Universidad ORT Uruguay
Facultad de Ingeniería

Geolocalizador: Plataforma para monitoreo de repartos

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

Rodrigo Arsuaga – 193006
Leticia Esperón – 190614

Tutor: Pablo Hernández Guimarans

2019

Declaración de autoría

Nosotros, Rodrigo Arsuaga y Leticia Esperón, declaramos que el trabajo que se presenta en esa obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos el proyecto de grado de la carrera de Ingeniería en Sistemas;
- 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.

Rodrigo Arsuaga

3 de marzo de 2019

Leticia Esperón

3 de marzo de 2019

Agradecimientos

En primer lugar, queremos agradecerle a nuestro tutor, Pablo Hernández, por su orientación a lo largo del proyecto y apoyo para la conclusión del mismo.

Asimismo, nos gustaría expresar nuestro agradecimiento a quienes trabajan en el laboratorio de ORT Software Factory, en especial a Gastón Mousqués, Rafael Bentancour y Álvaro Ortas por sus valiosos aportes y perspectivas durante reuniones o revisiones.

También quisiéramos agradecer a Martín García y Nicolás Benítez, representantes de la empresa Formisur, quienes supieron ser clientes muy dispuestos y responsables. Se valora su compromiso y su energía, y esperamos haberles contribuido.

A Belén Iglesias, quien hizo un excelente trabajo con los diseños del sistema. Al equipo Android de XmartLabs, que nos ayudaron y aconsejaron en la tecnología cuando nos encontramos estancados. Estamos extremadamente agradecidos de haber contado con su ayuda incondicional.

A nuestros compañeros de clase, con quienes tuvimos el placer de transitar juntos cinco años de carrera y que hoy se han convertido en personas muy importantes para nosotros. No nos imaginamos haber llegado hasta acá sin ustedes.

Nuestro más grande agradecimiento a nuestra familia y amigos, quienes nos apoyaron y contuvieron a lo largo del año. Somos afortunados de contar con ustedes y de hoy poder compartir la alegría de haber culminado esta etapa.

Finalmente, agradecernos entre nosotros. Por haber trabajado como equipo una vez más, habiendo superado cualquier obstáculo que se nos enfrentó. Por saber encontrar la motivación dentro de nosotros mismos y por estar más unidos que nunca.

A todos, muchas gracias por contribuir a que hoy estemos aquí.

Abstract

El proyecto surgió a raíz del problema presentado por representantes de la empresa de logística Formisur en el laboratorio Software Factory de la Universidad ORT Uruguay, donde se buscó construir un sistema para permitir el ingreso y seguimiento de los repartos asignados a cada repartidor.

El sistema construido está compuesto por una aplicación móvil para el repartidor y una aplicación web para el administrador.

La aplicación web permite programar un reparto a partir de un archivo de pedidos que se genera con un sistema externo. Las direcciones de los destinatarios finales, que son ingresadas en cualquier formato, son auto geolocalizadas o corregidas y memorizadas para futuros repartos. El sistema provee la ruta más rápida para entregar los hasta ciento treinta pedidos que componen un reparto.

La aplicación móvil permite a un repartidor comenzar un reparto que se le haya asignado y registrar la entrega o rebote de cada pedido. El administrador puede monitorear en tiempo real el reparto y visualizar alertas sobre posibles ineficiencias de parte de los repartidores.

Durante el proceso de desarrollo se procuró cumplir con altos estándares de mantenibilidad del código requeridos por el cliente, además de con otros atributos de calidad priorizados por el equipo como disponibilidad, eficiencia en el uso de datos móviles y usabilidad en la aplicación Android dado el contexto de uso en la ruta.

El proyecto involucró un mes de investigación y pruebas con el fin de solucionar el enrutamiento de pedidos, y sufrió un cambio drástico de implementación de la solución al surgir una oportunidad de negocio con una empresa emergente casi al finalizar la etapa de desarrollo.

Durante la etapa de desarrollo el sistema fue probado en repartos reales. La primer instancia de pruebas se llevó a cabo al finalizar el MVP, con el objetivo principal de recolectar retroalimentación que evolucionó el *Backlog* hacia el sistema final, y la segunda instancia se realizó al final de la etapa de desarrollo.

Hoy la empresa se encuentra exitosamente utilizando el sistema para los repartos del día a día y se evalúa el adaptar el producto construido para ser ofrecido como servicio a otras empresas del rubro bajo el modelo de distribución SaaS.

Palabras clave

Android; Aplicación del administrador; Aplicación del repartidor; Geolocalización; Cliente; Desarrollo ágil; Destinatario; Enrutamiento; Formisur; Google Play Store; Logística; Monitoreo; Pedido; Prototipación; Recorrido; Repartidor; Reparto; Ruby on Rails; Ruta; Scrum; Seguimiento;

Índice

Índice de tablas

Índice de ilustraciones

1. Introducción

Este capítulo pretende introducir al proyecto Geolocalizador, llevado a cabo por los estudiantes Rodrigo Arsuaga y Leticia Esperón bajo la tutoría de Pablo Hernández, como requisito para la obtención del título de Ingeniero en Sistemas de la Universidad ORT Uruguay en marzo de 2019.

En este capítulo se explica el origen del proyecto, los objetivos del equipo y la estructura del documento.

1.1. Origen del proyecto

El proyecto surge como propuesta de la empresa de logística Formisur, presentada en el laboratorio Software Factory de la Universidad ORT. El proyecto se presentó bajo el nombre “Geolocalizador de repartos” y consistía de una aplicación para dispositivos móviles, particularmente Android, y una web para el administrador, de uso interno y exclusivo para la empresa, cuya funcionalidad principal sería permitir el ingreso y seguimiento de los pedidos de un reparto asignados a un repartidor.

El equipo se postuló al proyecto y fue posteriormente asignado al mismo por el comité.

Uno de los principales factores que motivó al equipo a postularse al proyecto fue el interés e impacto que el proyecto tendría en la empresa, tal como lo transmitieron sus representantes. Esto significaría que los clientes estarían sumamente comprometidos con el proyecto lo cual el equipo consideró clave para el éxito del mismo. Por otro lado, los clientes no contaban con una solución definida por lo que los integrantes tendrían mucha influencia en las decisiones en cuanto al diseño de la solución. Finalmente, el proyecto aparentaba presentar complejidades técnicas desafiantes.

1.2. Breve introducción al cliente

Formisur S.A. es una empresa uruguaya en expansión dedicada a la logística empresarial. Dentro de los servicios que provee se encuentran la recepción, almacenaje, control y distribución de mercadería a todo el país, siendo el último el servicio en el cual el equipo se concentró en mejorar.

1.3. Objetivos

Una de las primeras actividades realizadas en el proyecto fue la definición de objetivos. El equipo reflexionó acerca de cuáles serían las metas que buscarían alcanzar con la realización del proyecto. El formalizar los objetivos permitió realizar seguimiento de los mismos durante todo el proyecto para asegurar así su cumplimiento.

Se dividieron los objetivos del proyecto según distintas categorías. A continuación se presentan los objetivos:

1.3.1. Objetivos académicos

Aprobación del proyecto final con al menos 90
El equipo apuntó a finalizar el proyecto y obtener una calificación de excelencia para la obtención del título de Ingenieros en Sistemas.

1.3.2. Objetivos del producto

Lograr un producto que sea puesto en producción antes de la entrega final
Uno de los objetivos más desafiantes que se planteó el equipo fue construir un producto que el cliente decida poner en producción antes de la entrega final.

Lograr un producto con cero defectos críticos reportados al momento de la entrega final
El equipo consideró que un producto de calidad no debe tener defectos críticos, por lo que se puso como objetivo entregar un producto final libre de ellos.

Conseguir un promedio mayor a 4 y ningún resultado menor a 3 en la encuesta de satisfacción a los interesados al terminar el proyecto (donde 5 representa el puntaje más positivo y 1 el peor).
El equipo consideró que la satisfacción del usuario final es uno de los mayores indicativos del éxito de un producto a construir. Se definieron encuestas de satisfacción para repartidores, administradores y clientes que evalúan el producto final y en el caso del cliente también otros aspectos del trabajo del equipo.

1.3.3. Objetivos del equipo

Lograr que al finalizar el proyecto ambos integrantes respondan afirmativamente a la pregunta ¿Volverían a trabajar en un proyecto juntos?
Se consideró fundamental mantener la relación durante el proyecto, formulando este objetivo como forma de evaluarlo. El medio para alcanzar este objetivo fue definir y cumplir con las prácticas especificadas en el reglamento interno en cuanto a políticas internas y comunicación.

1.3.4. Objetivos del proceso

Trabajar entre 10 y 30 horas semanales por persona a lo largo de todo el proyecto.
El propósito de este objetivo es balancear el trabajo a lo largo de la duración del proyecto, aumentando las posibilidades de alcanzar las expectativas en los

entregables sin recurrir a trabajar más de lo debido en períodos concretos por acumulación de tareas pendientes.

Obtener un porcentaje de retrabajo menor al 5% de las horas totales
El tiempo limitado y tamaño del equipo requerirían una buena planificación e investigación previa para mantener el retrabajo al mínimo con el fin de cumplir con el alcance del producto. Esto se plasmó directamente en uno de los objetivos con el fin de tenerlo presente continuamente.

Lograr que el tiempo dedicado a tareas de gestión o de coordinación como reuniones y actas sea menor al 10% de las horas totales para el final del proyecto
Se definió que un buen proceso debe organizar al equipo sin consumir demasiado tiempo. Se consideró que en un equipo de dos personas que trabajan presencialmente la mayor parte del tiempo, invertir más de un 10% del tiempo en estas tareas comienza a ser improductivo.

1.4. Estructura del documento

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

Introducción
En este capítulo se introduce el proyecto y los objetivos del equipo.

Problema y Solución
En este capítulo se introduce el proceso que seguía la empresa previa a la implantación del nuevo producto, los problemas que se desprendían del mismo y la solución final construida por el equipo.

Marco Metodológico
En este capítulo se detallan las características del equipo y del proyecto para explicar cómo impactaron en las principales decisiones acerca de la metodología aplicada en el proyecto.

Ingeniería de Requerimientos
En este capítulo se explica el proceso de relevamiento, especificación, validación y estimación de los requerimientos del sistema.

Arquitectura y Desarrollo
En este capítulo se describen los principales desafíos de la arquitectura del software y se explica cómo se cumplió con los atributos de calidad a través de la misma.

Gestión de Calidad
En este capítulo se definen los objetivos de calidad para el producto y el proceso y las actividades realizadas para garantizarlas. Se describen la gestión de incidentes, de

cambios y de configuración, así como las métricas de calidad obtenidas a lo largo del proyecto y sus conclusiones.

Gestión de Proyecto
En este capítulo se describe detalladamente la metodología seguida con las adaptaciones concretas realizadas. También se explica cómo se utilizaron algunas herramientas claves en la gestión y se muestran los resultados de las métricas definidas explicando cómo influenciaron en las decisiones durante el proyecto.

Conclusiones
En este capítulo se contrasta la culminación del proyecto con los objetivos iniciales, desprendiendo conclusiones, lecciones aprendidas y próximos pasos. Al final se incluye una copia de la carta de satisfacción redactada y firmada por el cliente.

2. Problema y solución

Este capítulo explica el contexto, el problema del cliente que lo llevó a acercarse a la ORT, se muestra la solución planteada por el equipo y se analizan los interesados.

2.1. Contexto del problema

Los clientes de Formisur son grandes empresas cuyos clientes representan los destinatarios finales de los pedidos a entregar. Las empresas clientes tienen en cuenta el sistema que utiliza Formisur, ya que son conscientes de que cualquiera sea el sistema que se utilice, tiene un impacto directo en la eficiencia de la entrega de su mercadería.

Al inicio del proyecto Formisur pretendía expandirse y se encontraba en búsqueda de firmar un contrato con una empresa cliente muy importante por lo que mejorar la profesionalidad de la gestión a los ojos de esta se convirtió en prioridad y objetivo del producto a construir.

Siendo esto así, el equipo se concentró en descubrir y entender las expectativas de las empresas cliente de una empresa de logística.

Para ello se realizó una entrevista a un representante de una empresa cliente de Formisur.

Los siguientes fueron algunas de las necesidades resultantes de la misma:

- **Que la mercadería sea entregada en el día previsto.**
  A veces no todos los pedidos de un reparto pueden ser completados en el mismo día por lo que hay pedidos que se postergan para el día siguiente. Esto puede ser un inconveniente para el destinatario final.

- **Que la mercadería sea entregada durante el día.**
  Una vez que se hizo la noche, realizar una entrega se considera invasivo para el destinatario final por temas de seguridad.

- **Que no desaparezca mercadería.**
  Si un pedido se marcó como entregado pero igual es reclamado, que exista evidencia de ello.

- **Que se pueda conocer el estado de un pedido de un reparto en curso.**
  Muchas veces los destinatarios llaman preguntando por su pedido. Es importante poder contestarles con el estado del mismo, por ejemplo “el repartidor llegó a la casa pero no había nadie” o “el repartidor todavía tiene tres pedidos por entregar antes que el suyo. Llegará en aproximadamente 10 minutos.”

Teniendo los anteriores puntos en cuenta, se buscó descubrir discordancias entre lo que esperan las empresas cliente versus lo que se ofrece para identificar oportunidades de mejora a través del software a construir.

2.2. Proceso antes del proyecto

Con el objetivo de identificar oportunidades de mejora, primero fue necesario entender las características de los repartos y el proceso que seguían los mismos en Formisur, previo al uso del software construido por el equipo.

El siguiente proceso fue descrito por Formisur:

1. La empresa cliente envía un archivo de texto con todos los datos de una campaña (lista de pedidos a entregar en los próximos 30 días) ya separados por zonas.

2. El administrador corrige las direcciones del archivo de texto para arreglar el formato o que sean entendidas por los repartidores, ya que son direcciones que muchas veces ingresa directamente el destinatario final sin previa validación.

3. El archivo de texto se carga en un sistema web propio de Formisur que lo procesa y entre otras cosas:

    - Registra datos de cada pedido como cantidad de bultos para a continuación poder distribuir entre camionetas y días.

    - Arma los repartos, separando los pedidos por día y por repartidor.

    - Por cada reparto emite un archivo Excel con los pedidos de ese reparto.

    - Se asigna un código y boleta a cada pedido.

4. Las boletas de los pedidos se imprimen.

5. Las boletas se entregan al repartidor correspondiente y se escanean en otro sistema mientras se hace esta asignación.

6. Cada repartidor analiza y ordena las boletas de los pedidos según la ruta que vaya a realizar. Suelen tomar una ruta similar diariamente.

