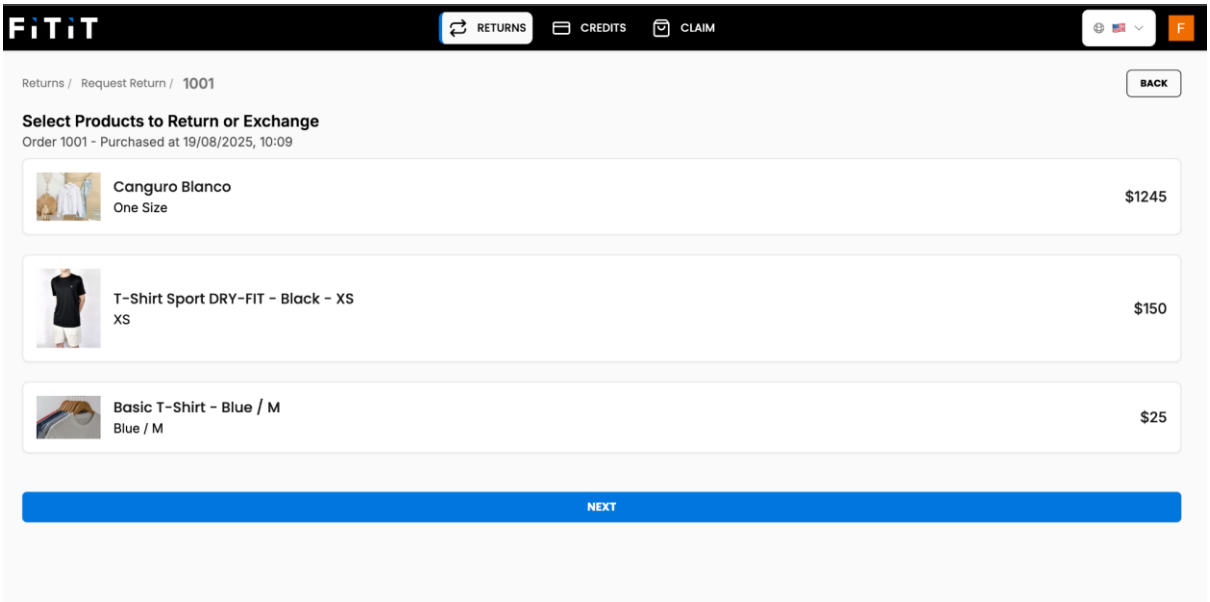


Ilustración 90 - Evolución inicio de flujo de devolución (usuario final) en el cuarto *release*

12.6. *Release Plan*

Ilustración 91 - *Release Plan*

12.7. Escalas y metodología de análisis de riesgos.

Para identificar los riesgos se utilizó la técnica de brainstorming entre los integrantes del equipo, complementada con las experiencias previas de cada uno en proyectos académicos y laborales. Una vez listados, los riesgos fueron analizados en términos de impacto y probabilidad de ocurrencia, asignando valores según escalas predefinidas.

El impacto se midió en una escala del 1 al 5, siendo 1 un impacto muy bajo y 5 un impacto muy alto en el proyecto:

Impacto	**Descripción**
0	No impacta
1	Muy bajo
2	Bajo
3	Medio
4	Alto
5	Muy alto

Tabla 16 - Escala de impacto de riesgos

La probabilidad se midió en valores de 0 a 1, donde 0 significa que no es probable y 1 que es casi seguro que se convierta en un problema:

Probabilidad	**Descripción**
0	No probable
0.2	Poco probable
0.4	Probable
0.6	Bastante probable

0.8	Muy probable
1	Se convierte en un problema

Tabla 17 - Escala de probabilidad de riesgos

La magnitud del riesgo se calculó como el producto entre impacto y probabilidad, lo que permitió priorizar aquellos con mayor incidencia.

Probabilidad/Impacto	**0**	**1**	**2**	**3**	**4**	**5**
0,0	0,0	0,0	0,0	0,0	0,0	0,0
0,2	0,0	0,2	0,4	0,6	0,8	1,0
0,4	0,0	0,4	0,8	1,2	1,6	2,0
0,6	0,0	0,6	1,2	1,8	2,4	3,0
0,8	0,0	0,8	1,6	2,4	3,2	4,0
1,0	0,0	1,0	2,0	3,0	4,0	5,0

Tabla 18 - Matriz de probabilidad vs. impacto de riesgos

Esta representación permitió identificar rápidamente los riesgos más críticos del proyecto. Los valores más altos, marcados en rojo, indican aquellos que requerían atención inmediata, mientras que los valores medios (amarillos) y bajos (verdes) se controlaron con menor prioridad. Con esta información se definieron las estrategias de mitigación y los responsables de seguimiento para cada riesgo.

12.8. Análisis cualitativo de riesgos

Riesgo	Impacto	Probabilidad	Magnitud	Plan de contingencia	Plan de respuesta
RK1 – Problemas de sincronización del equipo	4	0.6	2.4	Implementar rutinas de coordinación semanales.	Se instauraron *standups* dos veces por semana y se reforzó el uso de Whatsapp y Jira.
RK2 – Cambios en requerimientos	5	0.8	4.0	Mantener *backlog* flexible y priorizar con el cliente.	Se reorganizaron *releases* (ej: incorporación de créditos en R2) y se reestimaron *sprints*.
RK3 – Productividad del equipo	4	0.6	2.4	Ajustar estimaciones a la disponibilidad real de horas.	Redistribución de tareas y criterios de *story points* más realistas.
RK4 – Falta de experiencia en alguna tecnología puntual	2	0.4	0.8	Investigar antes de usar la tecnología.	Dedicar horas de aprendizaje y apoyarse en la documentación oficial.
RK5 – Errores estimación de tareas	2	0.4	0.8	Dejar un margen de seguridad en la planificación.	Reajustar la estimación y redistribuir tareas dentro del *sprint*.
RK6 – Descoordinación en la documentación	2	0.2	0.4	Revisiones cruzadas frecuentes.	Integrar los cambios en un único documento central y corregir inconsistencias.

Riesgos que aparecieron en el proceso					
RK7 – Comunicación con el cliente	3	0.6	1.8	Calendarizar reuniones periódicas y demos en cada *release*.	Documentar avances en Jira y compartir mails en ausencia de *feedback* formal.