7. Cada repartidor se encarga de entregar los pedidos de sus boletas, escribiendo en la misma boleta si fue entregada o no, la firma del destinatario o el motivo de rebote.

8. Cuando llega un repartidor a la base luego de completar un reparto, se separan las boletas físicas, entre entregadas y rebotadas, y a su vez las rebotadas por el motivo, y se escanean en el sistema. Esta parte toma aproximadamente treinta minutos, considerando que pueden llegar a ser ciento treinta boletas por reparto.

Algunos datos obtenidos durante entrevistas que pueden parecer menores pero en realidad fueron clave para ciertas decisiones son:

- Suelen ir dos repartidores por reparto. Uno de ellos conduce y el otro organiza los pedidos, entrega, etc.
- Cada reparto toma una jornada entera y consiste de entre ochenta y ciento treinta pedidos.
- Todos los pedidos suelen ser en una misma zona.
- Cada reparto entrega pedidos de una empresa cliente.
- Los repartos de un mismo cliente suelen contener siempre los mismos pedidos, difiriendo en hasta diez.
- La empresa armó a mano un camino base para cada reparto. Los repartidores conocen mejor el camino y quizás no utilicen el camino base.
- Los vehículos suelen ser camionetas.
- Los celulares que se le entregan a los repartidores para realizar llamadas tienen sistema operativo Android.
- Los repartidores tienen un plan de datos de 4 GB por mes.
- Las camionetas no tienen rastreador.
- Es un hecho que algunos repartidores realizan paradas personales, pierden tiempo en redes sociales entre pedidos y marcan pedidos rebotados aunque nunca fueron al lugar.

2.3. Problemas encontrados

Juntando la información obtenida sobre las necesidades de las empresas cliente, la motivación de Formisur de satisfacer estas necesidades, y las características del proceso actual, se identificaron los siguientes problemas a resolver:

1. En el software que utilizaban las direcciones de los destinatarios deben ser corregidas manualmente para que puedan ser entendidas rápidamente por los repartidores. Ya que la mayor parte de los pedidos se repiten diariamente, se terminan corrigiendo las mismas direcciones todos los días, consumiendo mucho tiempo al administrador.
2. El telefonista de Formisur no sabe el estado de un pedido hasta que el repartidor vuelve a la base con las boletas marcadas, impidiendo que responda reclamos de un reparto en curso.

3. Reciben quejas por pedidos que fueron rebotados aunque el destinatario estaba en su casa y no tienen forma de argumentar o verificar si el repartidor efectivamente fue o no.

4. Las boletas se suelen perder, y junto a ellas la firma del destinatario que prueba que el pedido fue aceptado.

5. Los repartidores pierden tiempo ordenando, manipulando o buscando entre boletas durante el reparto. El tiempo extra que le insume puede implicar no terminar con los pedidos a tiempo y tener que postergar alguno para el día siguiente.

6. Se toma hasta media hora por reparto al finalizar cada jornada en escanear las boletas para registrar el estado de cada pedido. Esto es tiempo que se podría utilizar en otra tarea o reducir la carga horaria de los funcionarios.

7. La ruta sugerida a los repartidores se arma a mano, siendo difícil de escalar si mañana se tienen más repartos por día o los pedidos no son tan repetitivos.

8. La ruta predefinida o que toma el repartidor puede no ser la más rápida.

9. No pueden visualizar la mayoría de las ineficiencias de los cadetes. El repartidor puede tomarse paradas personales o ser ineficiente y no hay forma de saberlo. Nuevamente, el tiempo extra que le insume puede implicar no terminar con los pedidos a tiempo y tener que postergar alguno para el día siguiente.

2.4. Solución propuesta

A partir de los problemas mencionados anteriormente, se construyó una solución final que consiste de una aplicación para dispositivos móviles, particularmente *Android*, y una web para el administrador.

En el siguiente listado se enumeran las características de la solución final haciendo coincidentes los puntos problema-solución.

1. Registro de direcciones de destinatarios geolocalizadas para no tener que corregir las mismas direcciones todos los días.

2. Conocer el estado de un pedido apenas fue completado, antes de que el repartidor vuelva a la base.

3. Permitir ver la razón de un pedido rebotado y probar que el repartidor efectivamente estuvo ahí.

4. Permitir probar que un pedido fue entregado.

5. Eliminar la necesidad de boletas impresas durante el reparto, disminuyendo el tiempo que se pierde ordenándolas y manipulándolas.

6. Sincronizar automáticamente con el otro sistema el estado de los pedidos eliminando la necesidad de escanear boletas al finalizar cada reparto.

7. Armado automático de la ruta más corta.

8. Monitoreo de los repartos:

    - Seguimiento de los pedidos de un reparto
    - Visualización del recorrido del repartidor
    - Alertas ante detección de posible falta por parte del repartidor

A continuación se muestran algunas capturas de pantalla de las funcionalidades principales del sistema.

En el menú “Repartos” el administrador puede visualizar la lista de repartos, filtrarla y ordenarla.

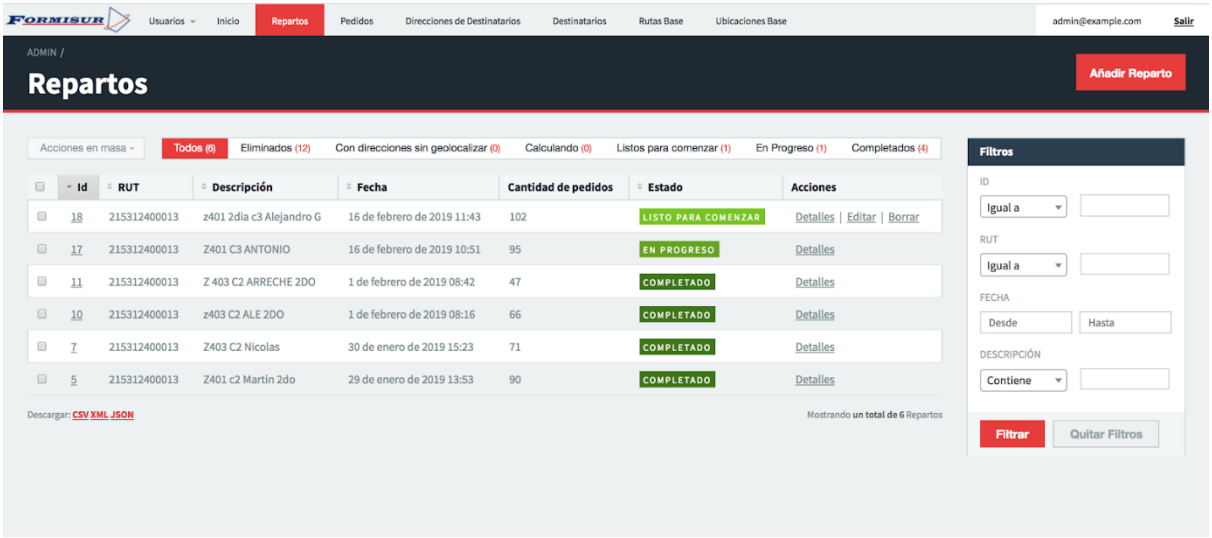


Ilustración 1 - Listado de repartos

El administrador tiene la posibilidad de programar un nuevo reparto a partir un archivo Excel de pedidos. Puede asignar repartidores, especificar de dónde llegan y vuelven los repartidores y algunas otras configuraciones.

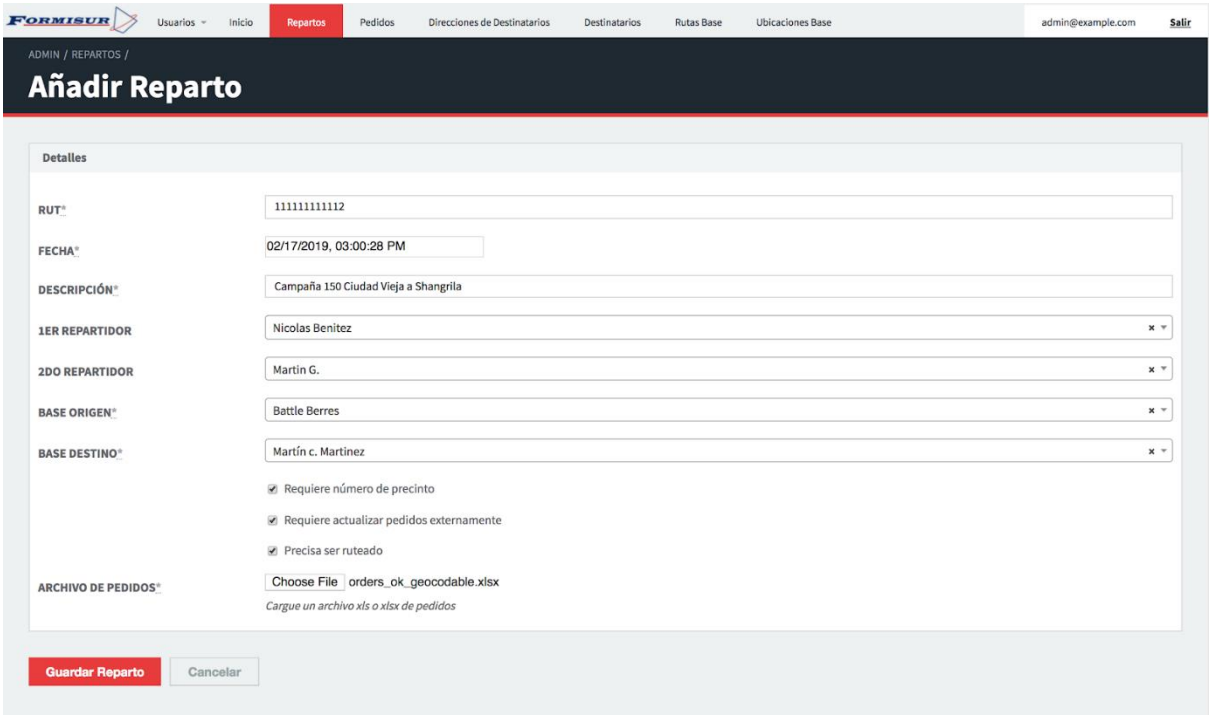


Ilustración 2 - Creación de un reparto

Los detalles de un reparto se visualizan organizados en secciones.

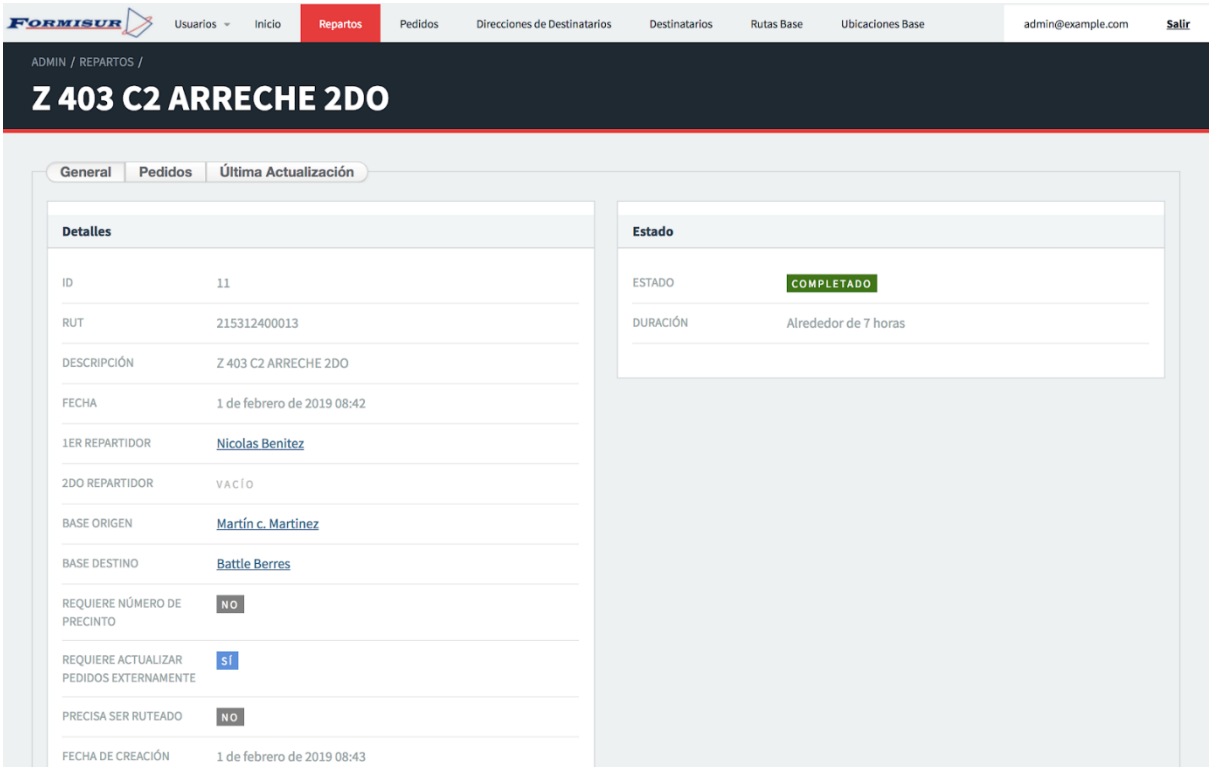


Ilustración 3 - Visualización de detalles de un reparto

La pestaña “Ruta Planificada” se encuentra disponible en repartos que han seleccionado la opción de enrutar los pedidos.