Tabla 19 - Análisis cualitativo de los riesgos

12.9. *Design Thinking*

12.9.1. Resultados de encuestas a usuarios

Ilustración 92 - Encuesta de usuarios pt.1

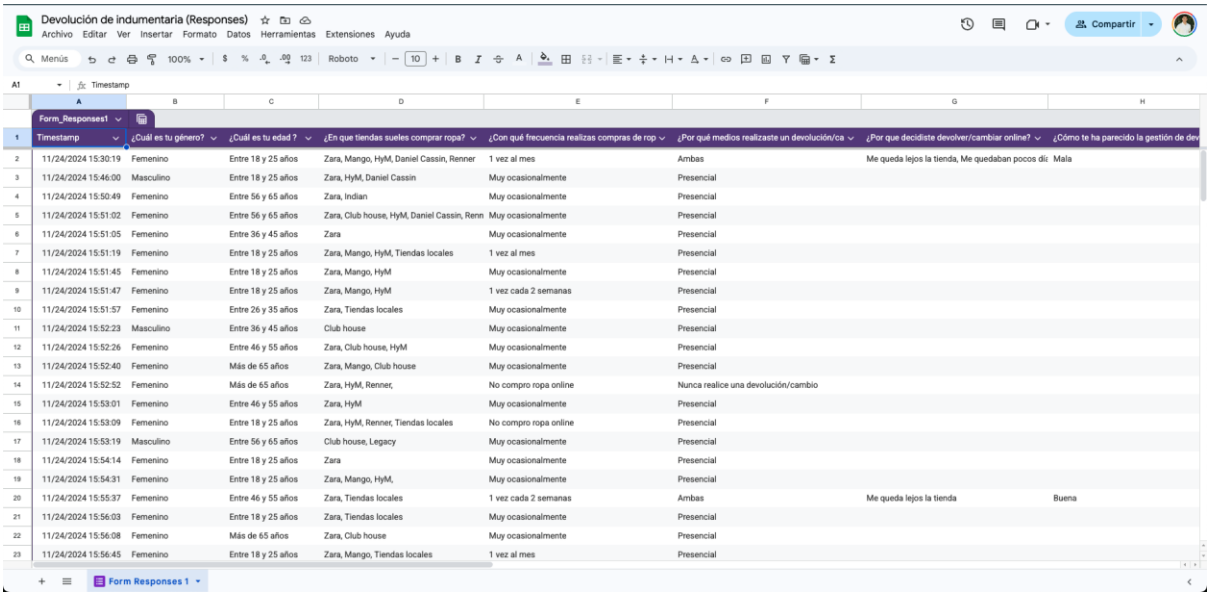


Ilustración 93 - Encuesta de usuarios pt.2

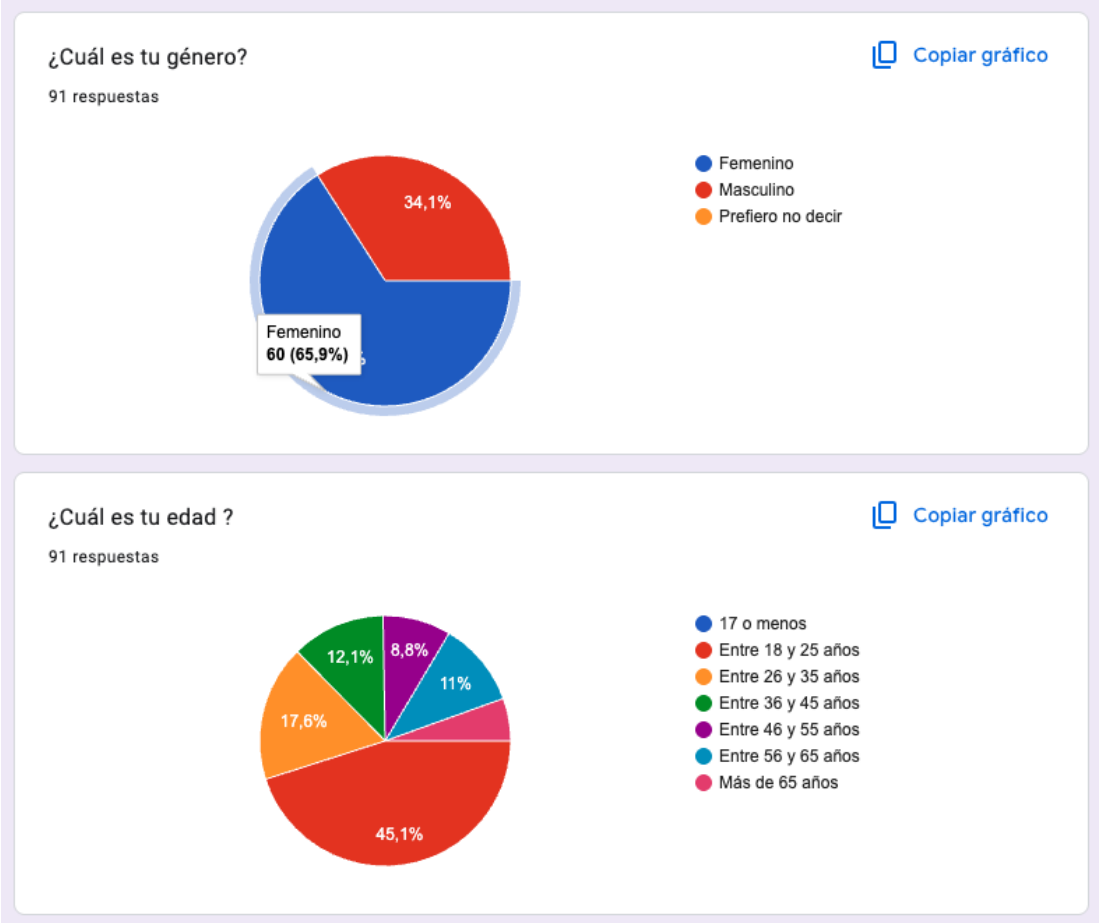


Ilustración 94 - Encuesta de usuarios pt.3

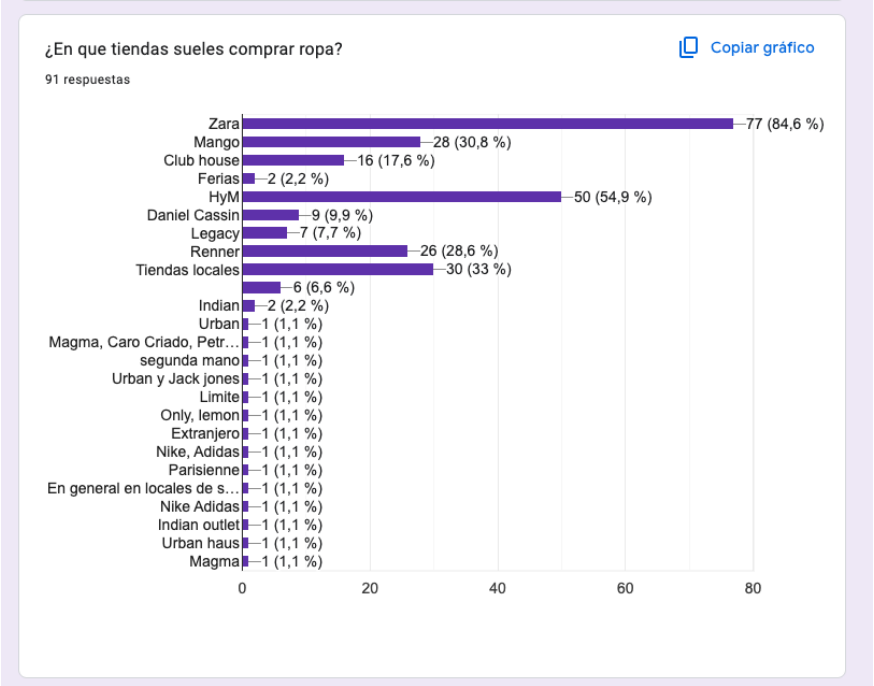


Ilustración 95 - Encuesta de usuarios pt.4

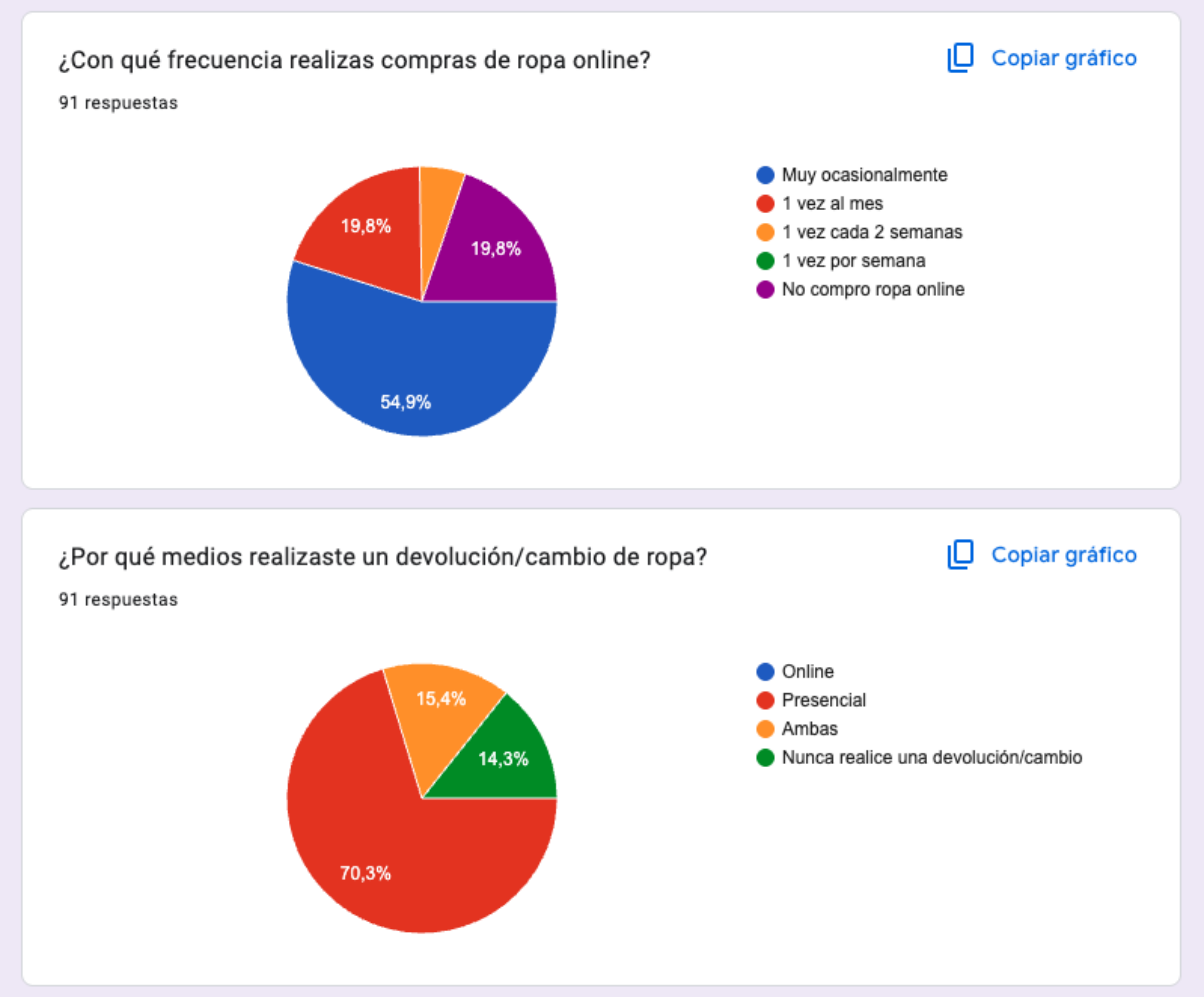


Ilustración 96 - Encuesta de usuarios pt.5

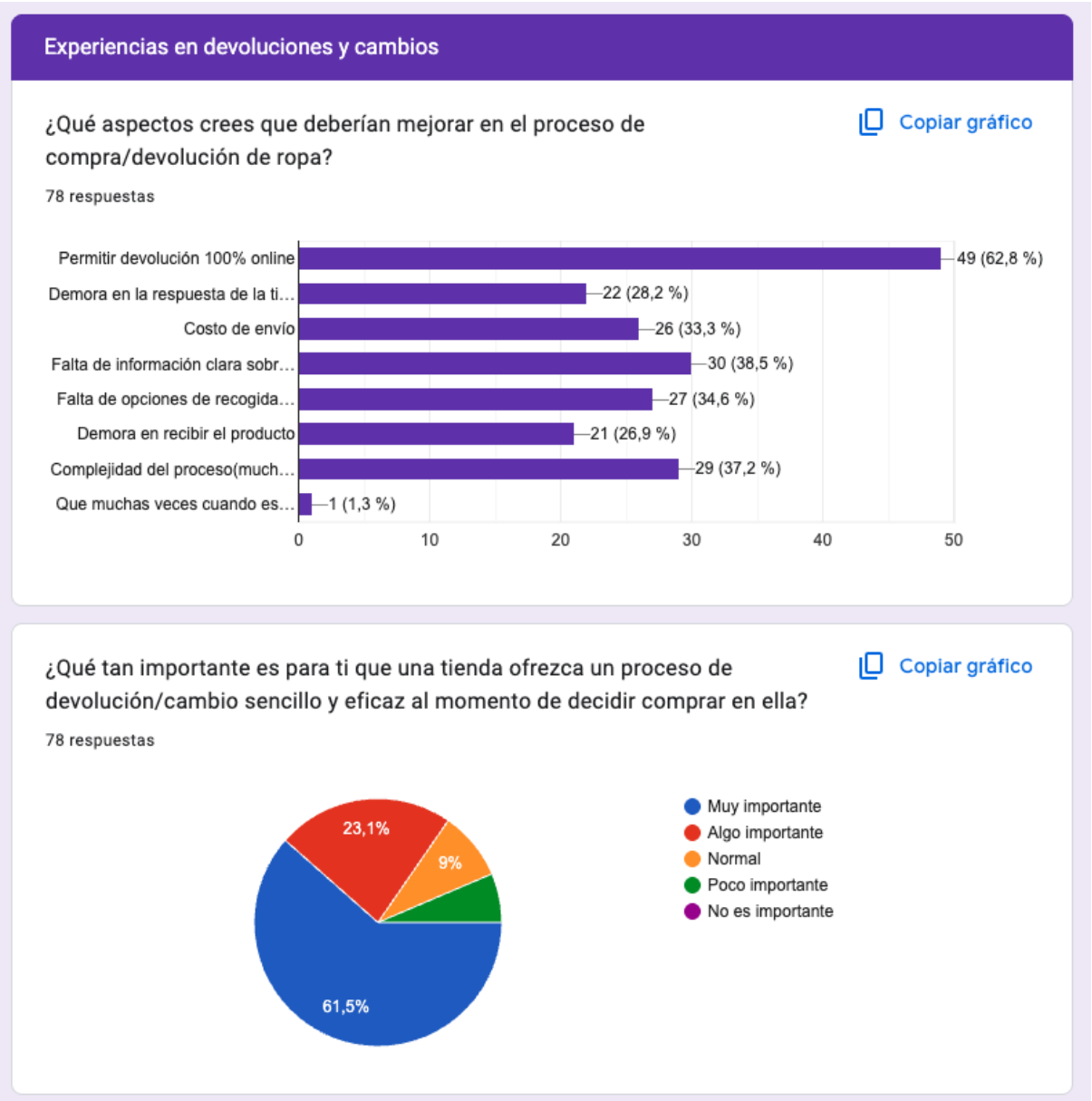


Ilustración 97 - Encuesta de usuarios pt.6

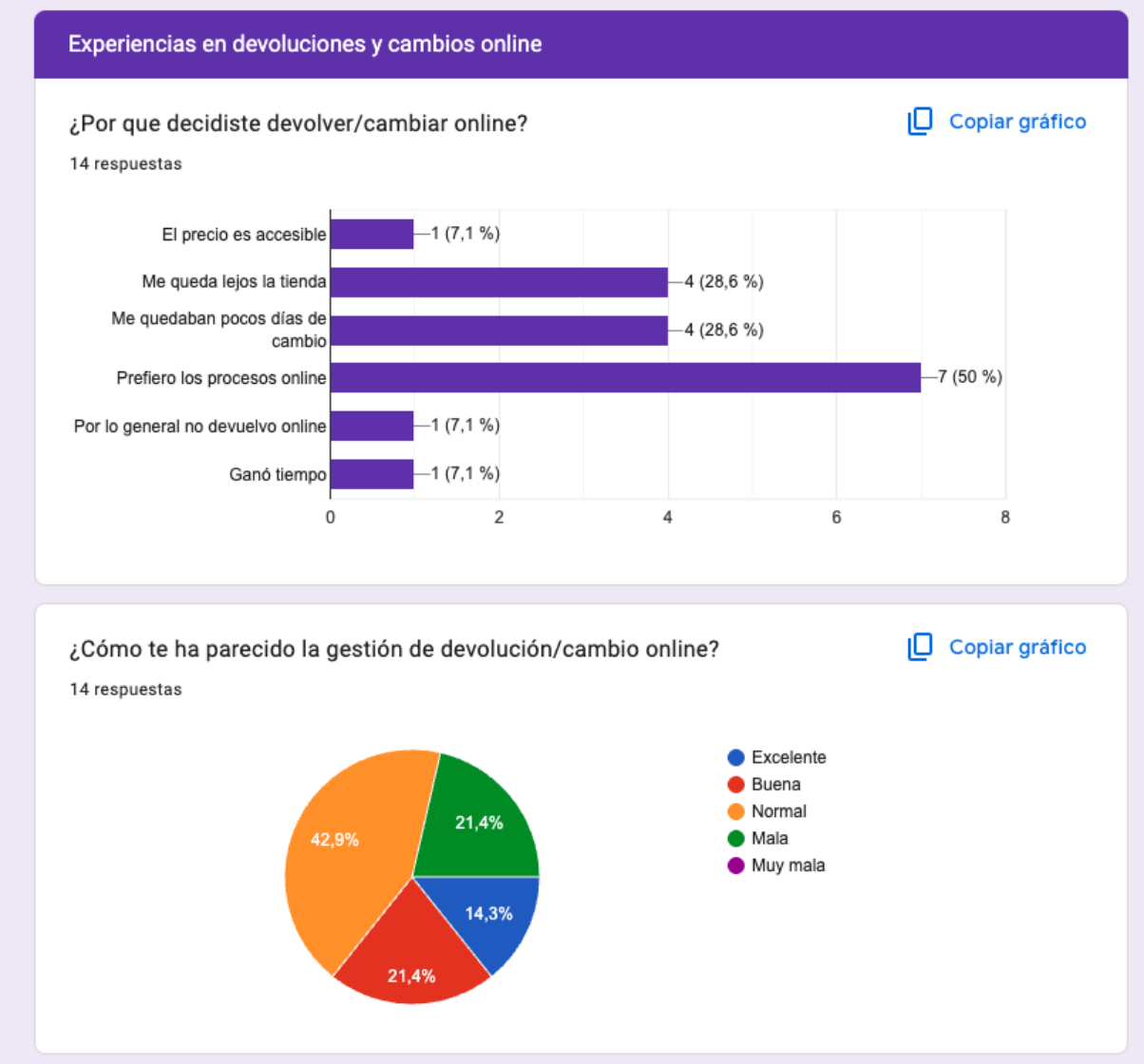


Ilustración 98 - Encuesta de usuarios pt.7

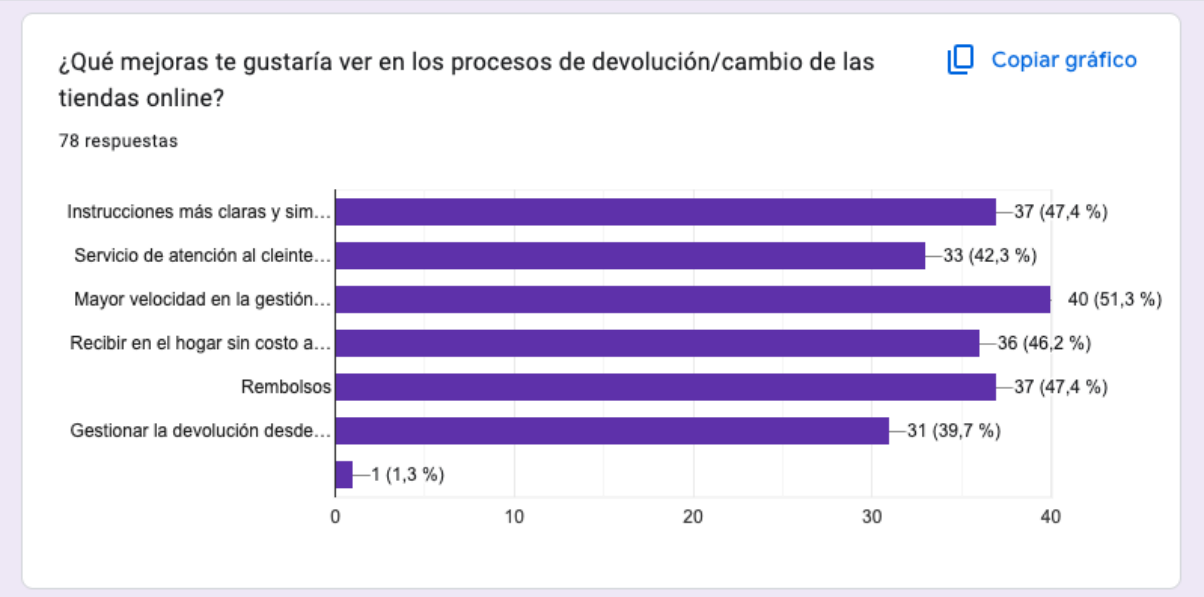


Ilustración 99 - Encuesta de usuarios pt.8

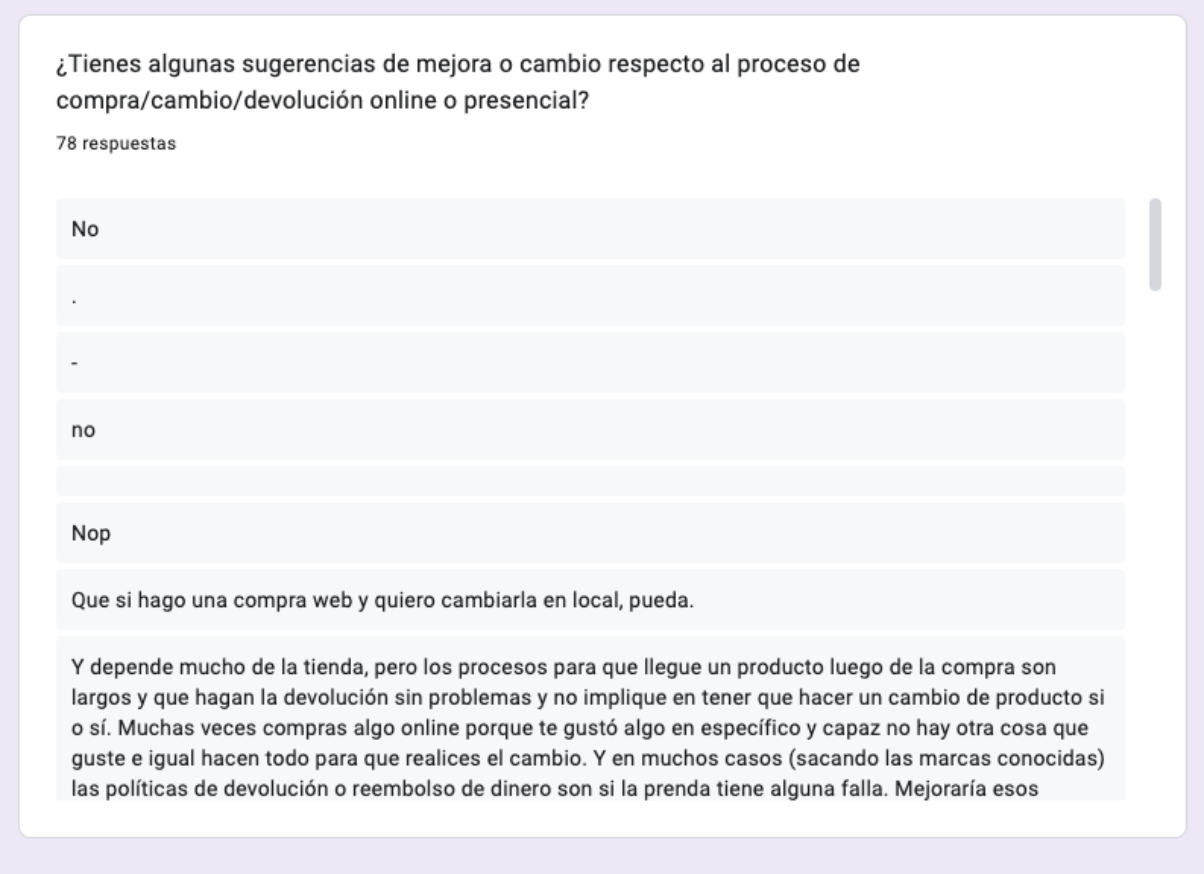
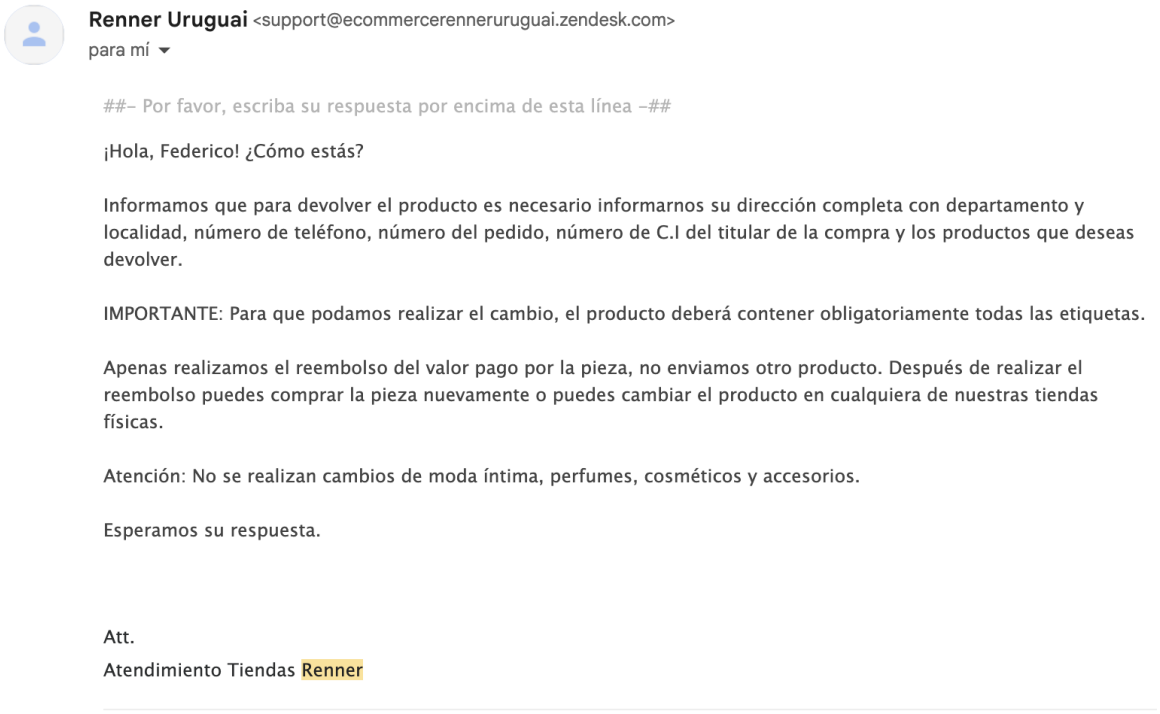


Ilustración 100 - Encuesta de usuarios pt.9

12.9.2. Email de devolución Renner



Renner Uruguai <support@ecommercerenneruruguai.zendesk.com>
para mí

##- Por favor, escriba su respuesta por encima de esta línea -##

¡Hola, Federico! ¿Cómo estás?

Informamos que para devolver el producto es necesario informarnos su dirección completa con departamento y localidad, número de teléfono, número del pedido, número de C.I del titular de la compra y los productos que deseas devolver.

IMPORTANTE: Para que podamos realizar el cambio, el producto deberá contener obligatoriamente todas las etiquetas.

Apenas realizamos el reembolso del valor pago por la pieza, no enviamos otro producto. Después de realizar el reembolso puedes comprar la pieza nuevamente o puedes cambiar el producto en cualquiera de nuestras tiendas físicas.

Atención: No se realizan cambios de moda íntima, perfumes, cosméticos y accesorios.

Esperamos su respuesta.

Att.
Atendimiento Tiendas Renner

Ilustración 101 - Proceso de devolución en Renner Uruguay

12.9.3. Preguntas para entrevista con tiendas

1. ¿Cómo manejan actualmente las devoluciones?
2. ¿Tienen políticas de devolución específicas? ¿Cómo las comunican a los clientes?
3. ¿Saben qué factores suelen causar más devoluciones?
4. ¿Qué herramientas o sistemas utilizan para registrar ventas e interacciones con clientes?
5. ¿Cómo evalúan la satisfacción del cliente tras una devolución?
6. ¿Qué tipo de comentarios o quejas reciben respecto a las devoluciones?
7. ¿Cómo afecta la falta de seguimiento en devoluciones al negocio?
8. ¿Qué beneficios creen que podrían obtener con un sistema para trackear devoluciones?
9. ¿Qué barreras ven para implementar un sistema de seguimiento de devoluciones?
10. Si tuvieran un sistema que relacione devoluciones con ventas, ¿qué decisiones estratégicas tomarían?
11. ¿Qué cadeterías utilizan en los envíos?