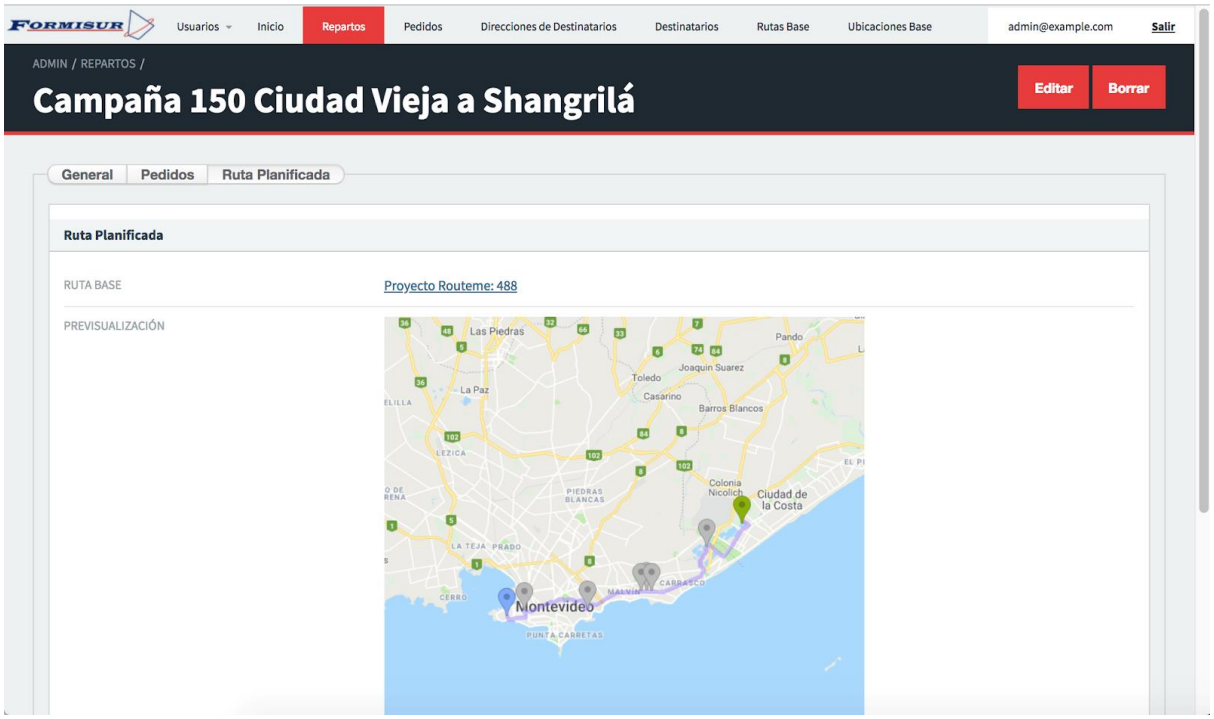


Ilustración 4 - Visualización de la ruta planificada de un reparto

En la pestaña “Pedidos” puede ver el resumen y listado de los pedidos del reparto.

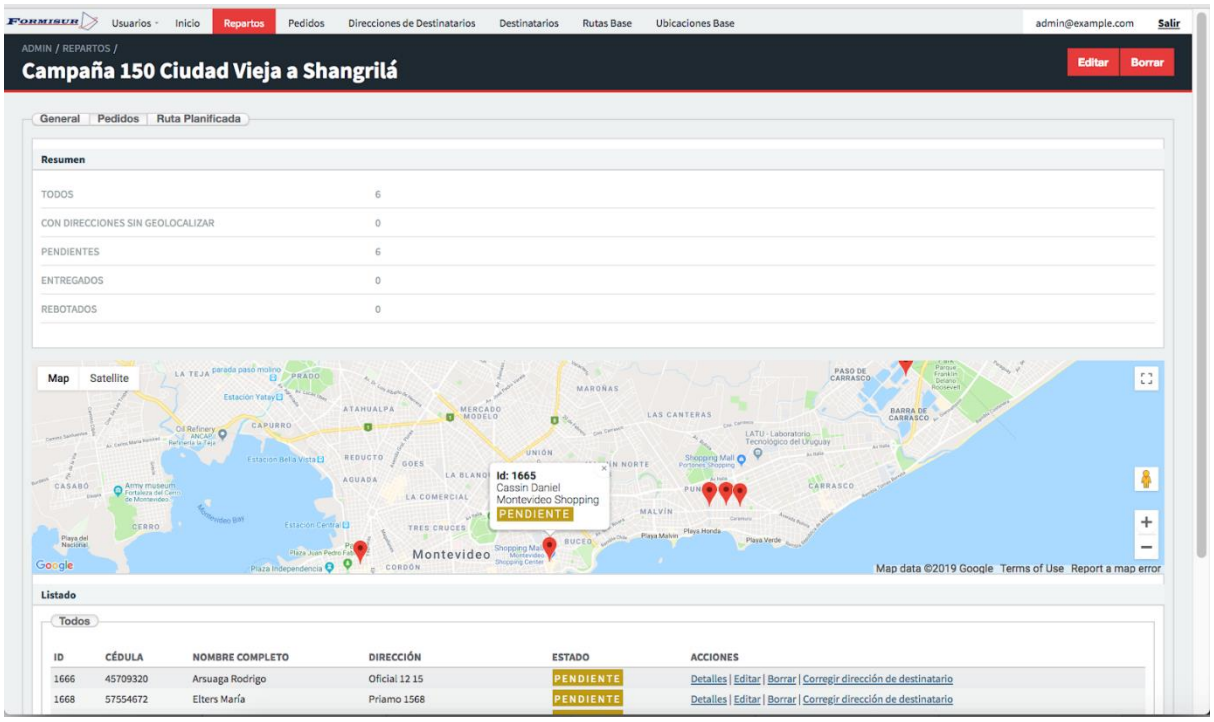


Ilustración 5 - Visualización de listado de pedidos de un reparto

En la pestaña “Última Actualización” se puede visualizar el recorrido realizado por el repartidor junto con los marcadores de las direcciones de los destinatarios.

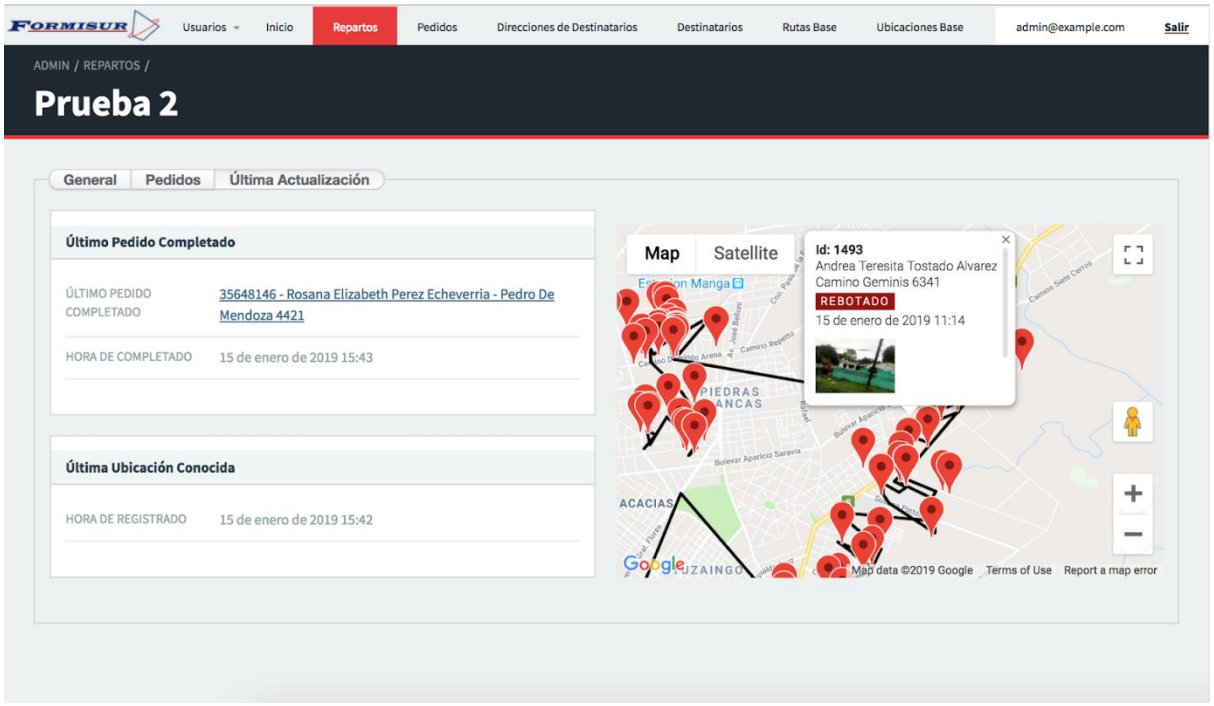


Ilustración 6 - Monitoreo de un reparto

Las alertas de ineficiencia son por ejemplo las alertas de posición, las cuales aparecen si un repartidor marcó un pedido como rebotado estando lejos de la dirección del destinatario. Los pedidos que han sido rebotados lejos de la dirección del destinatario se pueden visualizar de varias maneras: desde el listado de pedidos del reparto, en el menú de pedidos, en los detalles de un pedido o en el inicio de la aplicación.

A continuación se muestra el menú “Inicio” que resume los datos del sistema.

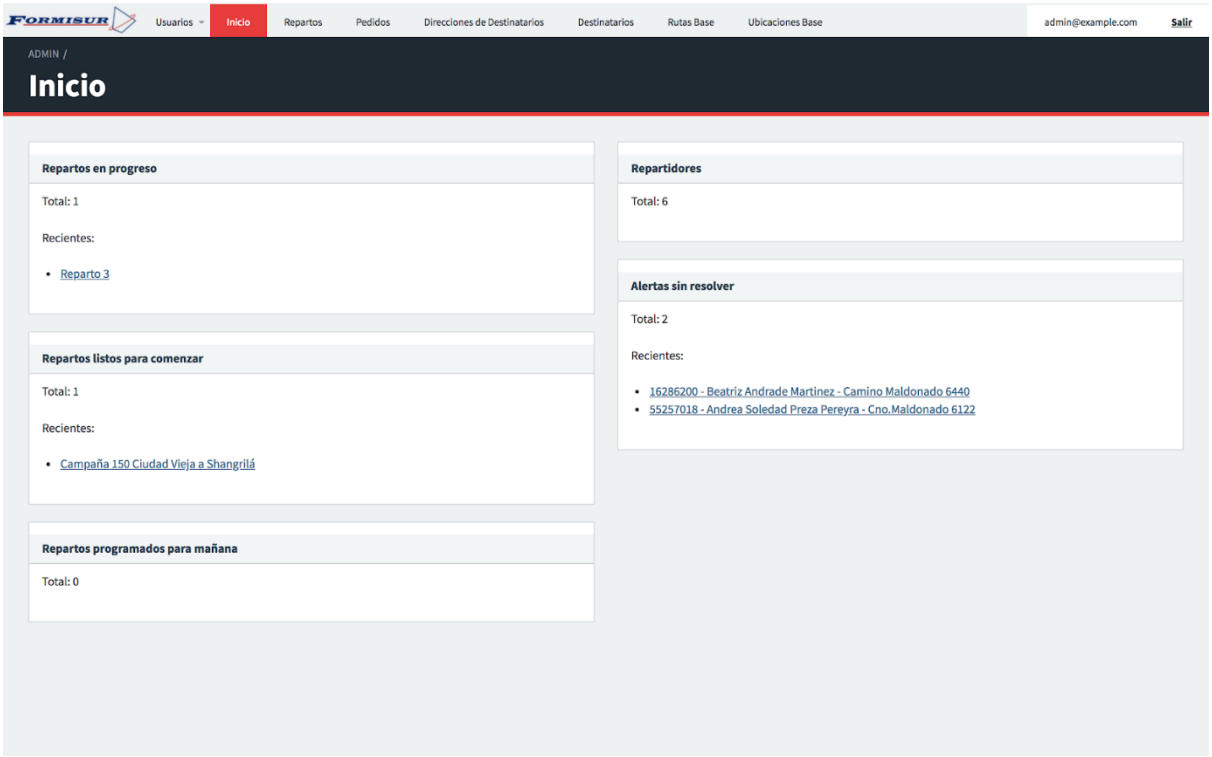


Ilustración 7 - Inicio de la aplicación del administrador

Por otro lado, en la aplicación móvil el repartidor puede comenzar un nuevo reparto.

Ilustración 8 - Pantalla inicial de la aplicación móvil

Una vez comenzado el reparto el repartidor visualiza la lista de pedidos a entregar en orden, aunque también puede postergarlos o completarlos fuera de orden.

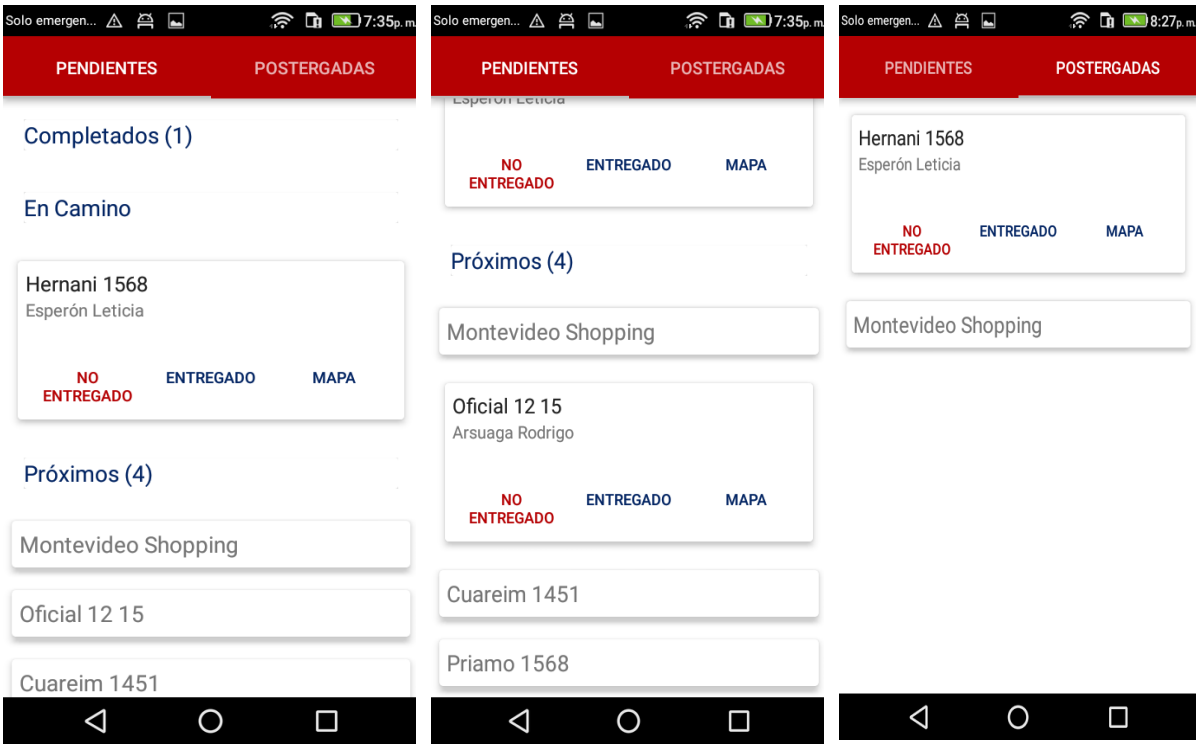


Ilustración 9 - Listado de pedidos en la aplicación móvil

Al entregar un pedido se piden detalles de confirmación. Estos datos dependen de la configuración del reparto y de si el pedido ha sido entregado o rebotado.