12.9.4. User Journey Map del proceso actual de devolución

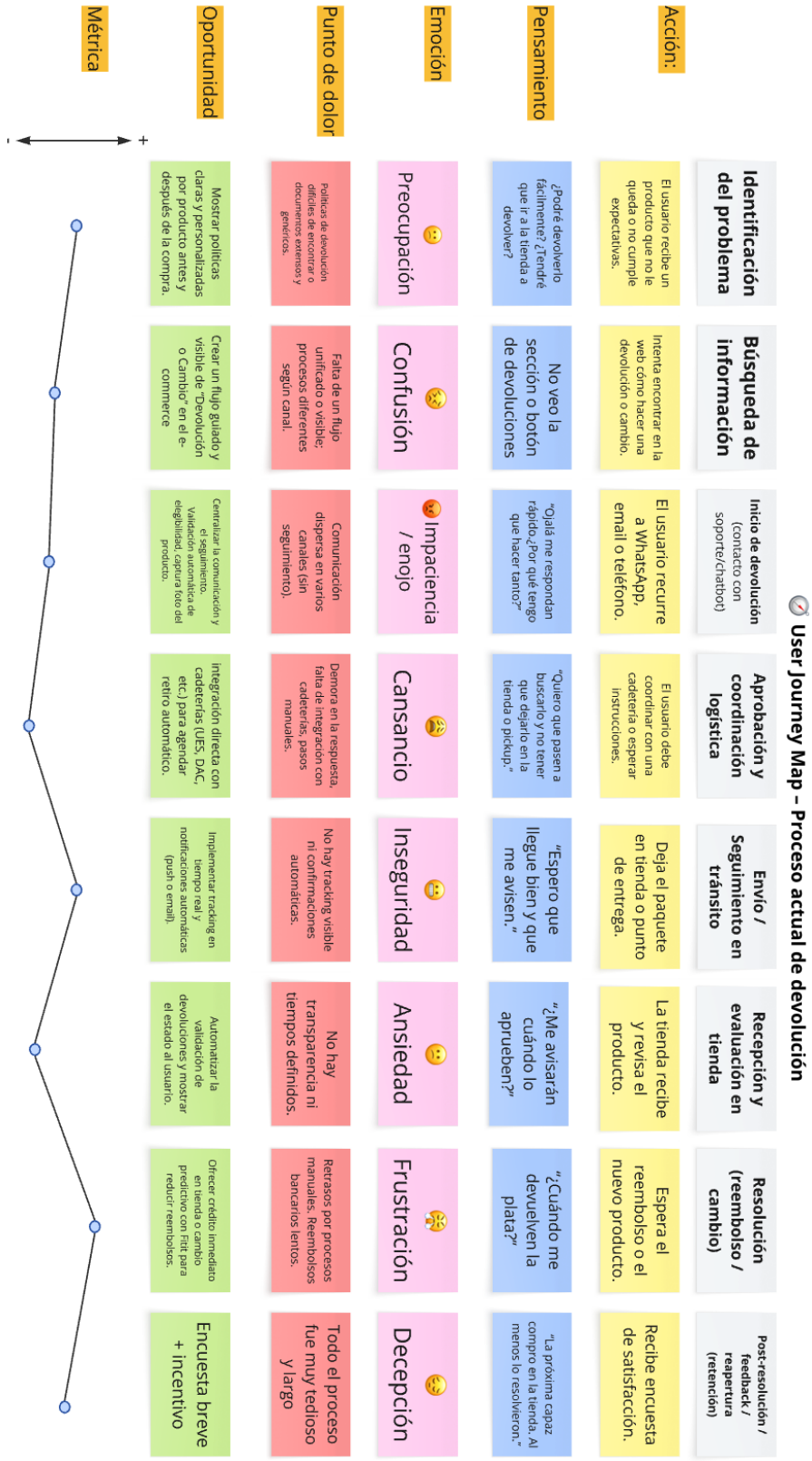


Ilustración 102 - Evidencia de User Journey Map del proceso actual de devolución

12.9.5. Saturar

https://miro.com/app/board/uXjVJ_DIzpk=/

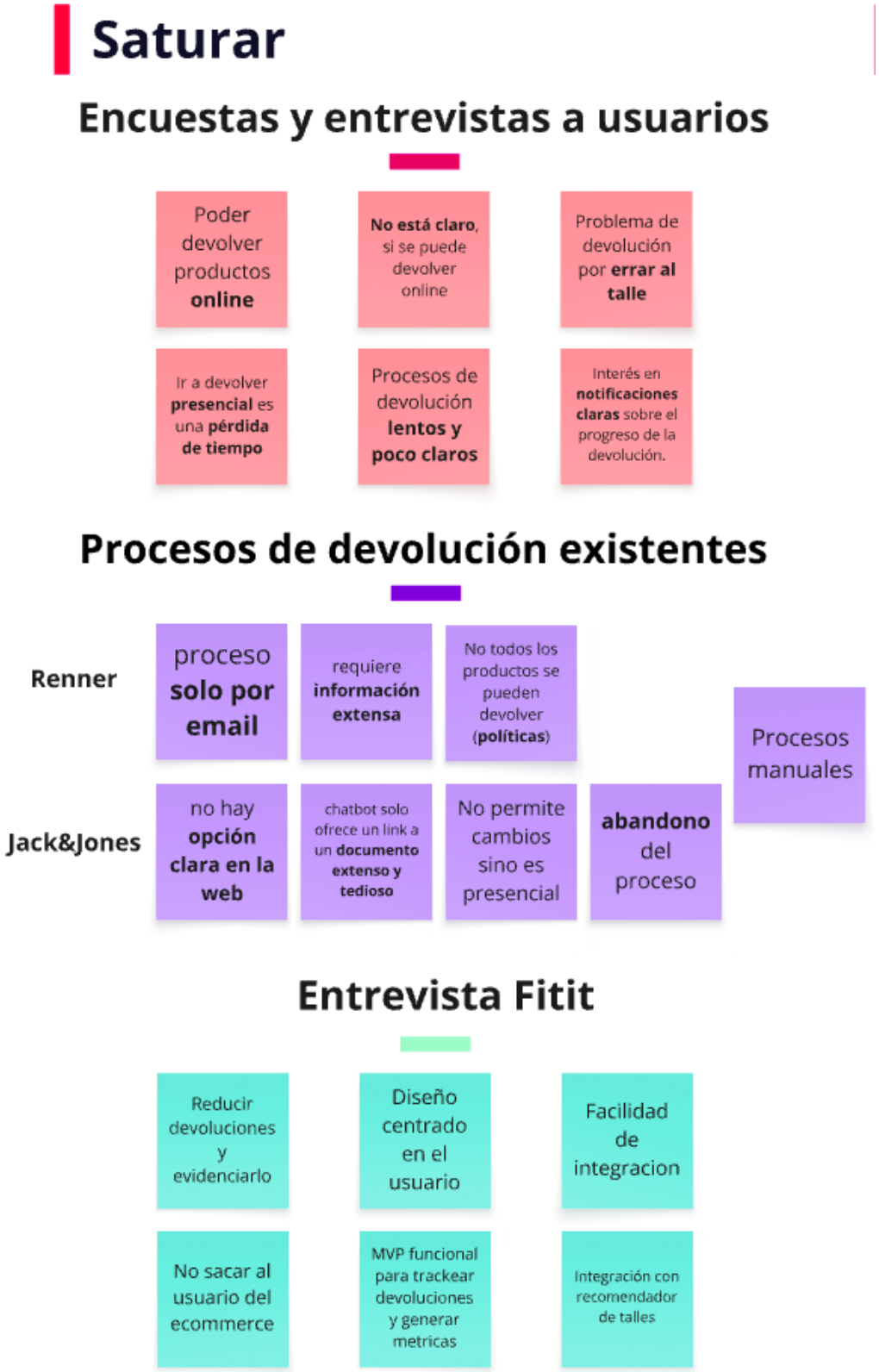


Ilustración 103 - Evidencia de saturar los insights del relevamiento

Saturar

Entrevista UES

No se diferencia un envío por compra o devolución

Algunos clientes reciben **dos etiquetas**: una para envío y otra opcional para devolución.

Es relevante **agendar envíos con anticipación** para clientes de alto volumen

Integración con sistemas de cadetería es clave para **optimizar tiempos y reducir errores.**

Permiten integraciones con ecommerce por **API, manuales o plugin**

Integración con terceros por **API**

Entrevistas a Tiendas

No registran **motivos de devoluciones** ni evalúan satisfacción

Falta de seguimiento genera problemas operativos y pérdidas de información.

Necesidad de **potenciar cambios sobre devoluciones** para fidelización

Uso de WhatsApp, email y llamadas para gestionar devoluciones (manual, lento).

Integración con cadeterías y visibilidad del estado del producto es un **punto clave**.

Las devoluciones presentan un costo alto

Tiempo de empleados dedicado es muy alto

No hay métricas de postventa.

Cada tienda usa cadeterías distintas.