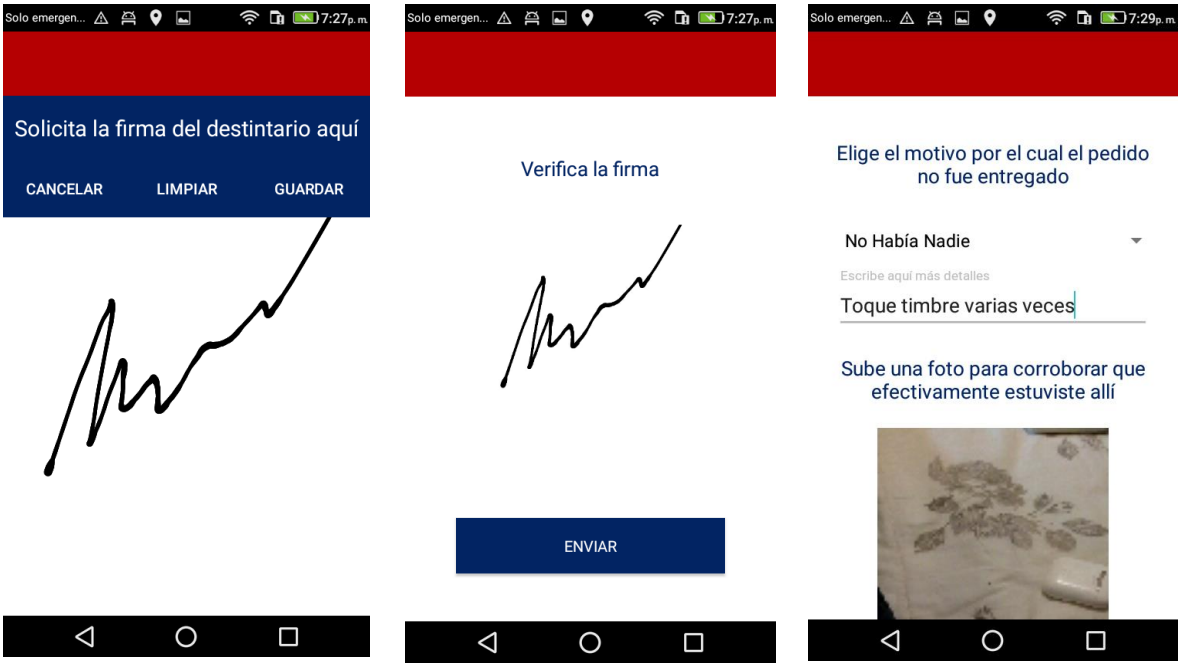


Ilustración 10 - Ejemplo de entrega y rebote de pedido

2.5. Interesados

Esta sección tiene el objetivo de identificar los roles de los interesados del proyecto, analizando su influencia en el proyecto, necesidades, intereses y expectativas para poder considerarlas en la solución a construir.

Martín García (Encargado de Sistemas)

Martín García fue uno de los dos representantes de la empresa para el equipo. Es el Encargado de Sistemas de Formisur y el hijo del director por lo que tiene un amplio conocimiento del dominio e influencia en la empresa. Es Licenciado en Sistemas y ex alumno de la Universidad ORT Uruguay. Su tesis de licenciatura conformó uno de los sistemas que utiliza Formisur.

Al ser quien realice el mantenimiento del producto a construir por el equipo una vez terminado, sugirió los lenguajes de programación a utilizar y expresó desde el principio su expectativa de que el código sea mantenible y siga las buenas prácticas de cada lenguaje.

Por todas estas razones Martín tuvo tanto alto poder como alto interés en lo que al proyecto respecta.

Nicolás Benítez (Administrador)

Nicolás Benítez es el segundo representante de la empresa para el equipo y uno de los administradores. Es quien se encarga de pre procesar el archivo con los pedidos de los destinatarios, subirlos al otro sistema, coordinar y monitorear los repartos, escanear las boletas, etc. Eventualmente también toma el rol de repartidor y trabaja muy en conjunto con ellos.

Nicolás conformaría uno de los usuarios finales del producto a construir, por lo que su opinión fue extremadamente clave para el equipo.

Su motivación principal es lograr un sistema que le permita coordinar los repartos lo más rápido posible, responder a consultas por pedidos y visualizar ineficiencias de los repartidores.

Se consideró entonces que Nicolás presentó un poder medio alto e interés alto en el proyecto.

Repartidores

Se consideró que los repartidores tendrían un interés medio alto en el sistema dado que lo utilizarían diariamente y las nuevas funcionalidades controlarían su actividad.

Aunque no tienen casi poder de decisión en la empresa, el equipo debería asegurarse que dominan la aplicación móvil a construir para minimizar tiempos o demoras en el reparto que impacten negativamente en la eficiencia de entrega.

Director

Se consideró que el director de la empresa tuvo el mayor interés en el proyecto ya la empresa precisaba este sistema para poder concretar el contrato con el nuevo cliente. Sin embargo, el director de la empresa no invirtió tiempo directamente en el proyecto sino que confió plenamente en Martín y Nicolás para representarlo.

De todos modos, el director tendría el poder suficiente para evitar que Nicolás y Martín continúen invirtiendo tiempo en el proyecto o que el software construido se utilice, por lo que además de considerarlo como un interesado importante se gestionó como un riesgo.

Empresa cliente

El sistema de gestión que utilice Formisur impacta directamente en la forma en la que se entregan y monitorean los pedidos de la empresa cliente, por lo que se consideró que estas empresas tienen interés medio alto en el proyecto.

A su vez, al ser pocos clientes pero importantes, sería posible que un nuevo requerimiento solicitado por una empresa cliente importante se tradujera en un nuevo requerimiento del software a construir. Por ello se consideró que estas empresas contaron con un poder medio bajo.

Destinatario final

El destinatario final tuvo un interés bajo en el proyecto ya que este afectaría pero indirectamente la forma en la que se entregan sus pedidos, por ejemplo si se entregan el día programado o se postergan, o si la empresa es capaz de responder a consultas o reclamos correctamente. En cuanto al poder, individualmente los destinatarios tuvieron poder bajo en el proyecto.

A partir de este análisis se graficó la matriz donde se puede visualizar gráficamente el conjunto de interesados que el equipo debería gestionar. En la matriz se graficó a cada interesado con su nivel de interés y poder correspondiente, considerando 1 como bajo y 5 como alto.

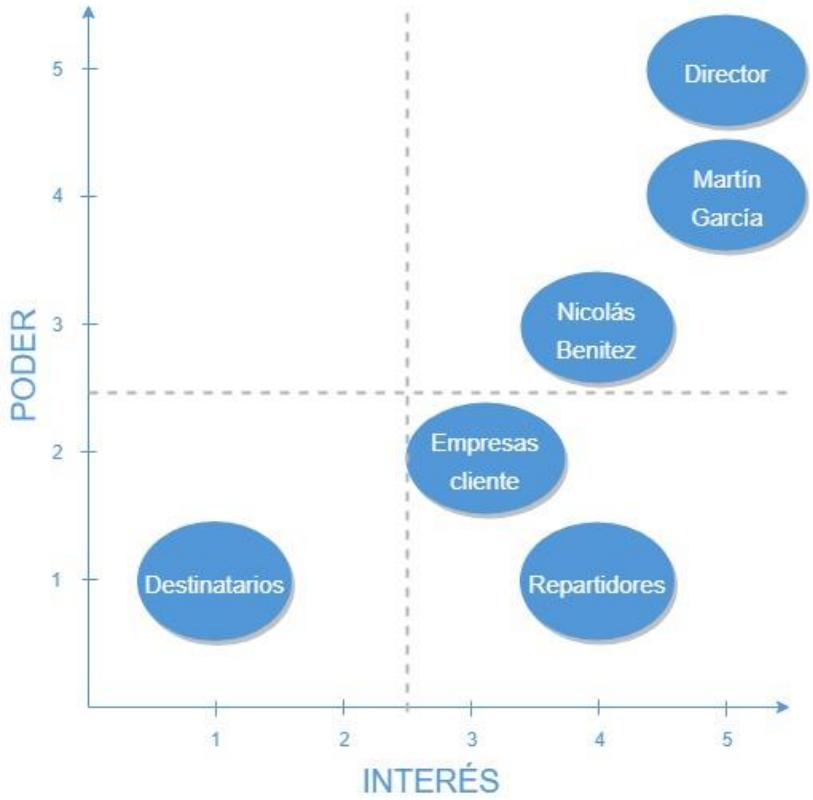


Ilustración 11 - Matriz de interesados

De los cuadrantes de la matriz se desprendió que los interesados que el equipo debió gestionar activamente fueron Martín García y Nicolás Benítez, que además representaron al director de la empresa. Otros grupos de interés que no se ignoraron fueron las empresas cliente y los repartidores. El mínimo esfuerzo del proyecto se dedicaría a los destinatarios.

3. Marco metodológico

En este capítulo se detallan las características del equipo y del proyecto y cómo impactaron en las principales decisiones, como en la metodología, distribución del trabajo y herramientas a utilizar.

3.1. Características del proyecto

A continuación se listan las características del proyecto que impactaron en las decisiones sobre la metodología.

Contacto directo con futuros usuarios del sistema
Uno de los dos representantes de la empresa era el administrador de repartos y ocasionalmente también tomaba el rol del repartidor, por lo que el equipo contó con contacto directo con quien sería un usuario final del sistema. Siendo su opinión tan valiosa, el equipo debería aplicar una metodología que permita sacar provecho de la situación.

Alta disponibilidad de los clientes
El grado de interés y disponibilidad de los clientes en el proyecto fue muy alto, y se consideró un punto clave para tomar las principales decisiones sobre el marco metodológico.

Cliente con conocimiento técnico
Uno de los dos representantes de la empresa contaba con conocimiento técnico de los lenguajes a utilizar, siendo quien mantendría el software a construir en un futuro. El cliente manifestó su interés por permanecer activamente informado de las soluciones a nivel de arquitectura y sobre las librerías o servicios utilizados.

Problema con alto nivel de complejidad
El problema de enrutamiento de las órdenes fue uno de los mayores desafíos técnicos del proyecto. El equipo debió asegurarse que la metodología y el cronograma a seguir tuviesen en cuenta el tiempo de investigación y prueba que esto requeriría.

Reuniones generadoras de muchas ideas
En las primeras entrevistas para conocer el problema y definir la solución, el equipo detectó que las reuniones eran instancias donde se generaban muchas ideas o retroalimentación, algunas de ellas muy valiosas. El marco de trabajo debería sacar provecho de la situación.

3.2. Características del equipo

Tamaño del equipo
El equipo estuvo integrado únicamente por dos estudiantes. Esto podría presentar tanto ventajas como desventajas que deberían ser consideradas a lo largo de todo el proyecto. Aspectos como la comunicación podrían verse favorecidos al ser un equipo pequeño, mientras que la sobrecarga de roles en los estudiantes podría ser un

problema. Sin embargo, el mayor desafío que enfrentaría el equipo fue el saber priorizar muy cuidadosamente, ya que se comprometieron a construir un producto listo para poner en producción al finalizar el proyecto donde la capacidad del equipo era muy limitada.

Conocimiento como equipo
Ambos integrantes habían trabajado juntos en la mayoría de los proyectos obligatorios a lo largo de la carrera, conociendo las fortalezas, debilidades, intereses y motivaciones de cada uno.

Equipo con disponibilidad horaria solapada
El equipo era capaz de reunirse en persona la mayoría de las veces.

Experiencia previa en desarrollo ágil
Ambos integrantes contaban con al menos un año de experiencia previa trabajando bajo el marco de metodologías ágiles, particularmente Scrum, en contextos laborales.

Experiencia previa en desarrollo de aplicaciones
Ambos integrantes contaban con experiencia previa trabajando en pequeños equipos desarrollando aplicaciones móviles. Mientras que Rodrigo había desarrollado habilidades de desarrollo móvil, Leticia de desarrollo back-end.

Aptitudes de los integrantes
Además de conocimientos previos y preferencias técnicas, ambos integrantes presentaban perfiles diferentes. Mientras que el fuerte de Leticia era el relevamiento de requerimientos y arquitectura, Rodrigo presentaba mayores habilidades de comunicación y gestión de clientes.

3.3. Principales decisiones

A continuación se detallan las principales decisiones acerca del marco metodológico tomadas considerando las características del proyecto y del equipo descritas anteriormente.

3.3.1. Fases del proyecto

La primera decisión del equipo fue dividir el proyecto en tres fases de alto nivel: inicial, desarrollo y final como se muestra a continuación.

Ilustración 12 - Duración de las fases del proyecto

A continuación se detallan las tres etapas resumiendo las principales actividades que se realizaron en cada una.

Fase Inicial 26/04/18 - 28/05/18 (4 semanas)

El equipo consideró oficialmente comenzada la etapa inicial del proyecto una semana después del dictado del comité, a partir de la primera reunión oficial con el tutor. En ella se comenzó llevando a cabo tareas de definición de la organización del equipo durante el resto del proyecto. Luego se realizaron repetidas entrevistas con los clientes para lograr comprender el problema y definir según las características del proyecto el marco metodológico más adecuado. Finalmente se procedió a definir la solución, iterando sobre prototipación y validación para concretar la lista de los requerimientos más importantes que integrarían la primera versión del Backlog, para finalmente definir una arquitectura inicial y un plan de calidad acorde.

Desarrollo 28/05/18 - 07/01/19 (30 semanas)

Se dedicó esta etapa al desarrollo de la solución propuesta y se realizaron tareas de control de calidad, *testing*, gestión de la configuración, gestión de proyecto, entre otras. Si bien en la fase inicial se prototiparon y validaron los requerimientos de mayor importancia, en la etapa de desarrollo se realizaron nuevamente instancias de validación utilizando el producto construido hasta el momento, que fueron modificando y agregando los requerimientos restantes al *Backlog*.

Fase final 07/01/19 - 06/03/19 (8 semanas)

Esta fase tuvo como objetivo el cierre del proyecto. En ella se buscó realizar las tareas necesarias para poder determinar el cumplimiento o no con los objetivos iniciales, como la realización de encuestas de satisfacción a los usuarios y clientes.