Multiples facturas en el cambio

Políticas distintas según producto

Costos asociados al envío

Ilustración 104 - Evidencia de saturar los insights del relevamiento

Ilustración 105 - Evidencia de saturar los insights del relevamiento

12.9.6. Agrupar

Ilustración 106 - Evidencia de agrupar los insights del relevamiento

Agrupar

Ilustración 107 - Evidencia de agrupar los insights del relevamiento

Ilustración 108 - Evidencia de agrupar los insights del relevamiento

12.9.7. User Personas

Ilustración 109 - Evidencia de User persona 1

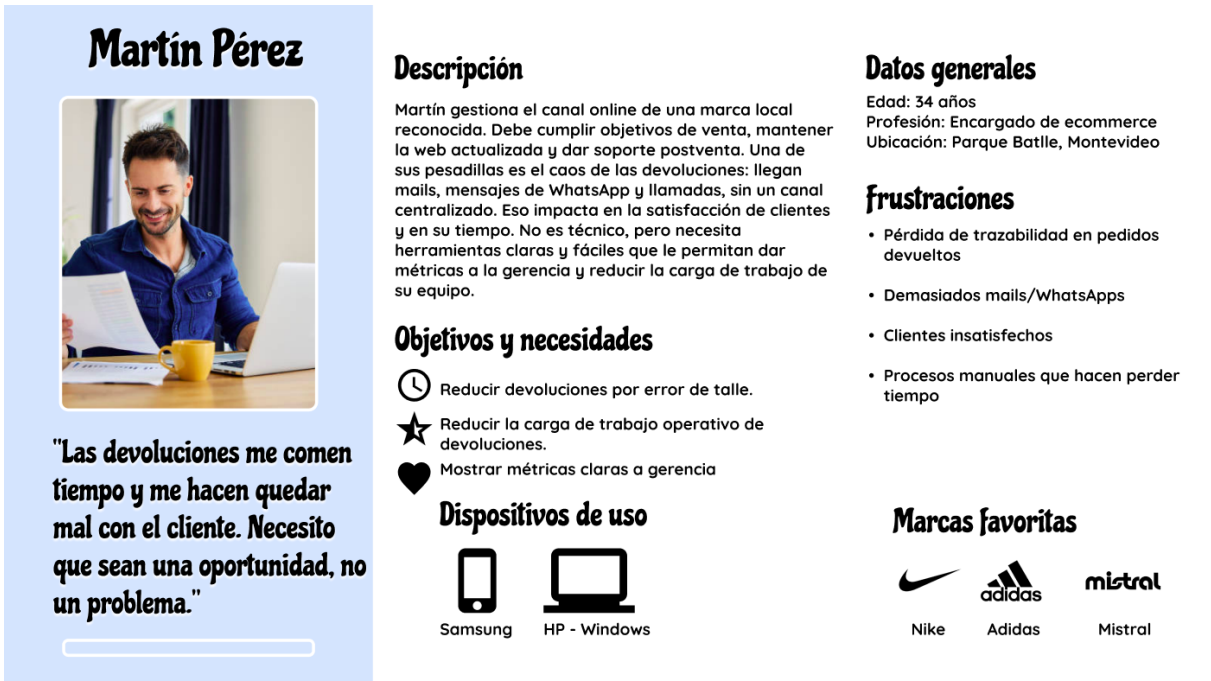


Ilustración 110 - Evidencia de User persona 2

12.9.8. Votación para priorizar ideas

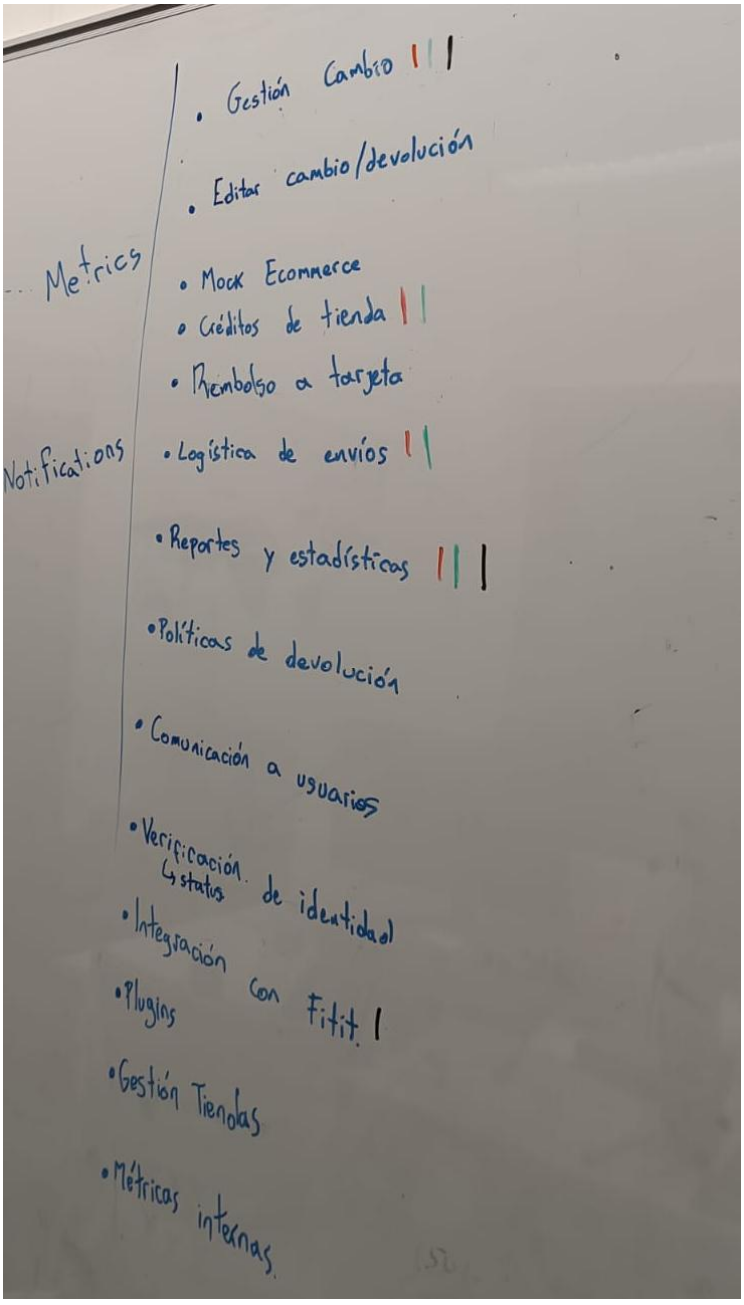


Ilustración 111 - Evidencia de votación para priorizar ideas

12.9.9. Prototipo de baja fidelidad

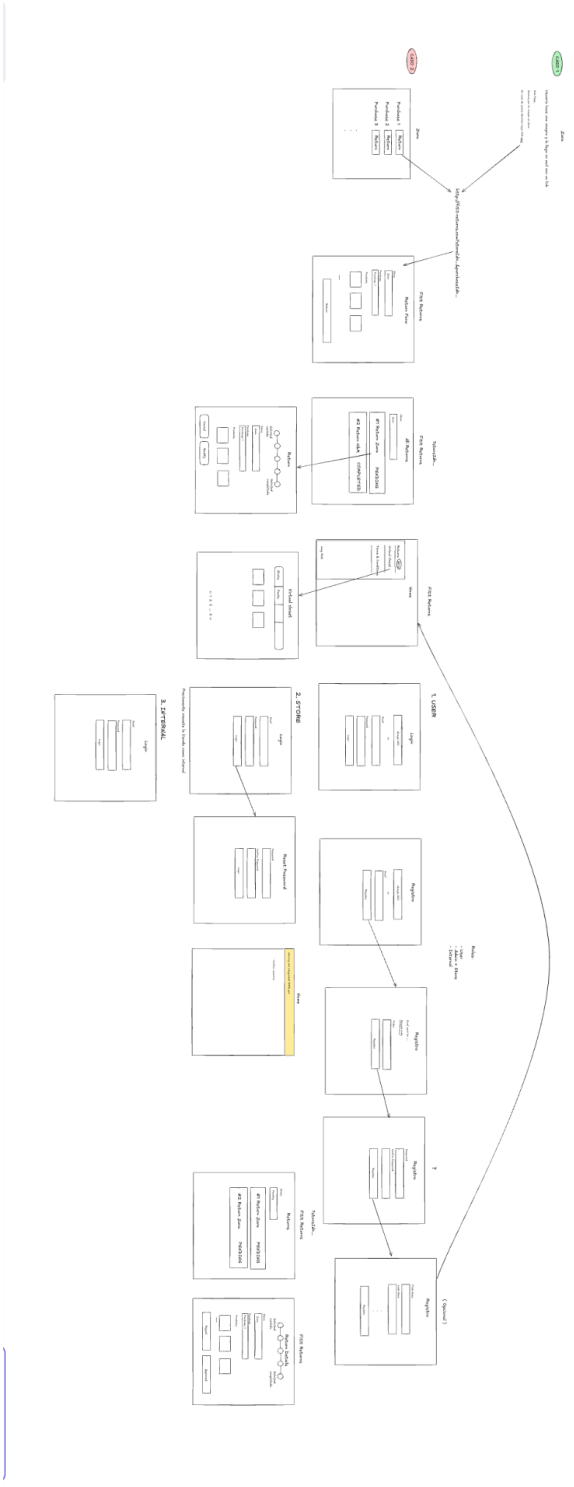


Ilustración 112 - Prototipo de baja fidelidad

12.9.10. Prototipos de alta fidelidad

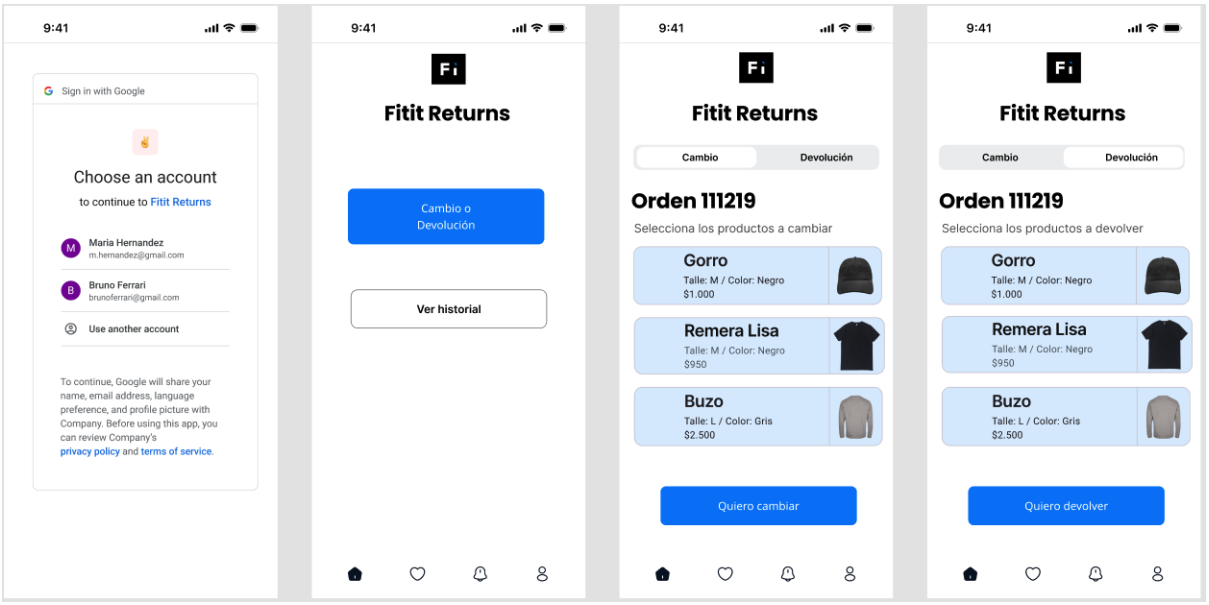


Ilustración 113 - Prototipos de alta fidelidad

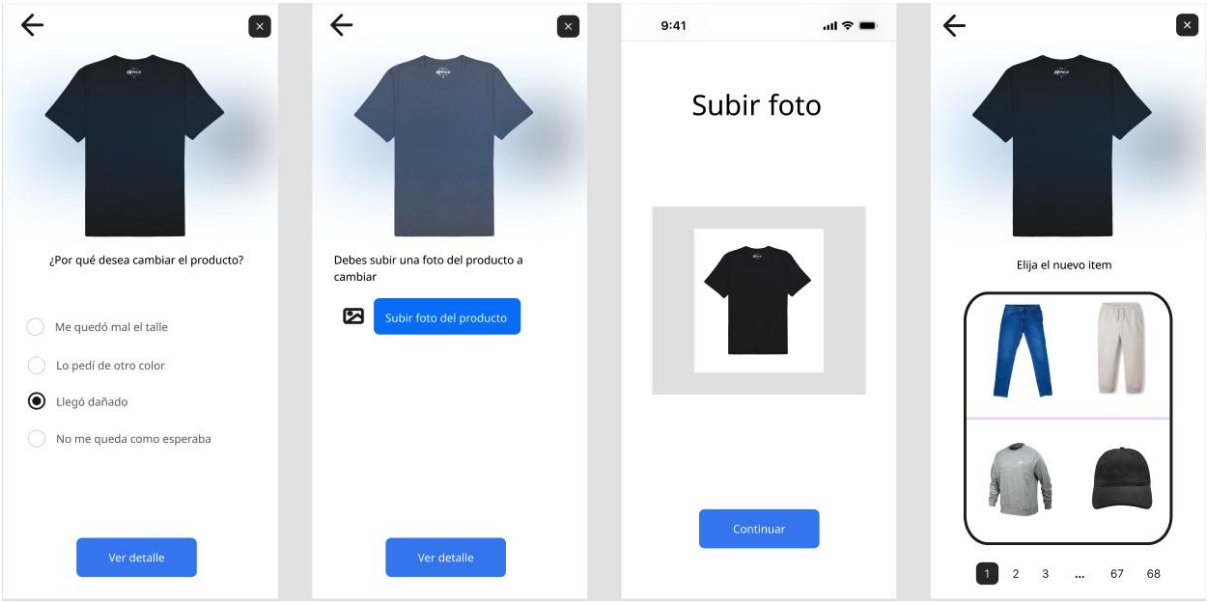


Ilustración 114 - Prototipos de alta fidelidad 2

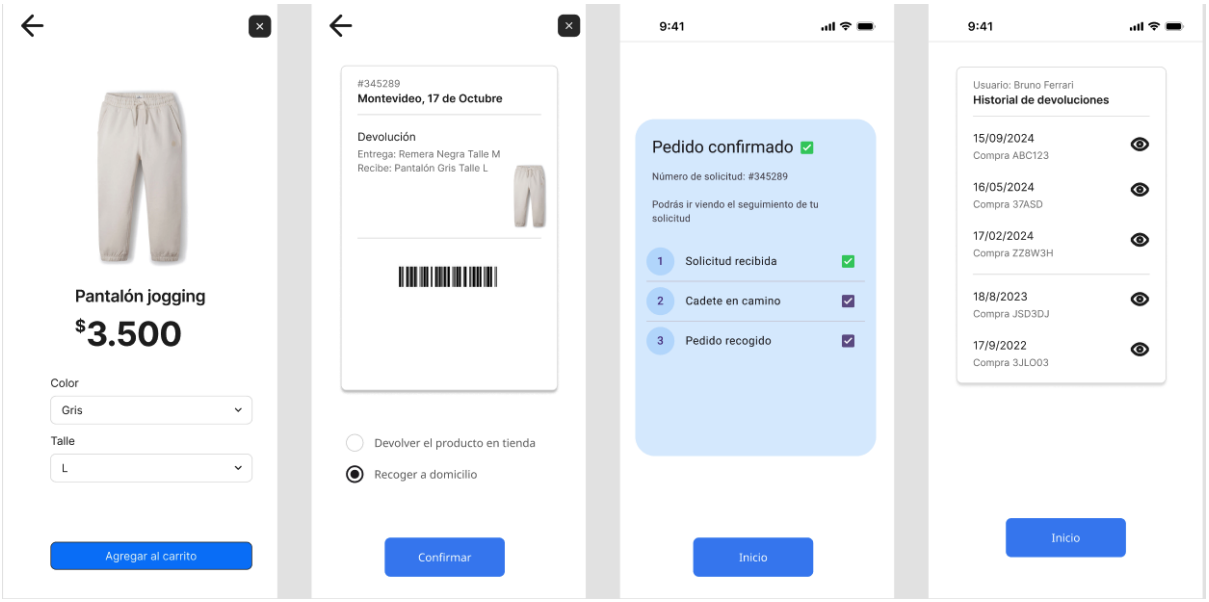


Ilustración 115 - Prototipos de alta fidelidad 3

12.10. Épicas

Épica	Descripción
E1–Gestionar devoluciones y cambios	Permitir al usuario iniciar, seguir y completar solicitudes de devolución o cambio de manera online y guiada.
E2–Panel de control para tiendas	Ofrecer a las tiendas un panel para visualizar, aprobar o rechazar solicitudes y consultar métricas operativas.
E3–Integración cadeterías	Conectar la plataforma con servicios de cadetería para automatizar retiros, entregas de productos devueltos y seguimiento de la devolución.
E4–Gestión de políticas de devolución	Permitir a las tiendas configurar reglas personalizadas según producto, categoría o tipo de operación.
E5–Integración con *e-commerce*	Asegurar la integración entre Fitit Returns y el *e-commerce*
E6–Gestión de créditos	Incorporar la posibilidad de emitir créditos de tienda automáticos según la política definida.
E7-Gestión de Tiendas	Permitir el alta, modificación y administración de tiendas dentro del sistema, incluyendo la configuración de políticas, integraciones y permisos de usuario.
E8-Gestión de Usuarios	Administrar los diferentes tipos de usuarios, controlando roles, accesos y permisos según sus responsabilidades.
E9-Gestión de ChangeLog	Implementar un sistema de registro de cambios que documente la evolución de los estados de devoluciones y

	cambios, garantizando trazabilidad y transparencia en el proceso.
E10-Integración con Fitit	Incorporar el recomendador de talles de Fitit dentro del flujo de cambio para sugerir talles correctos y reducir errores, además de cruzar datos que permitan medir su impacto en la disminución de devoluciones
E11-Notificaciones	Desarrollar un sistema de notificaciones que mantenga informados a usuarios y tiendas sobre el estado de las solicitudes, aprobaciones, retiros, entregas y emisión de créditos.

Tabla 20 - Épicas

12.11. Resumen de la ingeniería de requerimientos

Fase de ingeniería de requerimientos	**Etapa de *Design Thinking* de apoyo**	**Actividades realizadas**	**Propósito / Resultados**	**Artefactos generados**
1. Relevamiento de Requerimientos	**Empatizar**	- Entrevistas a dueños de tienda - Encuesta a usuarios de *e-commerce* - Entrevista con UES - Revisión de procesos actuales en tiendas - Reuniones con Fitit	Comprender el contexto, las frustraciones, necesidades y comportamientos de los distintos actores del ecosistema de devoluciones y cambios.	- *User Journey Map* - *Insights* - Síntesis de entrevistas
2. Análisis de Requerimientos	**Definir**	- Ingeniería inversa de soluciones existentes - Actividades de síntesis: Saturar, Agrupar, Identificar perfiles - Elaboración de *User Personas*	Sintetizar la información recolectada para identificar los problemas clave y definir claramente el alcance del sistema.	- Identificación de actores y perfiles - Declaración del problema - User Personas - Lista preliminar de necesidades y objetivos
3. Especificación de Requerimientos	**Idear**	- Reuniones de *brainstorming* y priorización de ideas (votación) - Diagramas de flujo de proceso - Técnica *SCAMPER* para pensar mejoras - Definición de alternativas de integración - Modelado de casos de uso y flujos	Generar, priorizar y documentar soluciones que respondan a los requerimientos detectados.	- Épicas - Historias de usuario - Criterios de aceptación - Diagramas de flujo

Fase de ingeniería de requerimientos	Etapa de *Design Thinking* de apoyo	Actividades realizadas	Propósito / Resultados	Artefactos generados
4. Validación de Requerimientos	**Prototipar / Probar**	- Revisión de requerimientos junto a Fitit - Creación de wireframes y mockups del flujo de devolución - *Feedback* iterativo del equipo y validación cruzada con insights - Revisión de trazabilidad entre necesidades y funcionalidades	Asegurar que los requerimientos reflejan correctamente las necesidades de los usuarios y son factibles para el *MVP*.	- Prototipo de flujo de devolución - Matriz de trazabilidad - *Feedback* documentado de *stakeholders*

Tabla 21 - Resumen de ingeniería de requerimientos

12.12. Proceso de gestión de cambios

Paso	Descripción
1. Registro del cambio	Cualquier nueva necesidad, ajuste o cambio detectado se documenta en el *backlog*. Se asigna un identificador único y se detalla su descripción, origen y motivación.
2. Evaluación del impacto	Se analiza cómo afectará la funcionalidad al sistema existente. Se estima esfuerzo y recursos necesarios y se determina prioridad relativa frente a otros elementos del *backlog*.
3. Aprobación del cambio	El equipo y el cliente validan la relevancia y el valor de la funcionalidad. Se decide si se incorpora al *backlog* inmediato, se pospone o se descarta.
4. Planificación e implementación	Se asignan tareas en Jira con responsables y fechas estimadas. Se integra en el release correspondiente, aplicando enfoque feature-driven o deadline-driven según corresponda.
5. Validación y retroalimentación	Se prueba la funcionalidad con los actores involucrados. Se ajusta según *feedback* antes de consolidar la versión final.
6. Registro final y trazabilidad	Se actualizan documentos, backlog y logs de decisiones para mantener un historial completo.

Tabla 22 - Proceso de gestión de cambios

12.13. Gráfica de descargas npm next vs. nuxt

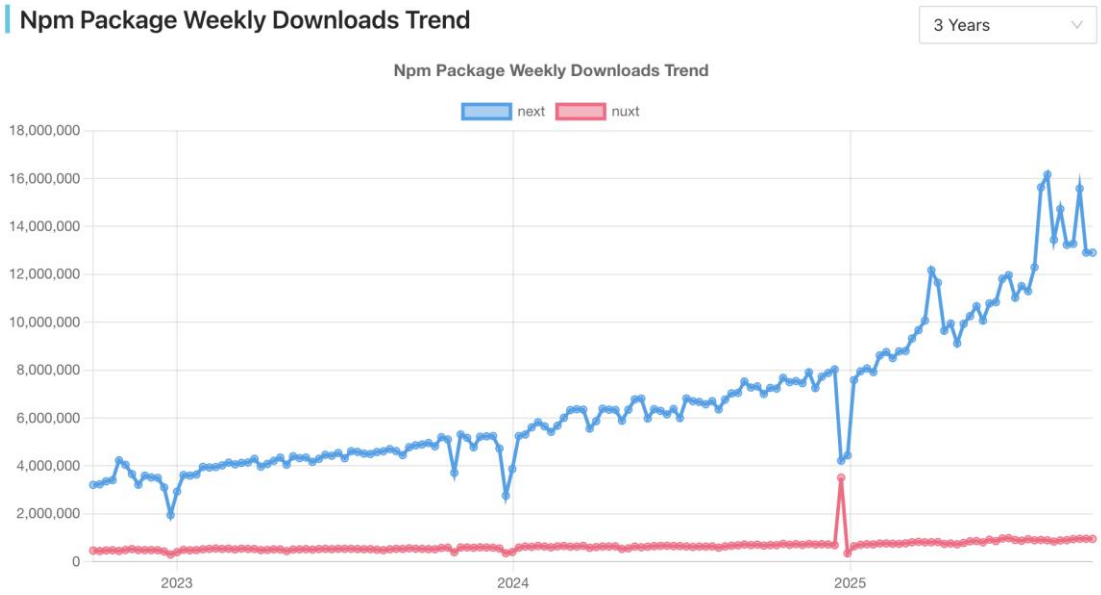


Ilustración 116 - Comparación de descargas semanales en NPM entre Next.js y Nuxt.js [32]

12.14. Tabla comparativa tecnologías Credits API

Criterio	**Express**	**Fastify**	**Koa**	**Comentario**
Madurez y comunidad	Muy alto, más de 10 años de uso, comunidad robusta y gran ecosistema de plugins	Creciente, muy activo, comunidad en expansión	Menor comunidad, más pequeño ecosistema	Express tiene el respaldo de una comunidad madura, lo que implica estabilidad y disponibilidad de soluciones ya probadas.
Rendimiento	Bueno	Muy alto (la industria lo posiciona como uno de los *frameworks* más rápidos de Node.js)	Alto	Fastify ofrece mejor *performance* en pruebas de throughput y latencia, especialmente en *APIs* de alto volumen.
Curva de aprendizaje	Baja	Media	Media	Express es sencillo de aprender y tiene abundante documentación y ejemplos.