Luego del armado y recopilación de documentos para la entrega final, el equipo realizó una última retrospectiva, definiendo conclusiones y lecciones aprendidas.

El diagrama a continuación resume las actividades descritas anteriormente.

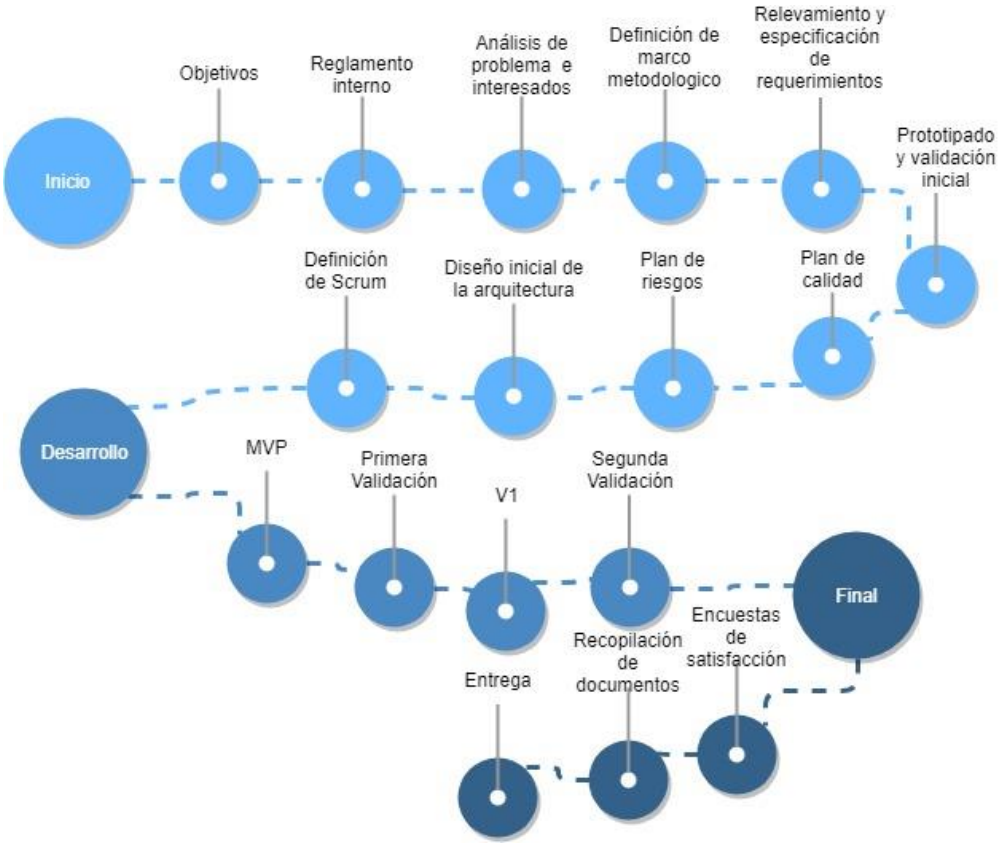


Ilustración 13 - Actividades en cada fase del proyecto

3.3.2. Ciclo de vida

Se decidió trabajar sobre un ciclo de vida iterativo incremental, repitiendo en cada iteración de la etapa de desarrollo un proceso de trabajo similar para proporcionar un resultado completo sobre producto final.

Ilustración 14 - Ciclo de vida

La etapa de desarrollo constó de quince iteraciones. En cada iteración se fue mejorando el producto, añadiendo nuevos requerimientos o mejorando los que fueron implementados. Como se mencionó en las características del proyecto, los clientes se encontraron sumamente interesados e involucrados, por lo que para esta etapa fue clave elegir una metodología que les permitiera observar avances en forma progresiva, satisfaciendo gradualmente su ansiedad y brindando retroalimentación sobre cada iteración.

Ilustración 15 - Fases del proyecto e iteraciones

3.3.3. Metodología Ágil

Una vez que el equipo seleccionó el ciclo de vida, definió utilizar una metodología ágil debido principalmente a las siguientes características descritas:

1. Experiencia previa en desarrollo ágil
2. Alta disponibilidad de los clientes
3. Contacto directo con futuros usuarios del sistema
4. Reuniones generadoras de muchas ideas

El equipo concordó que el trabajar bajo una metodología ágil sacaría provecho del hecho que los clientes estaban interesados, disponibles, llenos de ideas y opiniones y que serían quienes finalmente utilizarían el software.

3.3.4. Scrum

Teniendo en cuenta las características del equipo y del proyecto, se decidió utilizar Scrum sobre otras metodologías ágiles luego de un breve análisis comparativo el cual se encuentra detallado en el ANEXO 1 - “Análisis de metodologías ágiles”.

Además de la experiencia previa con la que los integrantes contaban en esta metodología, Scrum trajo consigo las siguientes ventajas:

Ritmo de trabajo cómodo y constante
Los procesos de Scrum están diseñados de forma que los integrantes puedan trabajar a un ritmo de trabajo cómodo, estando este punto vinculado a uno de los objetivos iniciales del equipo donde se buscó distribuir las horas de trabajo durante todo el proyecto para trabajar tranquilos. El cálculo de la velocidad del equipo fue fundamental para poder estimar un alcance final del producto y trabajar en un ritmo sostenible durante toda la etapa de desarrollo.

***Sprints* de corta duración y retroalimentación continua**
Tener *Sprints* acotados permitió presentarle al cliente avances periódicos de forma que pudiera validar el producto y dar *feedback* constantemente al equipo disminuyendo la probabilidad de que esto implique un cambio radical. Por otro lado, al utilizar *Sprints* cortos el equipo administró y distribuyó el tiempo de mejor manera y permitió reducir el impacto de un atraso en un *Sprint*.

Adaptabilidad
Scrum es un *Framework* por lo que trae consigo más adaptabilidad que muchas otras alternativas, permitiendo que el equipo lo modifique durante el proceso según le dé mejor resultado, mediante la revisión y evaluación interna continua resultado de las reuniones retrospectivas. Si bien el objetivo principal de las retrospectivas que plantea Scrum es evaluar el proceso y la aplicación de la metodología, el equipo las utilizó además para inspeccionar otros aspectos del proyecto como medir la alineación con los objetivos, examinar la evolución de los riesgos, pulir el *Backlog* y ajustar el *Release Plan*.

3.3.5. Equipo de trabajo

Dado a que el equipo estaba conformado por solamente dos estudiantes que se reunieron en persona la mayoría de las veces, la separación de roles no tomó tanta importancia.

Sin embargo, conociendo de antemano las fortalezas y motivaciones de cada uno, algunas tareas fueron previamente asignadas a un integrante que conformaría el rol de líder del área correspondiente. Esto no implicó que fuese el único en realizar las actividades del área pero sí de asegurarse que fuesen llevadas a cabo adecuadamente y/o de definir un plan para ello, con excepción del desarrollo donde los roles se definieron muy claramente.

Los roles definidos se resumen en la tabla a continuación:

Leticia	**Rodrigo**
Desarrolladora *back-end*	Desarrollador de la aplicación móvil
DBA (responsable de base de datos)	*Tester*
Ingeniera de requerimientos	Asegurador de Usabilidad

Tabla 1 - Distribución de roles

Los planes correspondientes a la arquitectura, SCM y SQA se definieron en conjunto y luego cada integrante se aseguró que se cumpliera para el *back-end* o *front-end* según lo asignado.

3.3.6. Reglamento interno

Al comenzar el proyecto el equipo definió un reglamento interno donde acordó prácticas estrechamente alineadas con los objetivos iniciales, a tener en cuenta a lo largo de todas las etapas del mismo para asegurar un clima de trabajo apropiado para desarrollar el mejor proyecto posible. El mismo se detalla en el ANEXO 2 – “Reglamento interno” y en el ANEXO 3 – “Gestión de la comunicación”.

3.3.7. Herramientas utilizadas

Al inicio del proyecto el equipo investigó las principales herramientas a utilizar para la comunicación, gestión, documentación, entre otros, con el objetivo de maximizar la eficiencia y evitar retrabajo en etapas avanzadas del proyecto debido a una pobre decisión de tecnologías. Se puede visualizar un resumen de las mismas en el ANEXO 4 – “Resumen de herramientas utilizadas” y en el ANEXO 5 - “Resumen de justificaciones de herramientas utilizadas”.

4. Ingeniería de requerimientos

En este capítulo se describen las técnicas y herramientas utilizadas durante el relevamiento, especificación, validación y estimación de los requerimientos.

El equipo definió los requerimientos más importantes en la fase inicial, permitiendo que el resto se fueran definiendo y priorizando a partir de retroalimentación que surgió durante las *Sprint Reviews* y las pruebas de validación.

4.1. Relevamiento y análisis

En primer lugar se buscó comprender el negocio y el proceso de los repartos para identificar a los involucrados y asegurarse de entender sus necesidades. La principal estrategia de relevamiento fue a través de reuniones y entrevistas, aunque se utilizaron otras técnicas como mapa de actores y mapas mentales para plasmar los datos.

A continuación se describen las técnicas y herramientas aplicadas durante esta etapa.

4.1.1. Mapa de actores

Para poder visualizar gráficamente cómo se relacionan e interactúan entre sí los interesados, se realizó un mapa de actores.

Como se describió en la sección Problema y Solución, los interesados más importantes fueron: Martín García, Nicolás Benítez, repartidores, empresas cliente, destinatarios finales y dueño de Formisur.

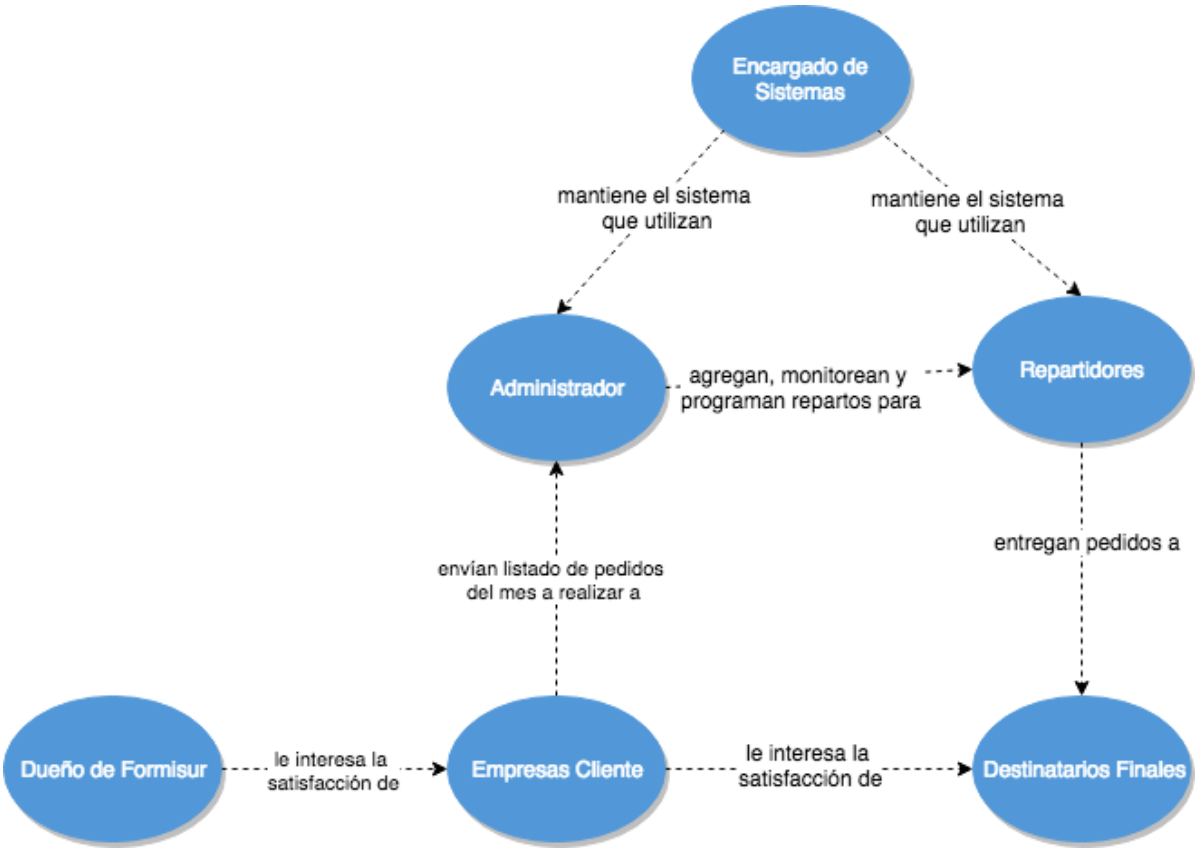


Ilustración 16 - Mapa de actores

4.1.2. Reuniones y entrevistas con interesados

Los dos representantes de la empresa con los que el equipo contó en las reuniones durante todo el proyecto fueron Martín García y Nicolás Benítez. Con ellos se mantuvieron varias reuniones con el fin de entender a fondo el problema y sus expectativas.

El equipo también tuvo la oportunidad de realizar una breve entrevista a un representante de una empresa cliente. Se buscó entender las expectativas de estas empresas sobre la empresa de logística que contratan, buscando descubrir discordancias entre lo que esperan versus lo que se ofrece para identificar oportunidades de mejora a través del software a construir.

Ilustración 17 - Reunión con interesados

4.1.3. Mapa mental

Luego de las entrevistas para entender el negocio y el problema, los integrantes del equipo habían reunido una colección de apuntes con grandes cantidades de información desorganizada. Decidieron realizar un mapa mental como un intento de ordenar y relacionar los datos obtenidos, y además seleccionar aquellos que fuesen importantes para el diseño de la solución.

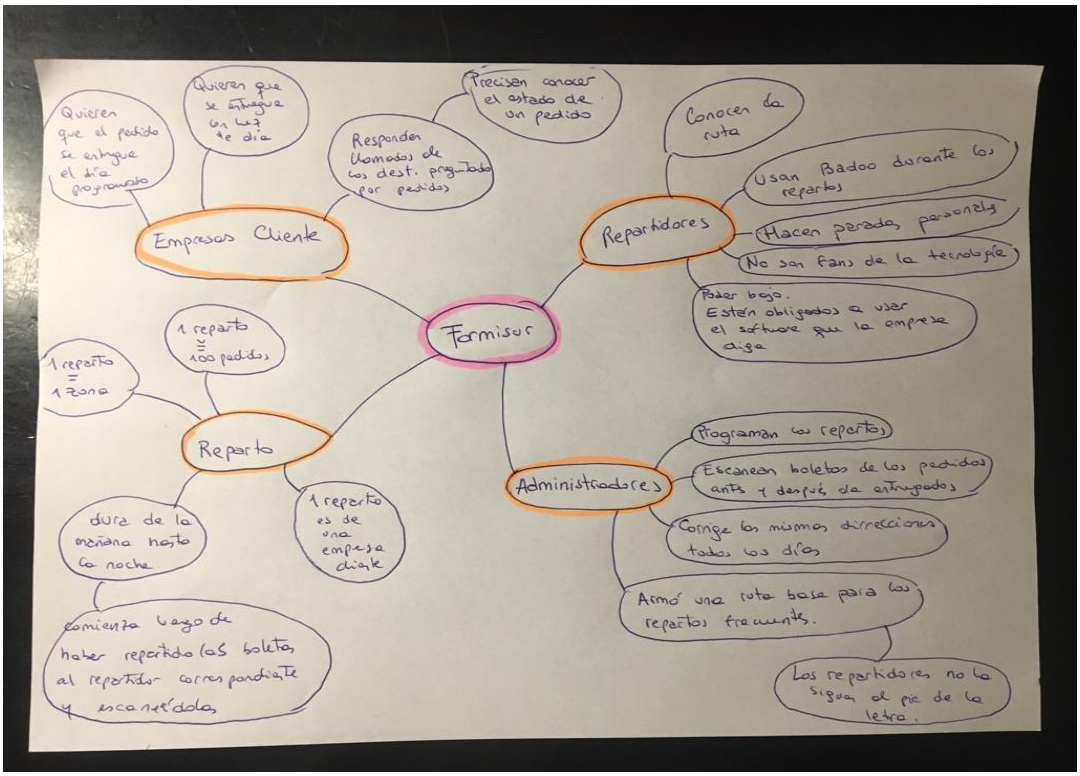


Ilustración 18 - Mapa mental del problema

El equipo también utilizó esta técnica más adelante para identificar los puntos claves de la solución a construir.

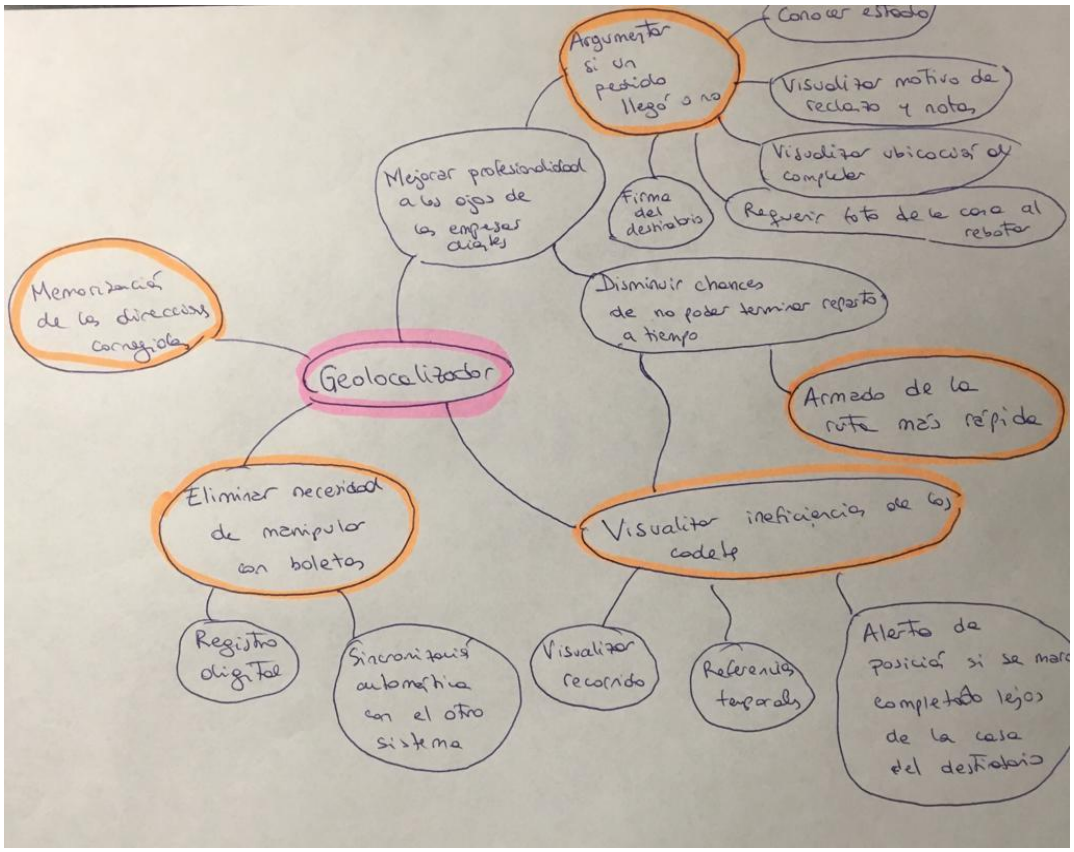


Ilustración 19 - Mapa mental de la solución

4.2. Especificación

Durante esta etapa se buscó definir el comportamiento del sistema a construir, a partir de la información obtenida en la etapa de relevamiento y análisis.

Las dos grandes técnicas utilizadas para la especificación de requerimientos fueron *Story Mapping* [1] e historias de usuario con criterio de aceptación.

El *Story Mapping* se intercaló con prototipación para validar progresivamente la solución diseñada. Primero se construyó lo que se denomina el "*back-bone*" del *Story Map*. Esto es, el flujo principal de la aplicación, donde se especifican las acciones de los actores ordenadas cronológicamente. Construyendo el *back-bone* se logró definir un flujo a alto nivel de las acciones a realizar y se hicieron prototipos en papel de la parte de la aplicación móvil. Una vez validada la idea del flujo principal con los clientes, se procedió con el resto del *Story Map* y se realizaron prototipos más elaborados.

A continuación se detalla en qué momento y de qué forma se utilizó cada técnica.

4.2.1. Story Mapping

El equipo utilizó esta técnica con el objetivo de facilitar la planeación de los *Releases*. Al ser un equipo de solo dos integrantes, la correcta priorización de los requerimientos fue clave para lograr un producto de valor en el tiempo limitado del proyecto.

De esta manera, una vez comprendido el problema el equipo construyó un *Story Map* donde se buscó diseñar el flujo principal del sistema a construir, intercalando las acciones necesarias del administrador y del repartidor.

Utilizando *Post-its*, el equipo plasmó horizontalmente la línea del tiempo de las tareas de cada uno de los actores, como “cargar archivo de pedidos", englobadas en un objetivo de más alto nivel, como “programar un reparto”.

Los objetivos de alto nivel que englobaron las tareas eran: Manejar sesión (administrador), administrar repartidores (administrador), manejar sesión (repartidor), crear reparto (administrador), realizar reparto (repartidor), monitorear reparto (administrador).

Ilustración 20 - *Back-bone* del *Story Map*

A partir de este flujo se elaboraron los primeros prototipos de la aplicación móvil. Una vez validado el *back-bone* con los clientes, se procedió a extraer los requerimientos a partir de las tareas en los *Post-its*.

En el contexto del *Story Mapping*, los requerimientos son las diferentes alternativas de realizar la tarea en cuestión como “cargar desde un *Excel*", “cargar desde un archivo de texto", o “importar desde otro sistema”. Los requerimientos se definen y anotan en *Post-its* y se pegan verticalmente debajo del *Post-it* de la tarea correspondiente. El equipo definió entonces las formas alternativas para realizar cada tarea (requerimientos).

No todos los requerimientos pensados entraron al *Backlog* inicial. De hecho, se buscó priorizar horizontalmente de forma de lograr en el MVP completar el flujo principal lo antes posible, permitiendo realizar las tareas cruciales de una sola forma. El equipo fue acomodando los *Post-its* con los requerimientos de arriba hacia abajo considerando su valor y una primera consideración del esfuerzo que tomarían y así determinaron realizar dos *Releases*. El resto de los requerimientos (aquellos que no

fueron planificados para ni el primer *Release* ni en el segundo) se colocaron en la tercer fila (*Icebox*).

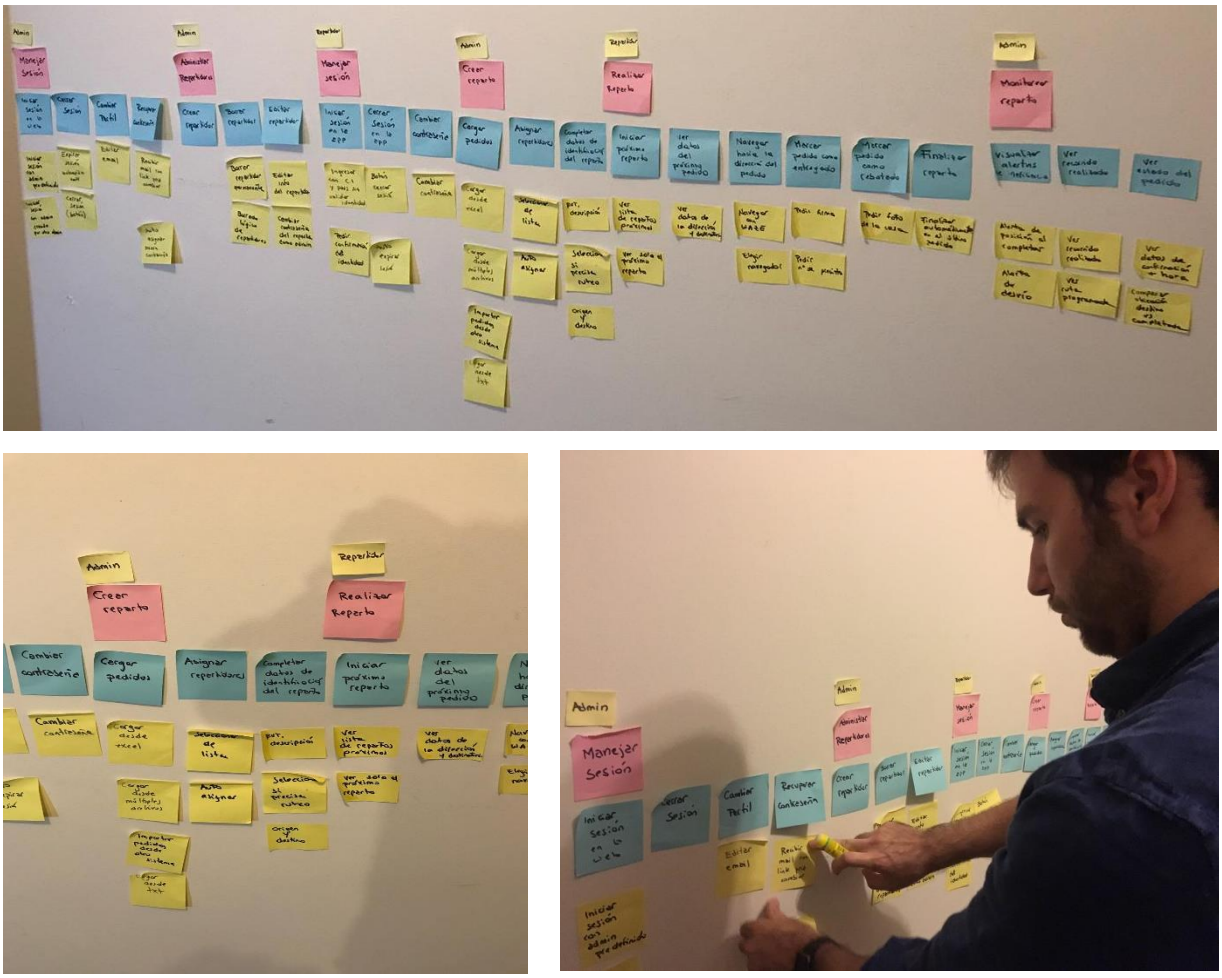


Ilustración 21 - Construcción del *Story Map*

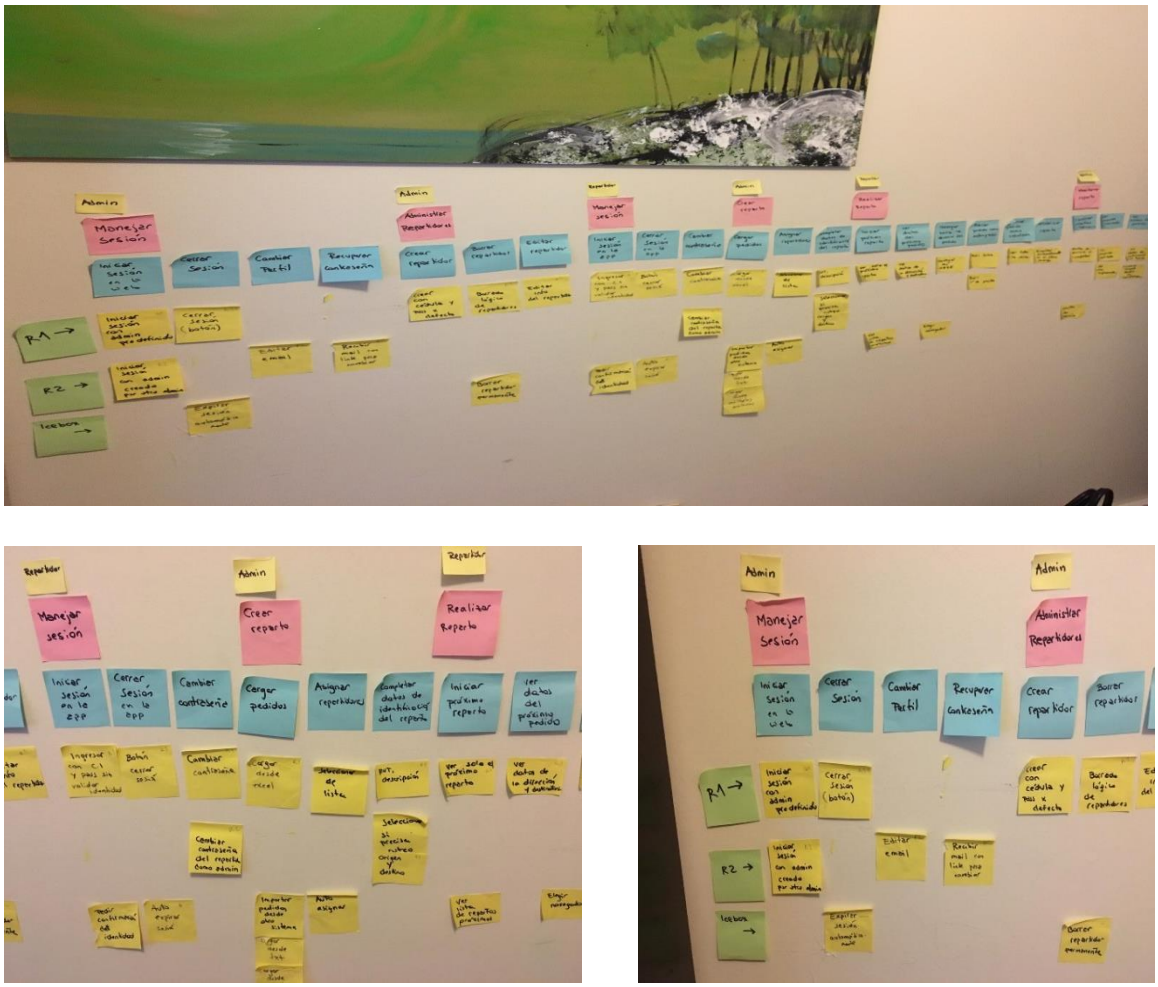


Ilustración 22 - Releases a partir del *Story Map*

4.2.2. Historias de usuario

Los requerimientos se especificaron utilizando historias de usuario con criterio de aceptación.