Tabla 23 - Tabla comparativa tecnologías Credits API

12.15. Plan de calidad

Fase de descubrimiento:

Actividad	Entradas	Salidas	Herramienta/Tecnología	Encargado	Participantes
Estudio del problema	*Feedback* de tiendas, análisis del dominio	Documento de alcance inicial	Reuniones con cliente, Google Docs, Meet	Todo el equipo	Equipo Fitit + actores principales
Buscar soluciones	Problema identificado	Posibles soluciones y enfoques	*Brainstorming*, prototipos	Todo el equipo	Equipo completo
Definición de roles	Perfiles de integrantes	Roles asignados (PO, SM, Devs)	Reuniones internas	Equipo	Todo el equipo
Definición del ciclo de vida	Problema del proyecto	Ciclo de vida incremental-iterativo	Reunión interna	Scrum Master	Equipo
Identificación de requerimientos	Procesos de devolución actuales	User stories y épicas	Docs y al final Jira	Ingeniero de requerimientos	Equipo + Cliente
Priorización de requerimientos	Lista de requerimientos	*Backlog* con *must/nice-to-have*	Poker Planning	Todo el equipo	Equipo completo
Estimación inicial	*Backlog* priorizado	Requerimientos estimados	Planning Poker	Todo el equipo	Equipo
Definición de alcance y *releases*	Requerimientos estimados	*Roadmap* + Plan de *releases*	Figma	Product Owner	Equipo
Validación de alcance	Plan de *release*	*Backlog* validado	Reuniones con cliente	Todo el equipo	Cliente + Equipo

Prototipado	Requerimientos validados	*Mockups* y prototipos iniciales	Figma	Dev team	Equipo + Cliente
Análisis de tecnologías	Requerimientos no funcionales	Selección de *stack* (Firebase, React.js, etc.)	Research + docs oficiales + restricciones	Arquitecto	Dev team
Diseño de arquitectura	Requerimientos y restricciones	Diagramas de arquitectura y flujos	Draw.io, Mermaid, Escalidraw	Arquitecto	Equipo
Validación de arquitectura	Documento de arquitectura	Arquitectura revisada	Revisión con expertos	Arquitecto	Expertos + Equipo

Tabla 24 - Plan de calidad

Fase de desarrollo:

Actividad	**Entradas**	**Salidas**	**Herramienta/ Tecnología**	**Encargado**	**Participantes**
Sprint Planning	Product Backlog	Sprint Backlog	Jira	Scrum Master	Equipo completo
Codificación	Historias priorizadas	Código fuente	VSCode, Firebase *SDK*, React.js	Dev team	Dev team
Revisión de código	Código en repositorio	Código aprobado	*Pull requests* en GitHub	Dev team	Dev team
Testing unitario	*Cloud Functions*	Pruebas automatizadas	Jest, Firebase Emulator	Dev team	Dev team
Testing de integración	Flujos cliente-tienda	Validaciones correctas	*Firebase Emulator, Shopify*	Dev team	Dev team
Testing manual	Prototipos y *releases*	*Feedback* de usabilidad	Navegación del sistema	Todo el equipo	Equipo

Daily/Weekly standups	Sprint Backlog	Avances y bloqueos claros	Slack, WhatsApp	Scrum Master	Equipo
Sprint Review	Incremento del producto	*Feedback* de cliente	Demo en Google Meet	Product Owner	Equipo + Cliente
Sprint Retrospective	Sprints completados	Mejoras de proceso	Reunión virtual, Miro	Scrum Master	Equipo
Métricas de calidad	*Releases* parciales	*Burndown chart*, reportes de horas	Clockify, Jira, Excel	Scrum Master	Equipo

Tabla 25 - Fase de desarrollo

Fase de documentación:

Actividad	Entradas	Salidas	Herramienta /Tecnología	Encargado	Participantes
Redacción incremental	Avances de sistema	Capítulos preliminares	Google Docs	Equipo	Equipo
Documentación técnica	Código final	Manual técnico en el repositorio (instalación)	Markdown, Docs, Draw.io	Dev team	Dev team
Revisión cruzada	Capítulos preliminares	Informe corregido	Google Docs	Equipo	Equipo

Tabla 26 - Fase de documentación

Entrega final:

Actividad	Entradas	Salidas	Herramienta/ Tecnología	Encargado	Participantes
Validación final	Sistema completo	Sistema estable	Firebase Hosting	Dev team	Cliente + Equipo
Checklist de calidad	Backlog	Validación contra criterios	Jira	Scrum Master	Equipo

Entrega de producto	Sistema y docs	Deploy final + documentación	Firebase + Google Docs	Todo el equipo	Equipo
Presentación	Producto terminado	*Feedback* de tutor/tribunal	Slides	Scrum Master	Equipo

Tabla 27 - Entrega final

12.16. Definición de estándares de codificación

Estructura y modularización

Estándar	Descripción	Ejemplo en el repositorio	Justificación
Monorepo	Repositorio único con varios paquetes independientes: fitit-store, fitit-user, fitit-backoffice, fitit-cloud-functions, fitit-credits-api, y shared.	/fitit-cloud-functions/common	Permite mantener coherencia en las dependencias y simplifica la CI/CD.
Separación de responsabilidades	Cada módulo agrupa una responsabilidad principal (*frontend, backend*, *API*, lógica compartida).	/fitit-cloud-functions/user	Facilita mantenimiento y escalabilidad.
Helpers reutilizables	Funciones y componentes comunes centralizados	@shared/web/lib/components	Evita duplicación y mejora la trazabilidad.

Tabla 28 - Estructura y modularización

Estándares de validación y contratos

Estándar	Descripción	Herramienta / Fuente	Justificación
Schemas de validación	Validación estricta de los *payloads* con Yup, aplicando *stripUnknown* para eliminar campos no esperados.	Yup	Mejora la seguridad y robustez de las *Cloud Functions*.
Autenticación y roles	Validación de permisos dentro de las *Cloud Functions* según	Firebase Auth	Previene accesos indebidos y garantiza una correcta

	el tipo de usuario y el contexto de la operación.		segregación de roles.
Firestore rules	Definición de reglas de seguridad que determinan qué usuarios pueden leer o escribir en determinadas colecciones de la base de datos.	Firestore Security Rules	Garantiza integridad y confidencialidad de los datos, evitando accesos o modificaciones no autorizadas.
Storage rules	Configuración de permisos para controlar el acceso a archivos almacenados, como imágenes de productos o comprobantes subidos por los usuarios durante una devolución.	Firebase Storage Rules	Asegura que los archivos sensibles solo puedan ser accedidos por usuarios autenticados y con los permisos correspondientes.

Tabla 29 - Estándares de validación y contratos

Estándares de manejo de errores

Estándar	Descripción	Ejemplo	Justificación
Errores semánticos	Uso de HttpsError con códigos específicos (not-found, invalid-argument, etc.).	*Cloud Functions*	Facilita el *debugging* y el entendimiento del flujo de errores.
Normalización de errores	Centralización de un manejador común para respuestas de error.	/shared/errorHandler.js	Mejora la consistencia en las APIs y simplifica el soporte.

Tabla 30 - Estándares de manejo de errores

Estándares de seguridad y configuración

Estándar	Descripción	Herramienta	Justificación

Gestión de secretos	Uso de .env y .env.example por paquete y también para los distintos ambientes, sin incluirlos en control de versiones. Para las github actions usar *GitHub Action Secrets*	Dotenv, cross-env y *GitHub Action Secrets*	Mantiene seguridad de claves sensibles y cross env permite tener muchos envs para los distintos ambientes
Service accounts separadas	Archivos distintos para entornos *dev* y *staging*.	Firebase Admin SDK	Evita errores entre entornos.
Roles y permisos	Validación explícita en handlers y Firestore Rules.	Firebase Auth + *Emulator Suite*	Controla accesos y protege la información.

Tabla 31 - Estándares de seguridad y configuración

Estándares de pruebas

Estándar	Descripción	Herramienta	Justificación
Unit tests	Cobertura completa del flujo de devoluciones.	Jest	Detección temprana de defectos.
Seeder	Datos de prueba consistentes y anonimizados para *local/staging*.	Script Node.js + Firebase Emulator (import/export).	Reflejan casos reales (tiendas, pedidos, políticas), garantizan reproducibilidad y reducen la inestabilidad de las pruebas.
E2E tests	Simulación de flujos completos de usuario.	Cypress	Garantiza la usabilidad del sistema.
Testing de carga	Medición de *performance* bajo alta concurrencia.	K6	Valida la estabilidad del sistema.

CI/CD testing	Ejecución automática de tests antes de *merge*.	*GitHub Actions*	Previene errores en ambiente pre-productivo.

Tabla 32 - Estándares de pruebas

Estándares de despliegue y documentación

Estándar	Descripción	Herramienta	Justificación
Documentación técnica	*README.md* en cada módulo con instrucciones de instalación y uso.	GitHub	Facilita la incorporación de nuevos desarrolladores y el entendimiento del cliente.
***Scripts* NPM unificados**	Comandos estándar (npm run test, npm run deploy, etc.).	NPM	Simplifica la ejecución y CI/CD.
Integración continua	Flujos automáticos con GitHub Actions para *builds*, *linters* y *tests*.	*GitHub Actions*	Asegura la calidad continua del código.

Tabla 33 - Estándares de despliegue y documentación

12.17. Configuración y aplicación de estándares

Ilustración 117 - Configuración de branch linter

Ilustración 118 - Configuración de prettier

Ilustración 119 - Configuración de code linter

12.18. Casos de pruebas funcionales

Usuario

ID	Descripción	Contexto	Pasos	Resultado esperado
CFU1	Iniciar devolución desde la tienda online	Una orden dentro del plazo permitido por la política	1. Acceder al formulario de devoluciones desde la página del comercio. 2. Visualizar productos elegibles y no elegibles. 4. Seleccionar el producto a devolver.	Se muestran los productos de la orden con su estado de elegibilidad según la política vigente.
CFU2	Completar motivo, comentario y evidencia	Producto elegible seleccionado	1. Seleccionar motivo de devolución (dañado, talle incorrecto, etc.). 2. Adjuntar imagen opcional. 3. Escribir nota o comentario. 4. Presionar “Enviar”.	La solicitud se registra correctamente con estado “Pendiente”.
CFU3	Validar producto no elegible	Producto asociado a política de liquidación o fuera de plazo	1. Ingresar número de orden con producto no elegible. 2. Revisar la lista.	El sistema indica “No elegible para devolución” y bloquea la selección.
CFU4	Subir imagen con formato no permitido	Usuario intenta subir archivo inválido	1. Intentar subir archivo > 5 MB o formato no permitido. 2. Presionar “Enviar”.	El sistema muestra mensaje de error e impide continuar.

CFU5	Visualizar confirmación tras envío del formulario	Devolución enviada exitosamente	1. Completar formulario y enviar.	Se muestra pantalla de confirmación con número de solicitud y mensaje de éxito.
CFU6	Consultar estado de devolución en seguimiento	Devolución en cualquier estado.	1. Ir al tablero de mis devoluciones. 2. Buscar con número de orden.	Se muestra el estado actual (“Pendiente”, “Aprobada”, “Rechazada”, “En entrega”) y detalles cargados.
CFU7	Ver crédito generado	Devolución aprobada por la tienda	1. Consultar estado de devolución. 2. Ver sección de crédito.	Se muestra el monto generado, fecha de creación.
CFU8	Intentar devolver producto de liquidación	Producto marcado con política de liquidación	1. Ingresar orden con artículo en liquidación. 2. Revisar elegibilidad.	El sistema muestra “Artículo en liquidación — no elegible para devolución”.
CFU9	Iniciar cambio de producto	Producto elegible según política	1. Ingresar número de orden. 2. Seleccionar producto. 3. Elegir “Cambio”. 4. Seleccionar talle o variante. 5. Enviar.	Se crea una solicitud de cambio con estado “Pendiente”.
CFU10	Validar comportamiento con varios productos seleccionados	Orden con múltiples ítems	1. Ingresar número de orden. 2. Seleccionar dos productos elegibles. 3. Completar motivos y adjuntar fotos. 4. Seleccionar tipo devolución 4. Enviar.	El sistema crea una solicitud para los productos seleccionados.

CFU11	Volver atrás en el formulario sin perder datos	Usuario completa parcialmente los pasos	1. Seleccionar producto. 2. Completar motivo y nota. 3. Presionar “Atrás”.	Los datos completados se mantienen al regresar a la pantalla anterior.
CFU12	Intentar enviar formulario incompleto	Campo obligatorio sin llenar	1. No seleccionar motivo. 2. Presionar “Enviar”.	El sistema muestra validación indicando que el motivo es obligatorio.
CFU13	Ver detalle del producto seleccionado	Producto elegible seleccionado	1. Ingresar número de orden. 2. Seleccionar producto.	Se muestra nombre, talla, color, precio y estado de elegibilidad.
CFU14	Filtrar devoluciones por estado, tienda y tipo	Todas las devoluciones	1. Ir a “Devoluciones”. 2. Aplicar filtro de estado pendiente, una tienda y un tipo (compra/dsevolución/cambio)	Se muestran únicamente las devoluciones en estado pendiente de aprobación/rechazo.

Tabla 34 - Casos de prueba funcionales de usuario

Tienda

ID	Descripción	Contexto	Pasos	Resultado esperado
CFT1	Iniciar sesión en el panel de Fitit Returns	Tienda registrada	1. Acceder al panel de Fitit Returns. 2. Ingresar credenciales válidas. 3. Presionar “Entrar”.	Ingreso exitoso al panel principal.
CFT2	Ver devoluciones pendientes	Devoluciones de clientes activas	1. Abrir la pestaña “Devoluciones”.	Se muestran todas las devoluciones.

CFT3	Revisar detalle de una devolución	Devolución pendiente	1. Seleccionar devolución.	Se muestran datos del cliente, producto, motivo, comentario e imagen.
CFT4	Aprobar devolución	Devolución válida y dentro de plazo	1. Revisar detalle. 2. Presionar “Aprobar”.	Estado pasa a “Aprobada”, se genera crédito automáticamente.
CFT5	Rechazar devolución	Devolución sin evidencia suficiente	1. Abrir solicitud. 2. Presionar “Rechazar”. 3. Indicar motivo.	Estado pasa a “Rechazada” y se guarda el motivo.
CFT6	Editar política de devolución	Política activa configurada	1. Abrir “Políticas”. 2. Editar plazos o condiciones. 3. Guardar.	La política se actualiza correctamente.
CFT7	Crear nueva política	Tienda con políticas activas	1. Ir a “Políticas”. 2. Presionar “Crear nueva”. 3. Completar datos y guardar.	Se registra la nueva política y aparece en la lista activa.
CFT8	Archivar política obsoleta	Política en uso anterior	1. Seleccionar política activa. 2. Presionar “Archivar”.	La política se mueve a la lista de archivadas.
CFT9	Ver métricas de devoluciones	Tienda con actividad registrada	1. Ir a “Dashboard”.	Se muestran estadísticas de devoluciones.
CFT10	Filtrar devoluciones por fecha	Devoluciones creadas en diferentes períodos	1. Ir a “Devoluciones”. 2. Aplicar filtro de fecha.	Se muestran únicamente las devoluciones del rango seleccionado.

CFT11	Filtrar devoluciones por estado	Devoluciones con diferentes estados	1. Ir a “Devoluciones”. 2. Aplicar filtro de estado pendiente.	Se muestran únicamente las devoluciones en estado pendiente de aprobación/rechazo.

Tabla 35 - Casos de prueba funcionales de tienda

12.19. Pruebas de carga con k6

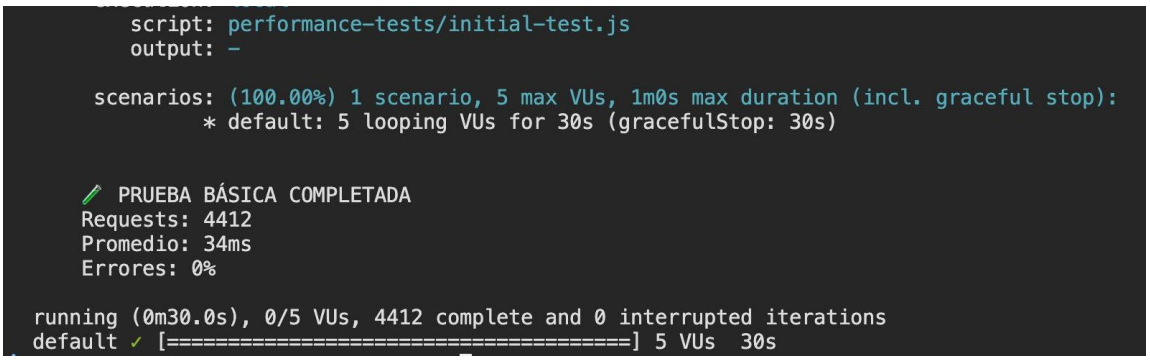


Ilustración 120 - Prueba de carga básica

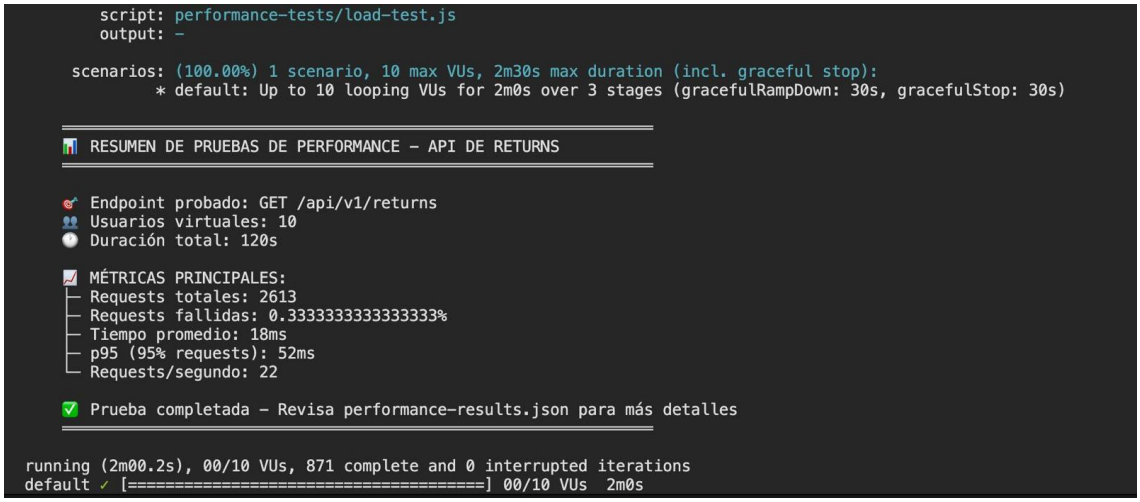


Ilustración 121 - Ejecución de prueba de carga con K6

12.20. Pruebas de usabilidad

12.20.1. Aplicación web de devolución para usuarios finales

Para la versión web utilizada por los **usuarios finales**, se realizaron pruebas con **15 participantes** representativos del público objetivo. A continuación se presentan los escenarios propuestos, junto con los resultados esperados y los obtenidos. Los valores de tiempo corresponden al promedio redondeado entre todos los usuarios y están expresados en segundos.

Iniciar una devolución

Desafío: Iniciar el flujo de devolución desde el enlace del pedido.
Objetivo: Que el usuario pueda ingresar a la página del ecommerce y seleccionar el botón para iniciar la devolución en **menos de 20 segundos**.

Métrica	Resultado esperado	Resultado obtenido
Tiempo	20	12
# de errores	0	0

Tabla 36 - Resultados de pruebas de usabilidad de “Iniciar una devolución”

Comentarios generales:

- Algunos comentaron que el texto del botón podría resaltar más visualmente.
- Ningún usuario necesitó ayuda para completar esta tarea.

Completar el formulario de devolución

Desafío: Seleccionar el producto y luego el motivo, subir una foto y escribir un comentario opcional.
Objetivo: Completar el formulario en **menos de 30 segundos**, con menos de 3 errores.

Métrica	Resultado esperado	Resultado obtenido

Tiempo	30	26
# de errores	3	1

Tabla 37 - Resultados de pruebas de usabilidad de “Completar el formulario de devolución”

Comentarios generales:

- Los usuarios comprendieron fácilmente los pasos y los campos obligatorios.
- El tiempo de carga de la foto fue percibido como adecuado.