Las historias se titularon con el formato “Como <usuario> debo <realizar tarea>“. El valor que agrega la tarea para el usuario se escribió en la descripción de la tarjeta.

Se permitió definir el criterio de aceptación de distintas maneras según fuese más apropiado. Para historias con complejidad media alta el criterio de aceptación llevó el formato “Dado, Cuando, Entonces” (en inglés “*Given, When, Then"*) pero en otras historias se permitieron formatos menos estrictos por ejemplo “El listado debe ordenarse según fecha de creación, recientes primero”.

Para gestionar las historias de usuario se utilizó la herramienta Trello [2]. La razón por la que se eligió Trello sobre otras alternativas se encuentra en el ANEXO 29 – “Análisis de herramientas de gestión”.

A continuación se muestra un ejemplo de un requerimiento especificado en Trello, con su respectivo criterio de aceptación.

Ilustración 23 - Especificación de un requerimiento con criterio de aceptación

4.3. Validación

El objetivo de esta etapa fue obtener retroalimentación de la solución en diferentes etapas del proyecto.

Se realizó validación antes de comenzar a desarrollar utilizando los prototipos. También se realizaron dos instancias de pruebas en repartos reales, donde la versión del software construida hasta el momento fue utilizada por el administrador y los repartidores para llevar a cabo todo el proceso de planificación y entrega de los pedidos.

4.3.1. Prototipación

A continuación se describen brevemente los diferentes prototipos que se llevaron a cabo a lo largo del proyecto y cómo se utilizaron para validar el software a construir. En el ANEXO 6 – “Prototipación” se describen más detalladamente los prototipos. En el ANEXO 7 – “Resultado de pruebas de los prototipos con usuarios” se registran las conclusiones extraídas de las pruebas realizadas con ellos.

El equipo definió un criterio de aceptación para los prototipos iniciales, concordando que entre 2 y 8 acotaciones por pantalla sería un número aceptable de modificaciones. Esto fue porque se consideró que menos de 2 sugerencias por pantalla es improbable ya que nadie acierta en el primer prototipo y significaría que los clientes no prestaron atención, no entendieron o no se sienten cómodos para opinar. Por otro lado, más de 8 sugerencias por pantalla podrían significar que la solución definida por el equipo

estaba equivocada. En cualquiera de los dos casos, el equipo debería comenzar de nuevo. Los resultados dieron entre 2 y 6 sugerencias por pantalla, lo cual cumplió con el criterio de éxito establecido.

Prototipo inicial de la aplicación móvil

Una vez que el flujo principal de la aplicación móvil fue discutido y acordado con los representantes de Formisur, el equipo procedió a realizar un prototipo inicial utilizando la herramienta Invision [3]. Esta herramienta permitió vincular las pantallas de forma que cada botón programado abra la próxima correspondiente pantalla simulando funcionar.

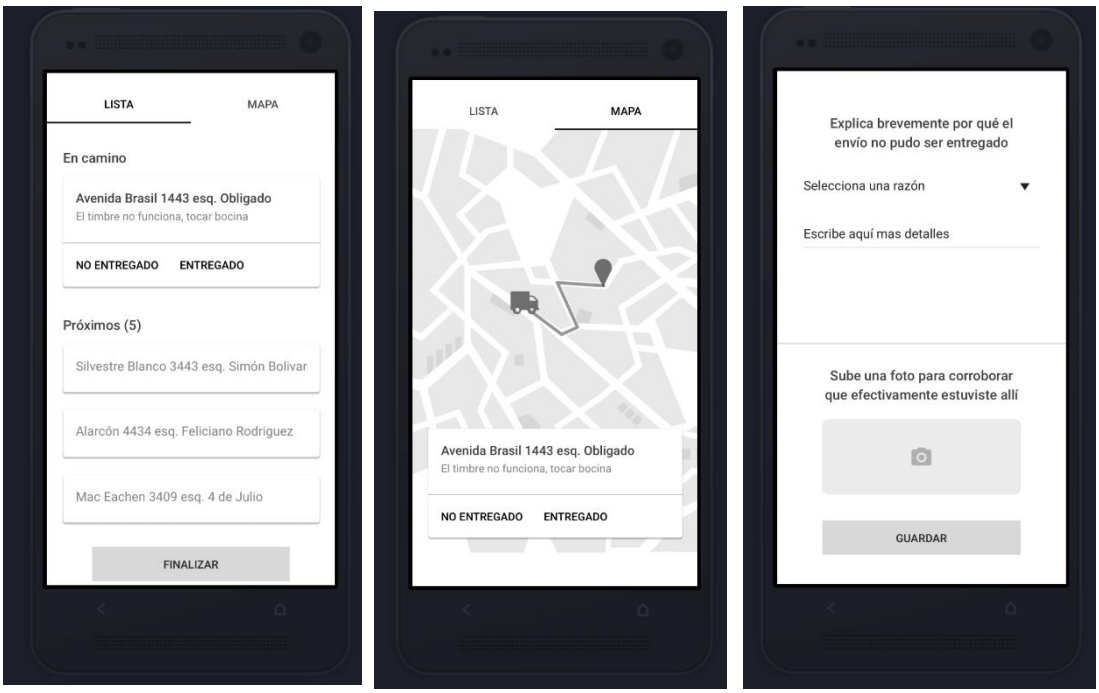


Ilustración 24 - Prototipos iniciales de la aplicación móvil

Se le pidió a Nicolás Benítez que navegue a través del prototipo intentando completar un reparto y explicando lo que entendía de cada pantalla.

Nicolás Benítez y Martín García encontraron el prototipo intuitivo e hicieron algunas recomendaciones. Por ejemplo, unas de las acotaciones que se tuvieron en cuenta para el próximo diseño fue que sería conveniente poder visualizar la cantidad de pedidos que se van completando y que la pestaña “Mapa" no debería tomar tanta importancia como el equipo había pensado en un principio.

El equipo procedió a continuar con el *Story Map* y a agregar el resto de los requerimientos al prototipo.

Segundo prototipo de la aplicación móvil

Considerando las acotaciones del primer prototipo, el equipo entregó el proyecto de Invision a una diseñadora que agregó los estilos a la aplicación utilizando el mismo diseño que la nueva página web de Formisur.

Esta segunda versión de los prototipos fue nuevamente presentada a los usuarios.

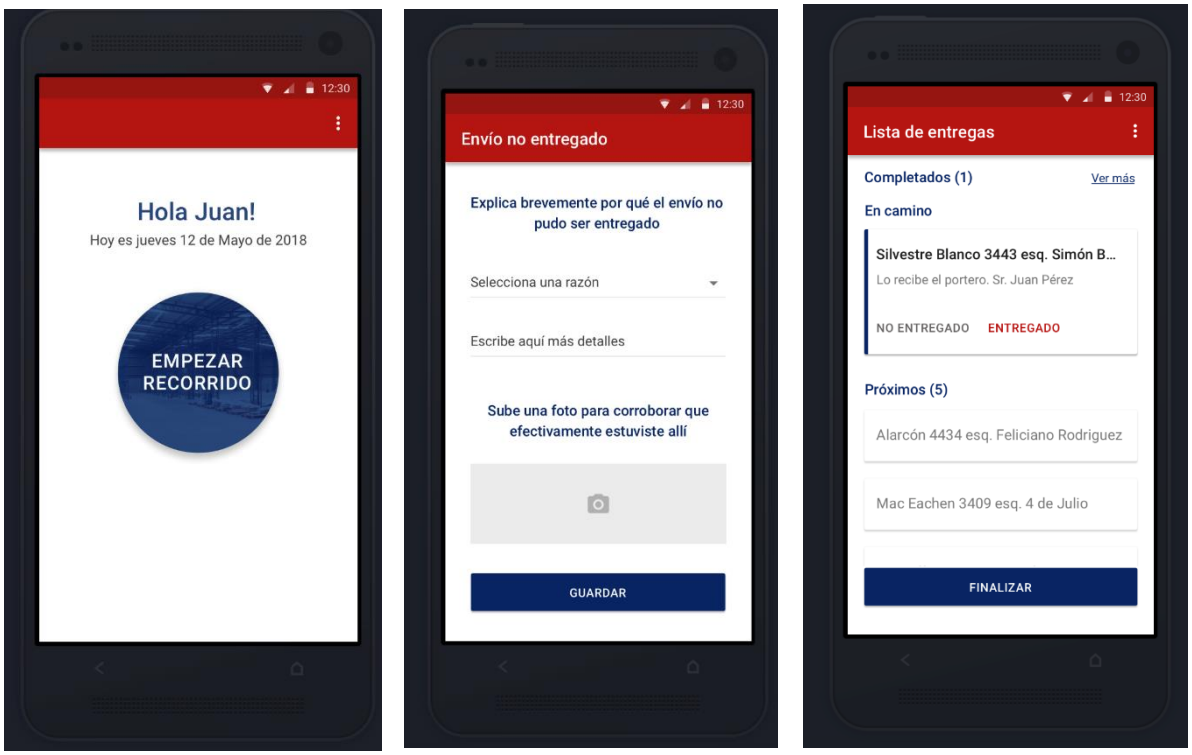


Ilustración 25 - Segunda versión del prototipo de la aplicación móvil

Prototipos de la aplicación web

Para prototipar la aplicación web se utilizó la misma herramienta que se utilizó para maquetar la interfaz final. Antes de comenzar el desarrollo se armaron rápidamente algunas pantallas importantes para la aplicación, como la de crear un reparto. Si bien estas no funcionaban permitieron al equipo mostrar fácilmente lo que tenían en mente y al administrador brindar retroalimentación.

Durante el desarrollo se realizaron prototipos funcionales para comprobar si los requerimientos relevados se habían implementado correctamente. El prototipo se fue incrementando a medida que se agregaron y modificaron funcionalidades.

Como Nicolás Benítez estuvo presente durante las *Sprint Plannings* y *Reviews*, el prototipo se fue validando iterativamente en todas las reuniones donde se discutía o mostraba algún nuevo requerimiento para la aplicación web. Esta agilidad fue extremadamente útil para el equipo ya que brindó retroalimentación de a poco y muy frecuentemente, reduciendo la probabilidad y cantidad de retrabajo al finalizar el proyecto.

A continuación se muestra una captura de un prototipo para la funcionalidad de creación de un reparto realizado a inicios del proyecto. Hoy en día esta pantalla se ve parecida pero agrega muchas otras opciones y difiere radicalmente en cuanto a estética.

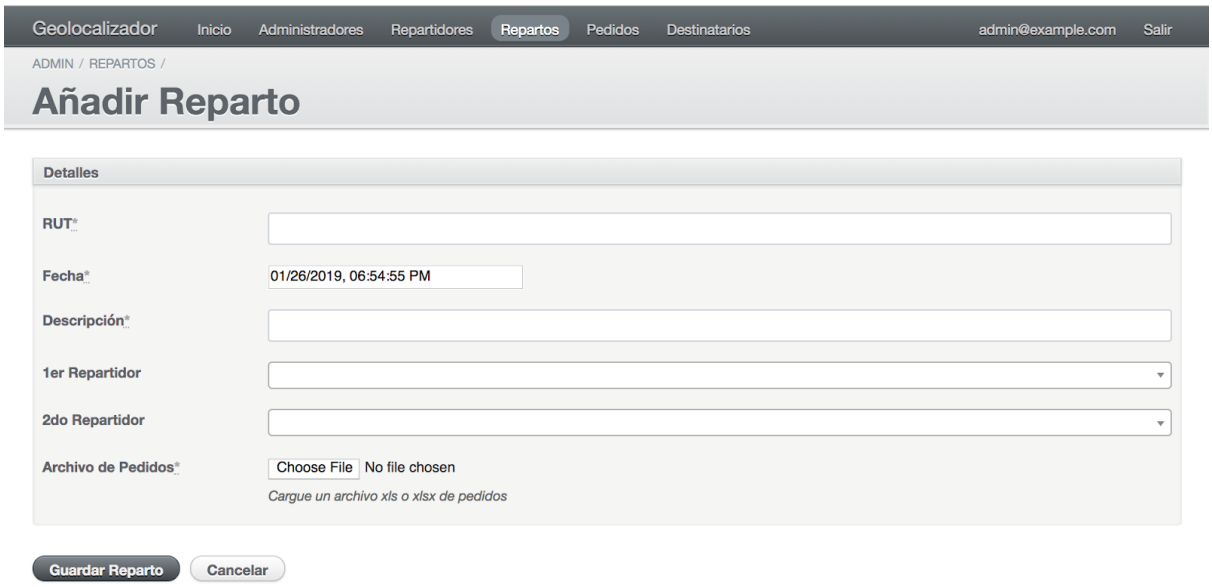


Ilustración 26 - Prototipo de la aplicación web

4.3.2. Pruebas de usabilidad

Se realizaron pruebas de usabilidad de tanto la aplicación móvil como la aplicación web con el objetivo de obtener retroalimentación que permita construir el sistema con el mayor valor posible para los usuarios finales. Para ello se utilizaron los prototipos o, en etapas más avanzadas del proyecto, la última versión del sistema construido.