Elegir tipo de resolución (devolución o cambio)

Desafío: Seleccionar entre solicitar una devolución o un cambio.
Objetivo: Tomar una decisión en menos de **10 segundos** sin errores.

Métrica	**Resultado esperado**	**Resultado obtenido**
Tiempo	10	5
# de errores	0	0

Tabla 38 - Resultados de pruebas de usabilidad de “Elegir tipo de resolución (devolución o cambio)”

Comentarios generales:

- Todos los usuarios entendieron la diferencia entre las dos opciones.

Realizar cambio de producto

Desafío: Completar el flujo completo de cambio, seleccionando un nuevo producto, confirmando su talle y el método de envío y confirmar.

Objetivo: Que el usuario pueda realizar el cambio en **menos de 60 segundos**, sin errores críticos.

Métrica	Resultado esperado	Resultado obtenido
Tiempo	60	49
# de errores	0	1

Tabla 39 - Resultados de pruebas de usabilidad de “Realizar cambio de producto”

Comentarios generales:

- Los usuarios comprendieron fácilmente el flujo de cambio y la visualización de créditos disponibles.
- La selección del nuevo producto fue intuitiva, algunos comentaron que sería útil mostrar un indicador de “productos habilitados para cambio” de forma más visible.
- En el paso de confirmación de talle, todos los participantes entendieron el selector y validaron correctamente la opción elegida.
- En la etapa final de selección de envío, la mayoría optó por “Punto de recogida” sin inconvenientes.
- El único error reportado fue intentar avanzar sin confirmar el método de envío, lo que generó un mensaje de validación claro y comprensible.

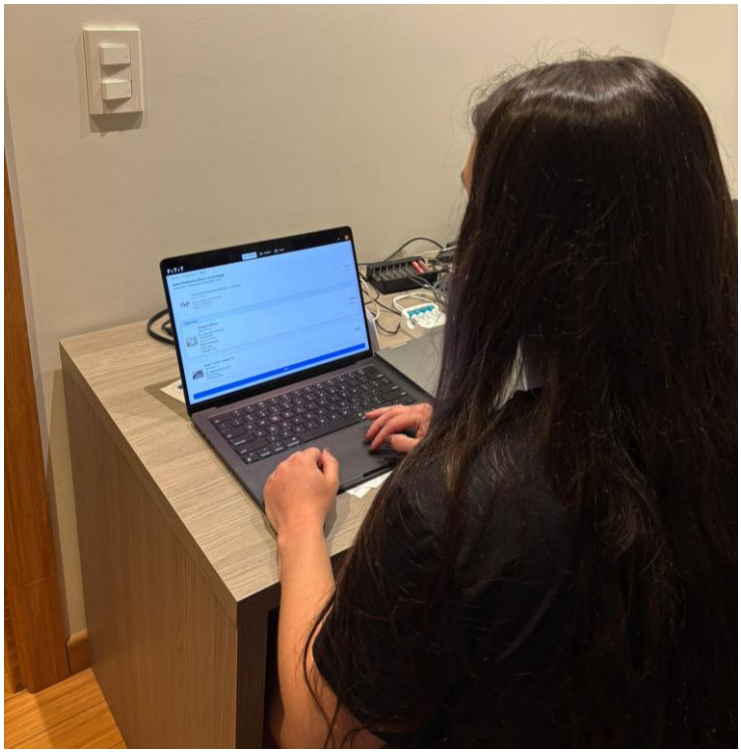


Ilustración 122 - Prueba de usabilidad con usuarios finales

12.20.2. Panel web para tiendas

Para la versión web utilizada por las **tiendas**, se realizaron pruebas con **5 administradores** de comercios asociados. A continuación se detallan los resultados de los principales escenarios evaluados.

Revisar solicitudes pendientes

Desafío: Consultar y filtrar devoluciones pendientes.
Objetivo: Que el usuario pueda listar las solicitudes en **menos de 15 segundos**, sin errores críticos.

Métrica	Resultado esperado	Resultado obtenido
Tiempo	15	11
# de errores	0	0
# de clicks	2	2

Tabla 40 - Resultados de pruebas de usabilidad de “Revisar solicitudes pendientes”

Comentarios generales:

- La lista fue reconocida fácilmente.
- Dos participantes sugirieron agregar un filtro rápido por estado y fecha.

Aprobar una devolución

Desafío: Aprobar una solicitud válida y generar crédito.
Objetivo: Completar el flujo de aprobación en **menos de 30 segundos** con menos de 2 errores.

Métrica	Resultado esperado	Resultado obtenido
Tiempo	30	27
# de errores	1	1
# de clicks	5	3

Tabla 41 - Resultados de pruebas de usabilidad de “Aprobar una devolucion”

Comentarios generales:

- Todos los usuarios comprendieron el proceso de aprobación.
- Se recomendó mejorar el mensaje de confirmación al generar crédito para que sea más visible.

Rechazar una devolución

Desafío: Rechazar una solicitud no válida.
Objetivo: Ejecutar la acción en **menos de 30 segundos** y con menos de una solicitud de asistencia.

Métrica	Resultado esperado	Resultado obtenido
Tiempo	30	24
# de errores	1	0
# de clicks	5	3

Tabla 42 - Resultados de pruebas de usabilidad de “Rechazar devolución”

Comentarios generales:

- El botón de “Rechazar” fue identificado con facilidad.
- Todos los usuarios mencionaron que el texto de confirmación era claro.

12.20.3. Sesiones externas (UserBrain)

Se realizaron pruebas remotas de usabilidad con usuarios reales mediante la plataforma **UserBrain**, en el entorno de *staging*.
El objetivo fue observar el comportamiento natural de los participantes y medir facilidad de uso, tiempos y comentarios espontáneos.

Participante	País	Plataforma	Flujo probado	Tareas finalizadas	Dificultades detectadas	Calificación de facilidad (1–5)	Comentarios
Lucía S.	Uruguay	Desktop (macOS)	Flujo completo de devolución y cambio	7/7	Ninguna bloqueante, detalles visuales menores	**5**	“*Estuvo todo claro. Mejoraría algunos detalles visuales.*”

Henry M.	EE. UU.	Desktop (macOS)	Devolución completa (selección de ítems y motivo)	7/7	Ninguna, flujo claro	**5**	"*Make the reason-for-return tab more visible.*"
Stephanie M.	Canadá	Mobile (iOS 18.6)	Flujo de devolución (inicio, motivo y seguimiento)	6/7	Dificultad menor al buscar estado de devolución	**3**	"*Allow users to see more info about their return*."

Tabla 43 - Sesiones externas de UserBrain

Análisis e interpretación

En las tres sesiones analizadas, los participantes completaron el flujo principal sin bloqueos funcionales.

- **Lucía S.** y **Henry M.** calificaron la facilidad con **5/5**, destacando la simplicidad general del proceso.
- **Stephanie M.**, en entorno móvil, presentó dificultades al consultar el estado de la devolución, lo que sugiere la necesidad de **optimizar la visibilidad del seguimiento en pantallas pequeñas**
- Observaciones menores sobre diseño (espaciado y contraste).
- Se realizaron ajustes visuales en iteraciones posteriores.

Los comentarios coincidieron en dos aspectos de mejora:

1. **Visibilidad del motivo de devolución**, que fue ajustado para resaltar mejor el paso activo.
2. **Información adicional en el seguimiento**, incorporada en versiones posteriores con etiquetas de estado ("pendiente", "aprobada", "rechazada").

El tiempo promedio de finalización fue de **menos de dos minutos** para completar el flujo inicial de devolución, sin requerir asistencia. En conjunto, las sesiones confirmaron que el flujo es **comprensible, directo y estable**.

12.21. Formato de registro de *bugs* en Jira

Este anexo presenta el formato utilizado por el equipo para registrar y dar seguimiento a los defectos detectados durante las pruebas de validación, tanto internas como externas. El registro se realizó íntegramente en Jira, utilizando el tipo de tarea Bug dentro del tablero general del proyecto. Cada incidente era documentado con información estandarizada, lo que permitió asegurar trazabilidad entre la detección, la corrección y la verificación de cada defecto.

Campo	Descripción	Ejemplo
ID	Identificador único asignado automáticamente por Jira.	FR-1
Título	Descripción breve y precisa del defecto.	Devolución no se crea en DB
Tipo de incidencia	Clasificación del defecto (Bug, Task, Improvement).	Bug
Prioridad	Nivel de impacto sobre el sistema o el usuario (Highest, High, Medium, Low, Lowest).	High
Componente afectado	Módulo o funcionalidad donde se detectó el error.	Formulario de devoluciones y *cloud function* de crear devolución.
Descripción detallada	Contexto, pasos para reproducir el error y resultado esperado.	Al presionar el botón de confirmar, no se crea la devolución en la base de datos.
Versión / Sprint	*Release* o iteración en la que se detectó el defecto.	Sprint 1 / Release 1
Responsable asignado	Integrante del equipo encargado de la corrección.	Florencia Batlle
Estado	Etapa del ciclo de vida del *bug*	En Curso
Evidencia	Captura o video adjunto que muestra el error.	Screenshot

Tabla 44 - Formato de registro de bug de Jira

Este formato permitió mantener consistencia en la documentación de defectos, facilitar su análisis en las retrospectivas y medir la efectividad del proceso de corrección a lo largo de los distintos *releases*.

Con el fin de optimizar la planificación de los *sprints* y asegurar una corrección ordenada, los defectos fueron categorizados por prioridad según su impacto en los flujos críticos del sistema. Se mantuvo la escala de Jira, adaptada a la lógica del dominio de Fitit Returns:

1. **Highest (Muy alta):** el defecto impide ejecutar procesos esenciales del sistema, como la creación de una solicitud de devolución, errores en uso de créditos de tienda o errores que bloquean la aplicación.

2. **High (Alta):** el defecto interrumpe el flujo de procesos secundarios, como la actualización del estado de una devolución o la edición de políticas de tienda.

3. **Medium (Media):** el defecto no bloquea el proceso, pero afecta la experiencia de usuario (por ejemplo, errores en el *loading state*, fallos en validaciones visuales o inconsistencias menores en los datos).

4. **Low (Baja):** defectos estéticos o de comportamiento poco frecuente, como errores de espaciado, íconos mal alineados o textos sin traducción.

5. **Lowest (Muy baja):** mejoras visuales o ajustes menores sin incidencia en la funcionalidad, por ejemplo, cambio de color de un botón o microajustes de tipografía.

Esta clasificación permitió priorizar los defectos críticos durante la planificación del *sprint* y diferir los de baja severidad a futuras iteraciones.

12.22. Detalle de *bugs* por iteración

Los *bugs* se enumeran en orden cronológico y el número asignado se utiliza únicamente como identificador dentro de esta tabla. No representan ningún ID oficial fuera del contexto de este documento.

Sprint	***Bugs* solucionados**	**ID de *bugs* reportados**	**Instancia**
SP1	-	-	-
SP2	1. El flujo de devolución no crea la solicitud en BD (prioridad alta).	1	Uso exploratorio del sistema.
SP3	2. Error al guardar la devolución con campos vacíos (prioridad media). 3. El botón “Iniciar devolución” no se visualiza en pantallas chicas (prioridad baja).	2 y 3	Pruebas de usabilidad y análisis visual.
SP4	4. Error de validación en el formulario de devolución (prioridad alta). 5. Texto del campo “motivo” no se guarda correctamente (prioridad media).	4 y 5	Pruebas funcionales.
SP5	6. Al actualizar una devolución, no se muestra mensaje de confirmación (prioridad media). 7. El dashboard de tienda muestra datos duplicados (prioridad alta).	6 y 7	Pruebas funcionales y entorno *staging*.
SP6	8. Inconsistencia en la fecha de creación de las devolución (prioridad media). 9. Botón “Aprobar” se superpone al texto (prioridad baja).	8, 9 y 10	Análisis de usabilidad.

	10. Errores de formato en los mensajes de error (prioridad baja).		
SP7	11. El cálculo del crédito de tienda arroja valores incorrectos (prioridad alta). 12. Los mensajes de confirmación no aparecen en algunos casos (prioridad media).	11 y 12	Pruebas funcionales y revisión de integraciones.
SP8	13. Error al filtrar devoluciones por fecha (prioridad media). 14. Problemas de scroll en *mobile* (prioridad baja).	13 y 14	Pruebas funcionales y exploración del sistema.
SP9	15. Los créditos no se reflejan en el balance del cliente (prioridad alta). 16. Error visual en íconos de estado (prioridad baja). 17. La tabla de devoluciones no se actualiza tras borrar un registro (prioridad media).	15, 16 y 17	Pruebas funcionales y entorno *staging*.
SP10	18. Los totales de devolución se muestran con dos decimales adicionales (prioridad baja). 19. El mensaje de error al ingresar monto de crédito vacío es incorrecto (prioridad media).	18 y 19	Pruebas funcionales.

SP11	20. Error al generar voucher de devolución (prioridad alta). 21. Las notificaciones no se disparan tras aprobar una devolución (prioridad alta). 22. La barra de carga no desaparece luego de enviar formulario (prioridad baja).	20, 21 y 22	Pruebas de usabilidad y exploratorias.
SP12	23. Problema de rendimiento al listar devoluciones con más de 200 registros (prioridad media). 24. Error en la vista del historial de políticas archivadas (prioridad baja).	23 y 24	Pruebas funcionales.
SP13	25. Credits API devuelve respuesta duplicada (prioridad alta). 26. Error al enviar correo de confirmación (prioridad media). 27. No se puede editar una política activa (prioridad alta).	25, 26 y 27	Pruebas funcionales y análisis del *backend*.
SP14	28. Falla intermitente al conectarse con Firebase Storage (prioridad media). 29. El mensaje de éxito se muestra antes de completar la subida de archivo (prioridad baja).	28 y 29	Pruebas funcionales.
SP15	30. Error al aplicar filtros combinados en el dashboard (prioridad alta). 31. Las fechas de devolución no se actualizan en algunos registros (prioridad media).	30 y 31	Pruebas de usabilidad.

SP16	32. Problema de scroll la pantalla de políticas (prioridad baja). 33. Los botones de acción quedan ocultos en pantallas pequeñas (prioridad media).	32 y 33	Análisis de usabilidad.
SP17	34. El sistema permite duplicar devoluciones si se recarga la página rápidamente (prioridad alta).	34	Pruebas funcionales.
SP18	35. Los vouchers presentan errores en formato (prioridad media).	35	Pruebas de usabilidad y entorno *staging*.

Tabla 45 - Detalle de *bugs* por iteración

12.23. Aplicación de Heurísticas de Nielsen

Heurística	Ejemplo en la solución
1. Visibilidad del estado del sistema	*Spinner* en cargas, estado de solicitud visible (pendiente/aprobada/en entrega/rechazada), mensajes de lista vacía cuando no hay devoluciones.
2. Correspondencia con el mundo real	Terminología cercana al usuario: “N.º de orden” en vez de *ID* de orden, motivos de devolución con lenguaje natural
3. Control y libertad del usuario	Botón atrás persistente, confirmación no bloqueante.
4. Consistencia y estándares	Estilos unificados en botones y etiquetas, patrones de interacción repetibles (mismo patrón de modal para aprobar/rechazar) en todas las pantallas.
5. Prevención de errores	Validación previa de formato/tamaño de imagen, deshabilitar Enviar hasta completar datos mínimos.
6. Reconocimiento antes que recuerdo	Lista de devoluciones con datos clave visibles (tienda, fecha, estado, número de orden) y acciones directas por fila, filtros con chips seleccionados a la vista.
7. Flexibilidad y eficiencia de uso	Filtros rápidos (estado/fecha/cliente) en panel de tienda, autocompletar de campos frecuentes, *drag-and-drop* de imágenes
8. Estética y diseño minimalista	Reducción de texto redundante, mejor contraste y jerarquías, foco en la tarea (CTA principal destacado, secundarios discretos).
9. Ayudar a reconocer y recuperarse de errores	Mensajes específicos y accionables (“La imagen supera 5 MB. Subí un archivo menor o comprimirlo.”) con enlaces de solución.
10. Ayuda y documentación	FAQ breve sobre devoluciones/cambios, tooltip que muestra dónde encontrar el N.º de orden

Tabla 46 - Aplicación de Heurísticas de Nielsen

12.24. Configuración de package.json

12.24.1. Extracto de package.json

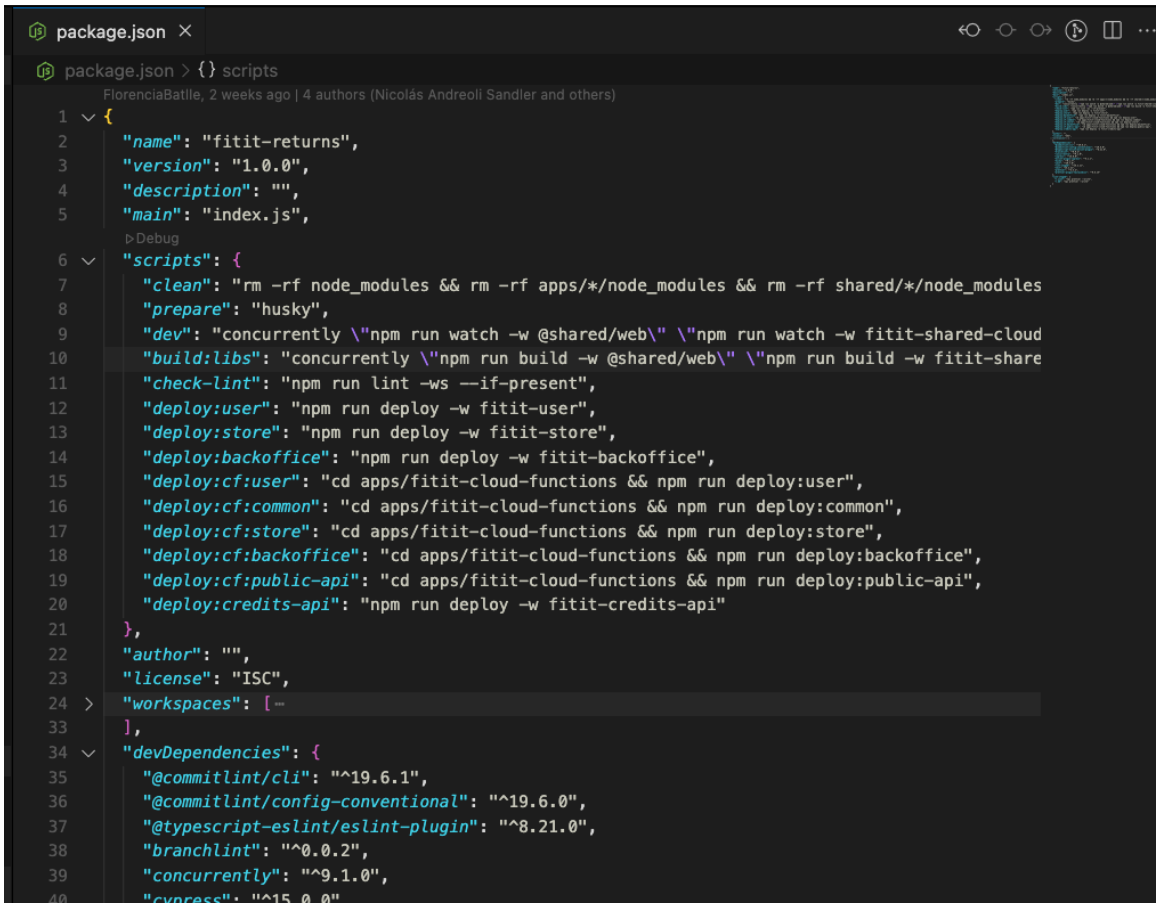


Ilustración 123 - Package.json global

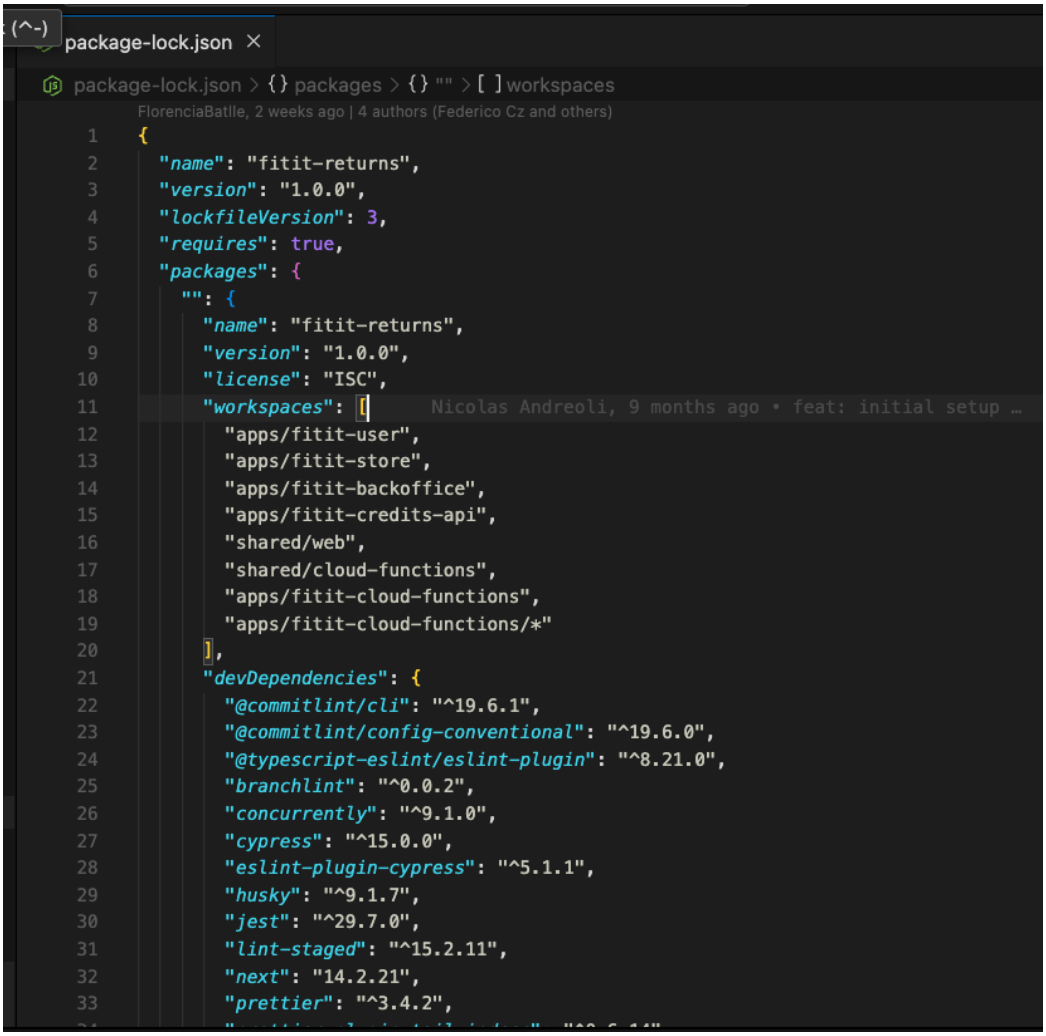


Ilustración 124 - Package-lock.json global

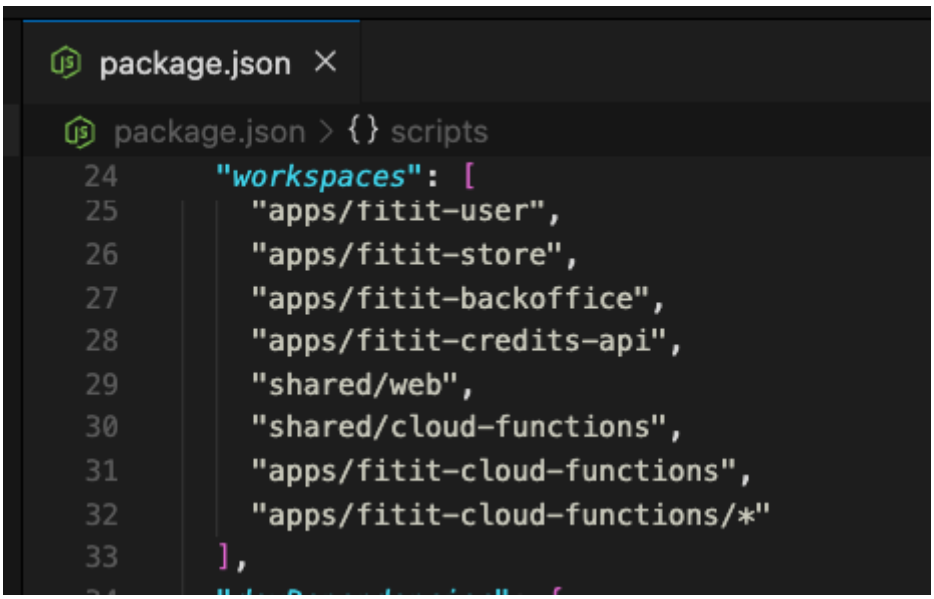


Ilustración 125 - workspaces

12.24.2. Github actions

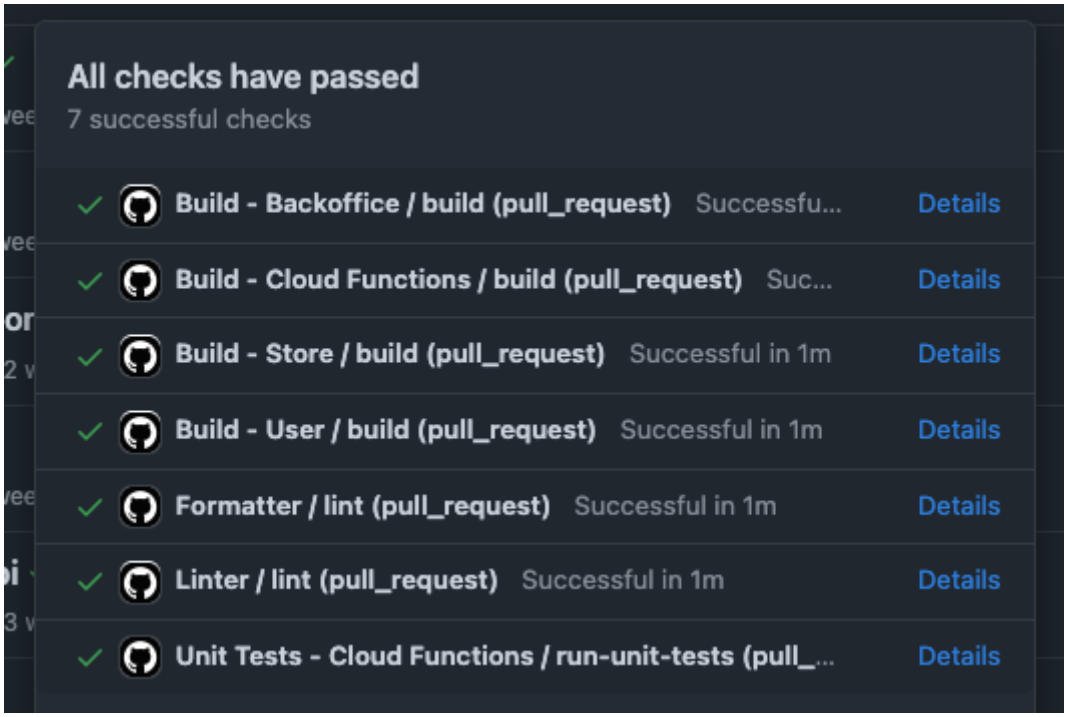


Ilustración 126 - Corrida de todo el pipeline de Github actions

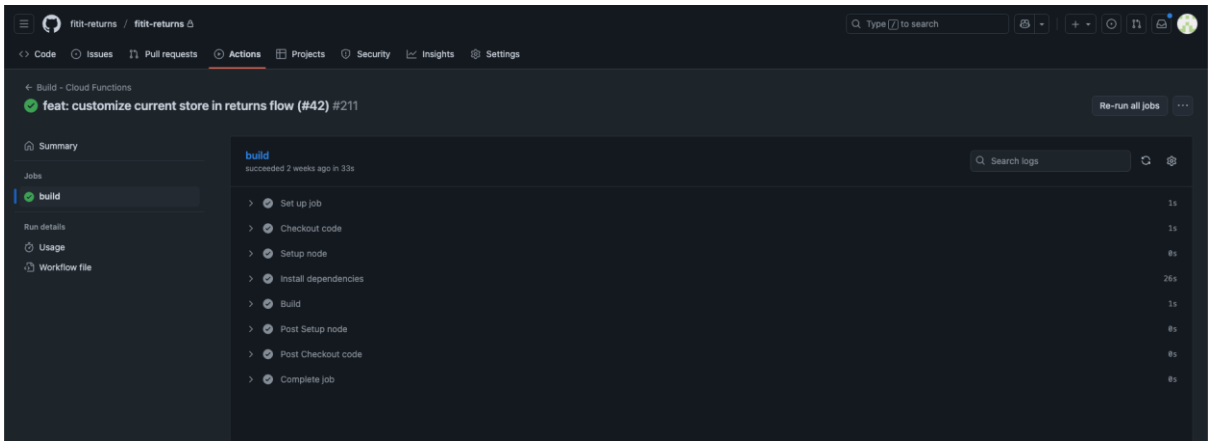


Ilustración 127 - Pasos de la Github action de build de *Cloud Functions*

12.25. Formulario inicial de devolución manual en fitit-store

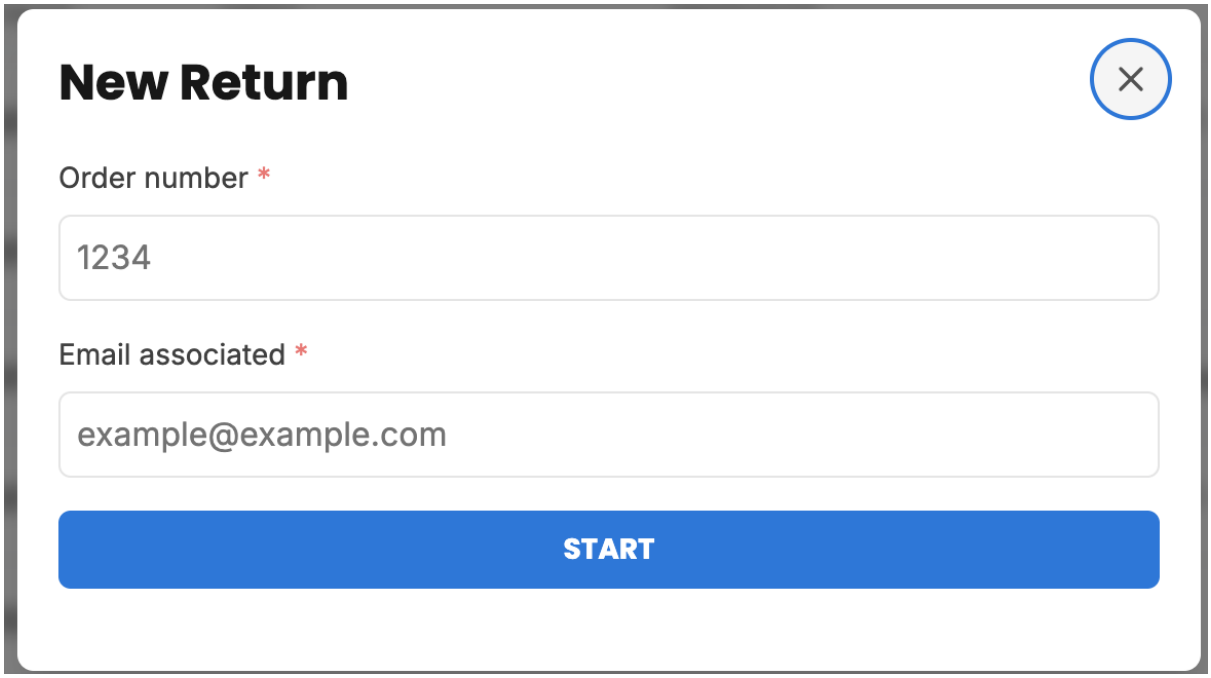


Ilustración 128 - Formulario inicial de devolución manual en fitit-store

12.26. Evidencia de pruebas en Cypress

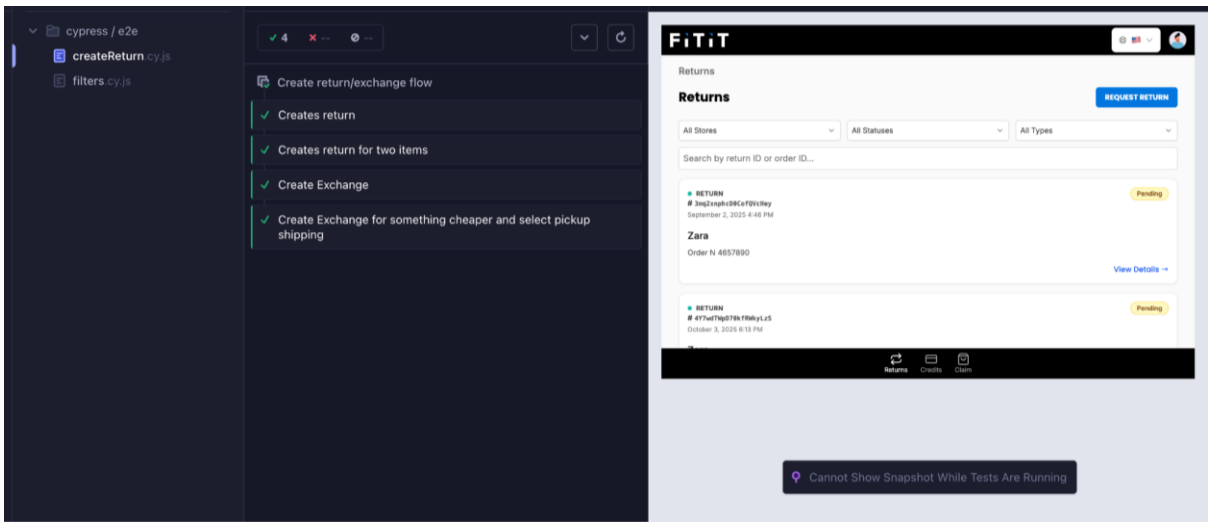


Ilustración 129 - Pruebas del flujo de usuario final en Cypress

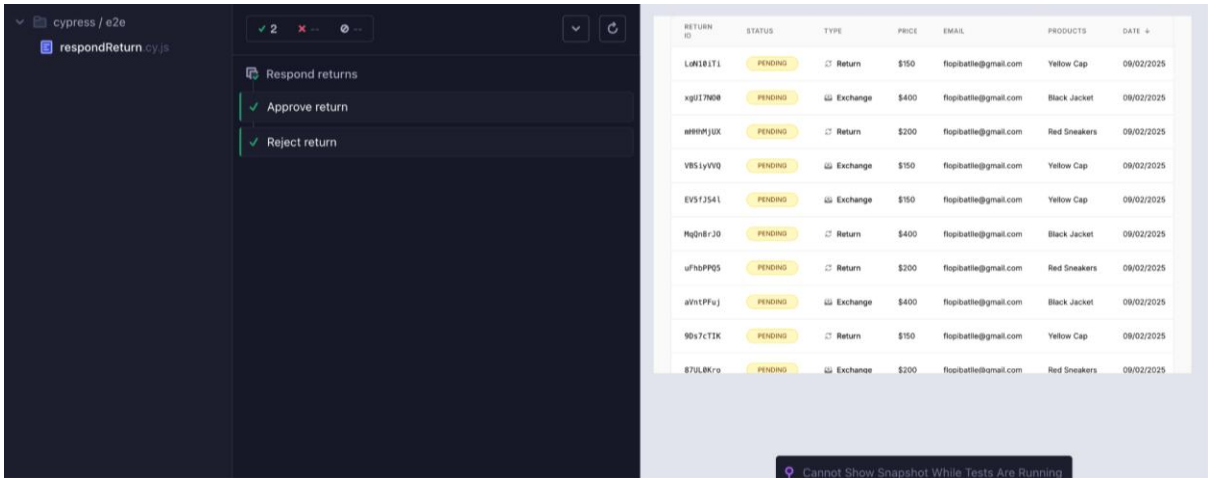


Ilustración 130 - Pruebas del flujo de tienda en Cypress

12.27. Escalidraw

12.27.1. Diagrama de paquetes de *backend* en Escalidraw

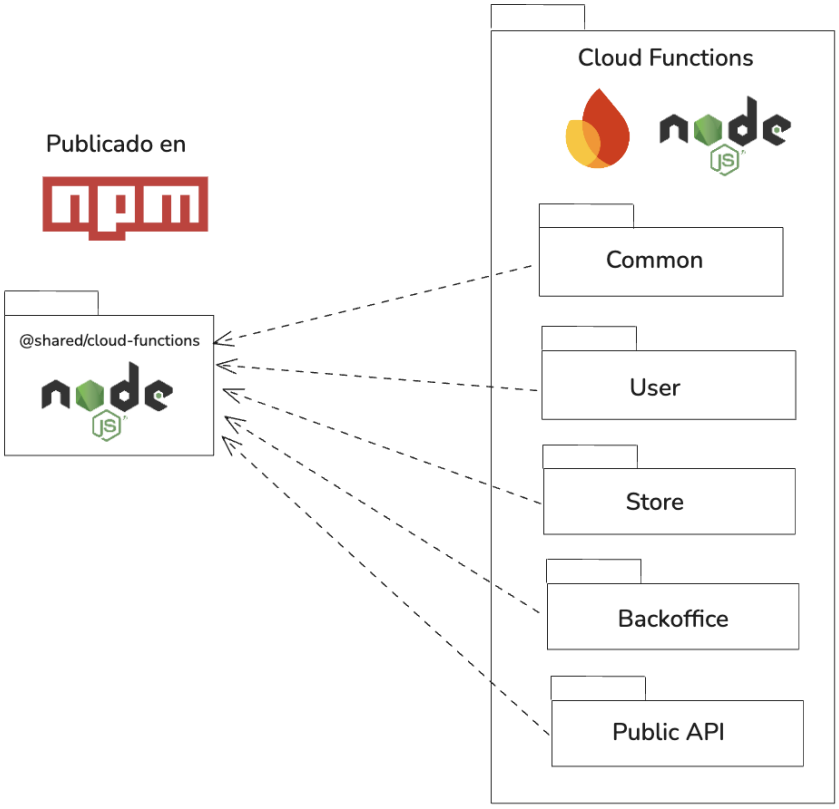


Ilustración 131 - Diagrama paquetes *backend* en Escalidraw

12.27.2. Diagrama de paquetes de *frontend* en Escalidraw

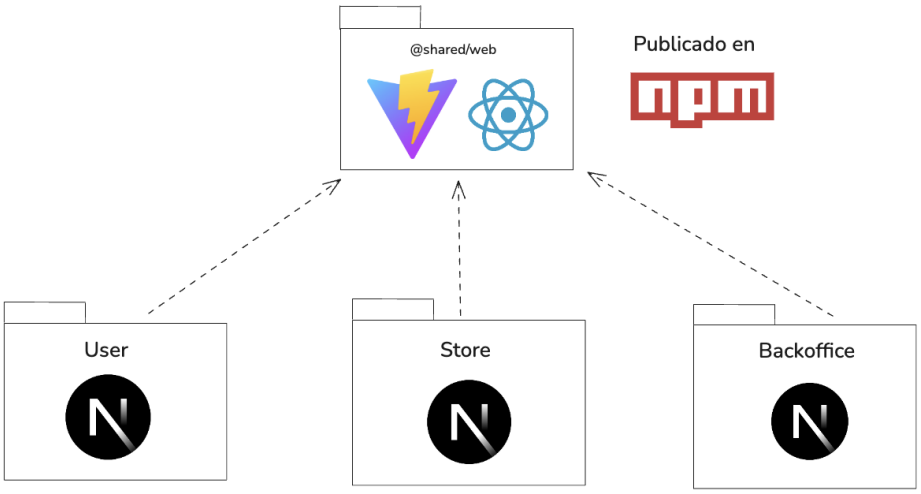


Ilustración 132 - Diagrama paquetes *frontend* en Escalidraw

12.28. Diseño y estilos de Fitit

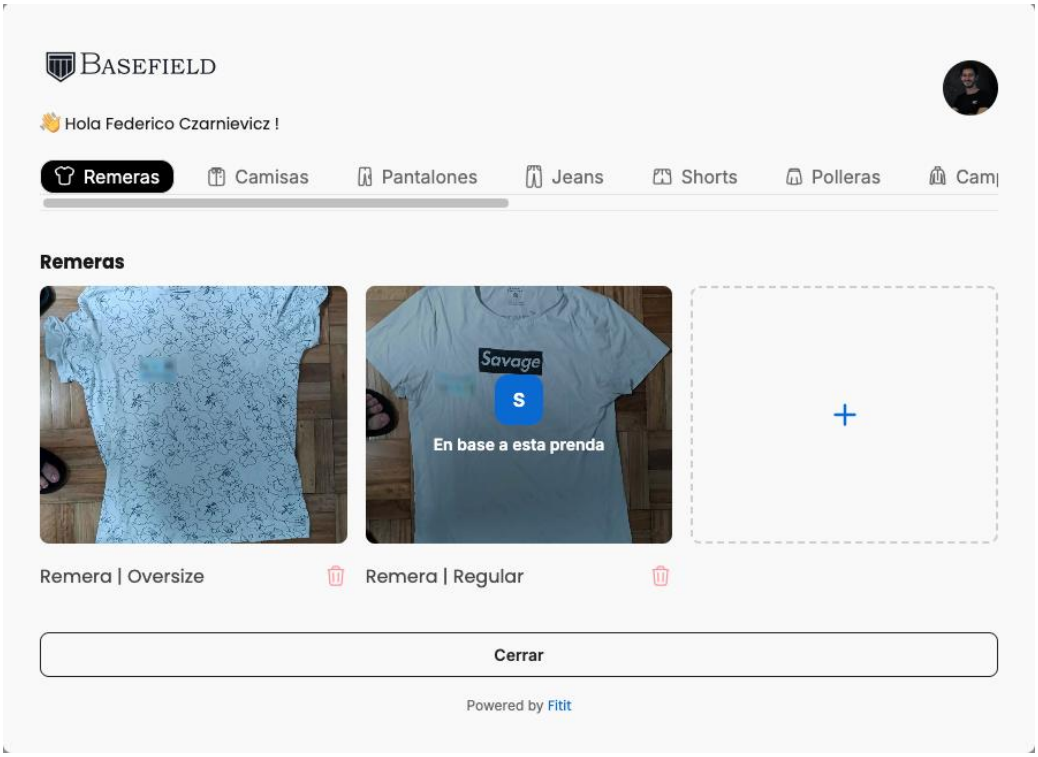


Ilustración 133 - Recomendador de talles

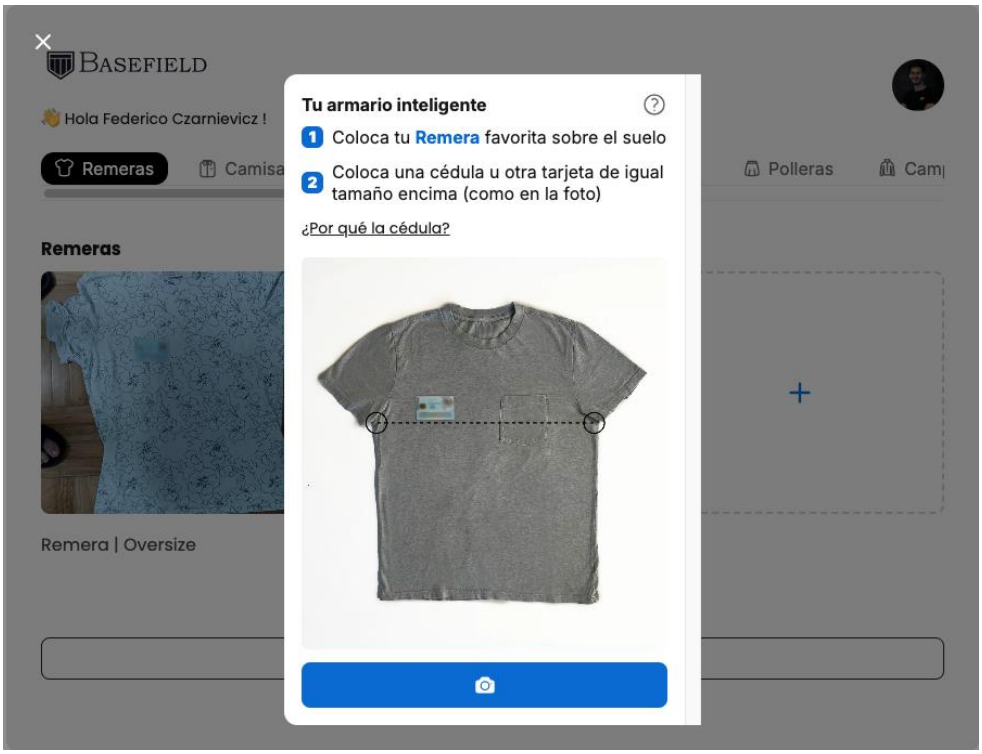


Ilustración 134 - Recomendador de talles pt.2

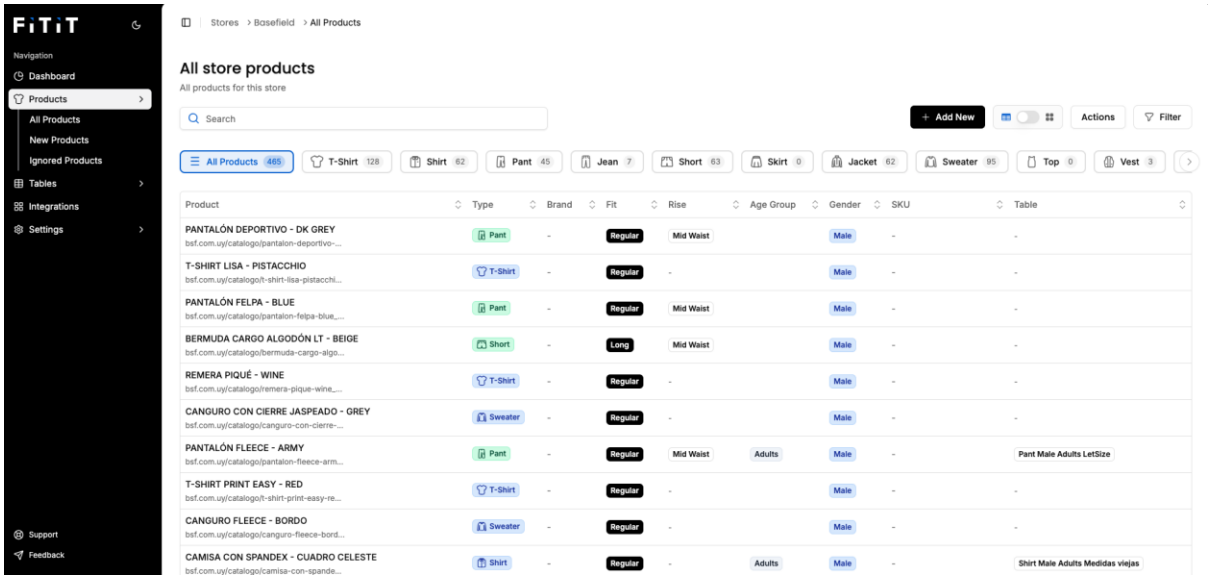


Ilustración 135 - Pantalla de *backoffice* de Fitit

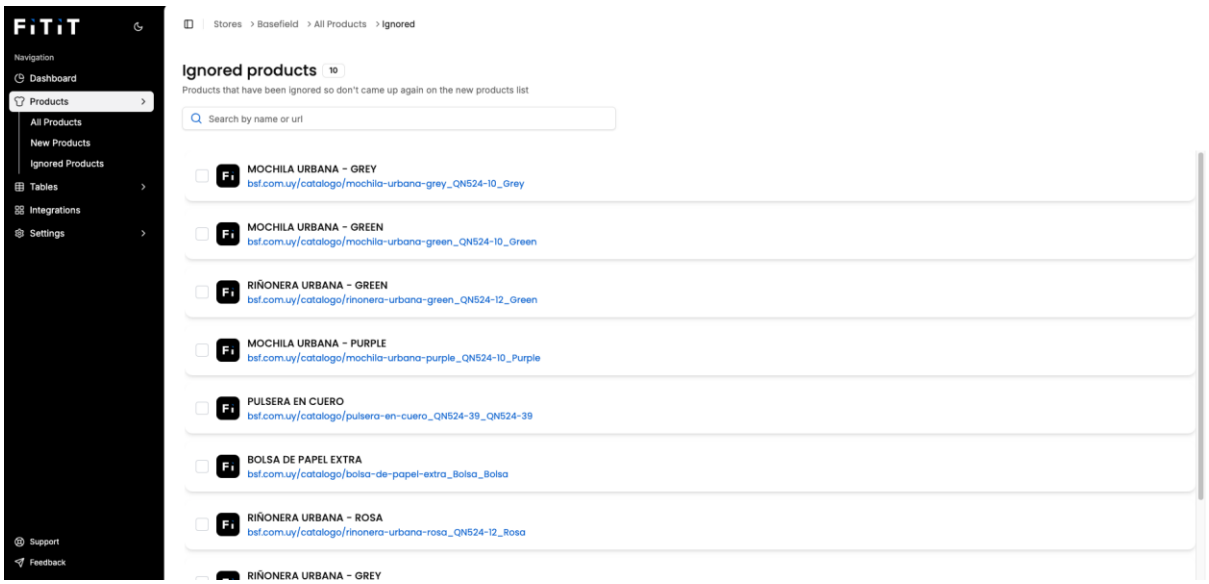


Ilustración 136 - Pantalla de *backoffice* de Fitit pt.2

12.29. Reunión en oficina de Fitit

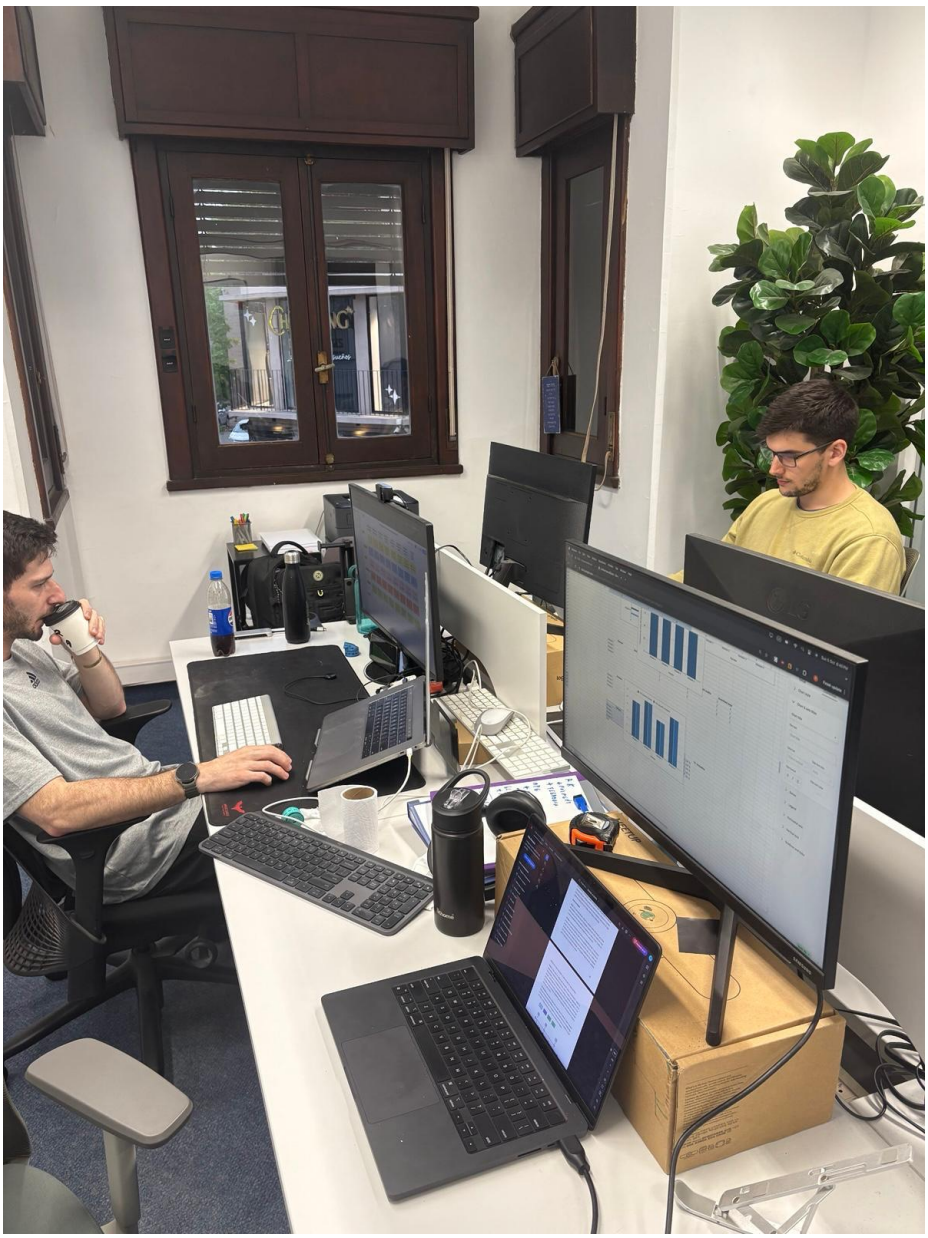


Ilustración 137 - Reunión de equipo en oficina de Fitit