En ambos casos se le pidió al administrador crear y al repartidor completar un reparto, navegando por las pantallas explicando qué entendían en cada una. Los integrantes del equipo observaron al usuario utilizando la aplicación, atentos a errores y dudas que fueron presentando.

Los resultados de las pruebas se encuentran en el ANEXO 7 – “Resultado de pruebas de los prototipos con usuarios” y en el ANEXO 8 – “Pruebas con los usuarios en repartos reales”.

Ilustración 27 - Pruebas con usuarios

4.4. Estimación

La estimación del esfuerzo de los requerimientos fue un punto clave en el proyecto para obtener una fecha para el primer *Release* y delimitar el alcance final del segundo.

A continuación se describe las técnicas y el proceso utilizados para estimar los requerimientos.

4.4.1. Puntos de esfuerzo

El equipo decidió estimar el esfuerzo de los requerimientos utilizando puntos de esfuerzo. Se decidió utilizar esta unidad y no otra como horas ya que el equipo no conocía aún su velocidad de trabajo, sobre todo en la parte móvil donde el desarrollador no contaba con tanta experiencia previa en el lenguaje. Por eso, se definió el esfuerzo relativamente y se permitió al equipo ir descubriendo su velocidad durante el proceso.

Por otro lado, se acordó utilizar la escala de Fibonacci [4] para asignar puntos. Se decidió seguir esta escala considerando el hecho de que al aumentar el tamaño de un requerimiento aumenta también el margen de error de su estimación. Esto significa que para un requerimiento de mayor tamaño una diferencia de 1 punto es muy difícil de predecir y Fibonacci reduce esa presión distanciando cada vez más los posibles puntajes que se pueden asignar.

Los puntos de esfuerzo se determinaron mediante comparación, asignando un puntaje arbitrario a un requerimiento y comparando desde allí. Se tomó como referencia la siguiente funcionalidad: “Como admin debo poder crear un repartidor” la cual se

consideró de esfuerzo medio y se le atribuyeron 3 puntos. El equipo también acordó criterios como “si implica un nuevo modelo en la API son al menos 2 puntos” o “si implica una nueva pantalla en la aplicación móvil son al menos 3 puntos”.

4.4.2. Spikes

Se decidió hacer uso de *Spikes* de investigación para requerimientos donde no se contó con suficiente conocimiento como para estimarlos de antemano.

Los *Spikes* otorgaron al equipo tiempo suficiente para investigar correctamente el problema a atacar, con el fin de definir la mejor solución posible y estimarla, organizando las tareas a llevar a cabo para fomentar la productividad.

Se utilizó principalmente para la funcionalidad del enrutamiento de pedidos, donde se planificó una historia de investigación en un *Sprint* previo a su realización.

Al comenzar el *Spike*, el equipo definió el criterio de aceptación y una lista de tareas a realizar al igual que con todas las otras historias de usuario.

En el criterio de aceptación se expresó el resultado esperado del *Spike*, que fue un documento resumiendo la investigación, un plan con la solución a utilizar para resolver el problema, y su correspondiente estimación.

En la lista de tareas se incluyeron puntos como realizar entrevistas a expertos, intentar conocer cómo lo solucionan otras empresas, investigar servicios en la nube, investigar alternativas manuales utilizando algoritmos, explorar posibles fuentes para obtener las matrices de costos, entre otras tareas que se agregaron en el proceso.

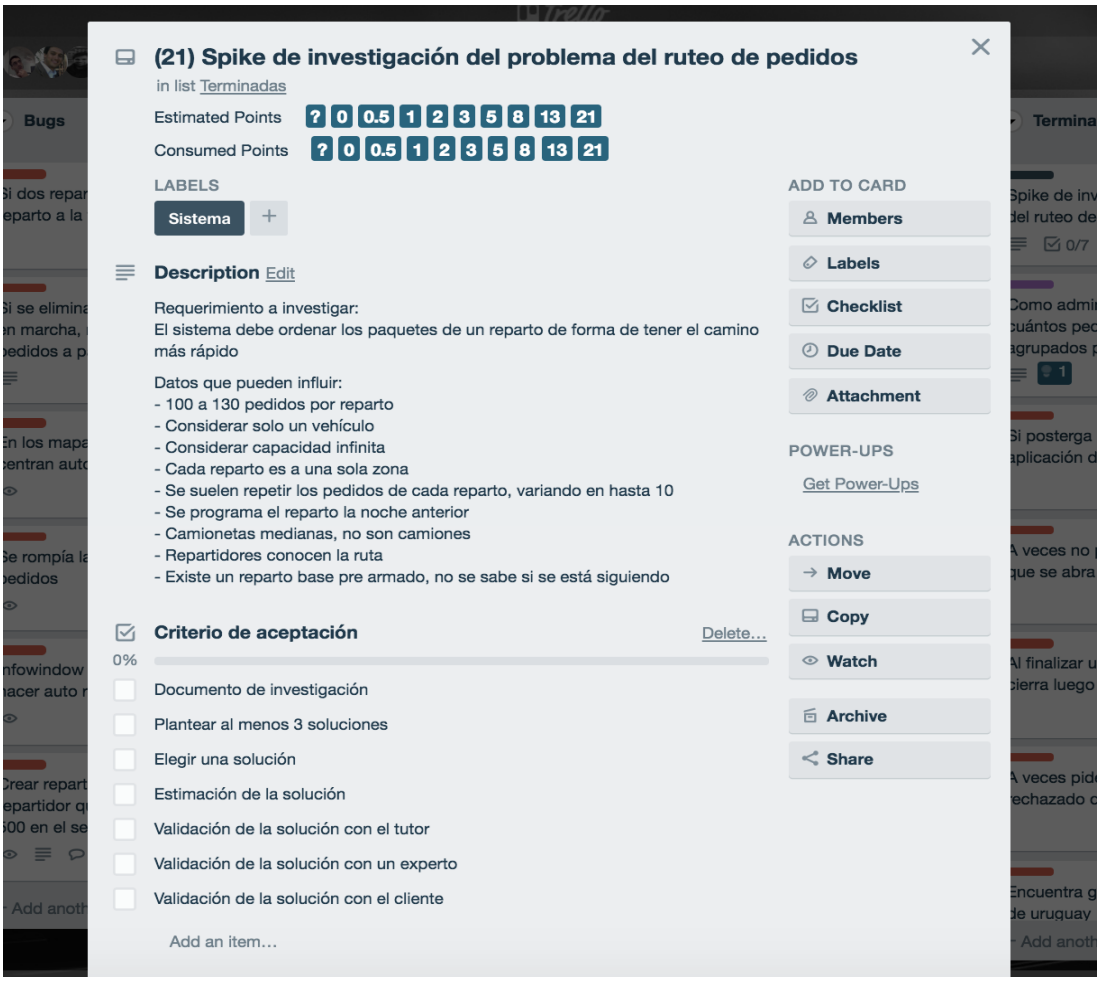


Ilustración 28 - Planificación de un *Spike* de investigación

4.5. Lista de requerimientos

La lista de requerimientos relevada en un principio fue modificada a lo largo del proyecto. En esta sección se resume la lista final de los requerimientos construidos. Los requerimientos en formato historia de usuario se detallan en el ANEXO 9 – “Product Backlog”.

4.5.1. Requerimientos funcionales

Los requerimientos funcionales originalmente se fueron escribiendo en formato de historias de usuario pero a continuación se realiza un resumen de los mismos. En la mayoría de los casos se agrupan varios requerimientos con el fin de agilizar la lectura.

Aplicación web para el administrador

Sesión del administrador

El sistema debe permitir al usuario administrador ingresar a la web con correo electrónico y contraseña. El primer administrador es creado por consola y luego él puede administrar los demás usuarios desde la aplicación. El administrador debe poder también cerrar sesión y restaurar su contraseña.

Administración de usuario administradores

El sistema debe permitir dar de alta, modificar, ver detalles o eliminar usuarios administradores.

Administración de repartidores

El sistema debe permitir dar de alta, modificar, ver detalles o eliminar repartidores. En el caso del alta se debe incluir nombre, apellido, cédula y contraseña. En la modificación no se puede editar la cédula ya que es identificatoria.

Administración de destinatarios

El sistema debe permitir dar de alta, modificar, ver detalles o eliminar destinatarios. Solo se puede borrar un destinatario que no esté presente en ningún reparto ya que se quiere mantener un histórico completo y consistente. En la función de ver detalles se debe poder visualizar todas las direcciones de destinatario que tiene asociadas.

Administración de direcciones de destinatarios

El sistema debe permitir dar de alta, modificar, ver detalles o eliminar direcciones de destinatarios. Solo se puede borrar una dirección de destinatario que no esté presente en ningún pedido ya que se quiere mantener un histórico completo y consistente. En la función de ver detalles se debe poder visualizar la ubicación en formato de mapa. En la función de editar se debe poder corregir la dirección, permitiendo seleccionar una dirección a mostrar al repartidor y corregir las coordenadas seleccionando la posición en un mapa.

Administración de ubicaciones base

Las ubicaciones base representan los locales de Formisur de donde salen o llegan los repartidores. El sistema debe permitir dar de alta, modificar, ver detalles o eliminar ubicaciones base. Solo se puede borrar una ubicación base que no esté presente en ningún reparto ya que se quiere mantener un histórico completo y consistente. La posición de una ubicación base debe seleccionarse en un mapa.

Administración de rutas base

Las rutas base representan direcciones de destinatarios en un determinado orden y las ubicaciones base de origen y destino. El orden representa el enrutamiento de los

pedidos. El administrador debe poder visualizar el orden de la ruta base en forma de listado ordenado y gráficamente como recorrido en un mapa.

Listados

El sistema debe permitir al administrador visualizar listados de los usuarios administradores, repartidores, pedidos, repartos, destinatarios, direcciones de destinatarios, ubicaciones base, rutas base y cualquier otro modelo que pueda administrar. Los listados de los recursos deben estar paginados, mostrar solamente un resumen de los datos más importantes e incluir accesos directos a las acciones más relevantes sobre cada registro como ver, editar y borrar.

Filtrados

El sistema debe permitir el filtrado de usuarios administradores, repartidores, pedidos, repartos, direcciones de destinatarios, destinatarios, ubicaciones base, rutas base y cualquier otro modelo que pueda administrar.

Borrados lógicos

En los casos de repartidores, repartos y pedidos, el borrado debe ser lógico. Esto es, el administrador debe ser capaz de visualizar los registros previamente borrados y de restaurar un registro.

Creación de un reparto a partir de un archivo de pedidos

El sistema debe permitir al administrador crear un reparto desde la aplicación web ingresando entre otros datos el RUT del cliente, fecha y hora, descripción, repartidores asignados, ubicación base de origen, ubicación base de destino, etc. El administrador debe seleccionar un archivo de pedidos en formato Excel a partir del cual se crearán los pedidos del reparto, cada uno con su dirección de destinatario correspondiente que puede existir previamente o ser creada automáticamente con los datos del archivo.

Requerir número de precinto opcionalmente

El sistema debe permitir a un administrador marcar en el reparto que los pedidos de este deben requerir número de precinto al completarse. Este valor debe poder ser sobrescrito para un pedido en particular.

Enrutamiento de pedidos opcional

El sistema debe permitir a un administrador seleccionar si quiere que las direcciones de un reparto sean reordenadas para obtener el camino con el menor tiempo de traslado posible en la calle.

Requerir sincronización con sistema externo de pedidos opcionalmente

El sistema debe permitir a un administrador seleccionar si quiere que los pedidos del reparto sean actualizados contra el sistema externo de pedidos. En caso afirmativo, el administrador debe poder visualizar el ID externo del pedido que se obtiene del Excel y si se pudo sincronizar o no. También debe poder filtrar los pedidos que no fueron correctamente sincronizados y reintentarlos.

Ver datos de confirmación de pedidos rebotados y entregados

El sistema debe permitir al administrador visualizar el estado de los pedidos de un reparto fácilmente. En los detalles de los pedidos entregados se debe poder ver la firma del destinatario y el número de precinto si correspondía. En el caso de rebotado se debe ver el motivo seleccionado (que es de una lista fija de motivos posibles), notas y foto de la casa para verificar que el repartidor efectivamente estuvo allí.

Ubicación al entregar un pedido

En tanto los pedidos rebotados como entregados el sistema debe permitir al administrador visualizar la ubicación en la que se marcó al pedido como completado, en un mapa, comparándola con la dirección del destinatario que estaba registrada en el sistema, y la hora en la que se completó.

Alertas de posición

El sistema debe permitir a un administrador visualizar y resolver alertas creadas por pedidos que fueron rebotados lejos de la dirección del destinatario. El radio de la alerta debe ser configurable desde una variable de entorno. Las alertas se deben visualizar en el inicio con el resumen de los datos, en los datos del pedido, o filtrando los pedidos con alertas.

Tiempo total del reparto

El sistema debe permitir a un administrador visualizar el tiempo total que tomó completar un reparto, para poder medir el rendimiento del repartidor y comparar rutas base.

Seguimiento del recorrido en tiempo real

El sistema debe permitir a un administrador visualizar el recorrido que realizó el repartidor, con referencias temporales en los pedidos que se van completando. Las sincronizaciones deben ser periódicas y solo si se contaba con conectividad. El administrador debe poder visualizar rápidamente la última posición conocida del repartidor y hace cuánto tiempo fue. También debe poder visualizar el último pedido completado y hace cuánto tiempo fue.

Aplicación móvil para el repartidor

Sesión del repartidor

El sistema debe permitir al repartidor iniciar sesión con su cédula y contraseña a la aplicación móvil, cerrar sesión y cambiar su contraseña. El sistema debe obligar a un repartidor a cambiar su contraseña en su primer inicio de sesión.

Iniciar reparto

El sistema debe permitir a un repartidor ver si tiene un reparto para hoy pronto para iniciar y comenzarlo.

Ver pedidos de reparto en curso

El sistema debe permitir a un repartidor visualizar los pedido a completar, en orden. Debe ver los detalles del siguiente pedido a realizar, la cantidad pendientes y la cantidad completados.

Finalización automática de reparto

El sistema debe finalizar el reparto automáticamente al completar el último pedido.

Rebotar pedido

El sistema debe permitir a un repartidor marcar un pedido como rebotado en caso de no haber podido entregarlo, pidiendo todos los datos de confirmación que apliquen.

Entregar pedido

El sistema debe permitir a un repartidor marcar un pedido como entregado, pidiendo todos los datos de confirmación que apliquen.

Navegación hacia pedido

El sistema debe permitir a un repartidor utilizar un herramienta de navegación, Waze, para visualizar el camino al próximo pedido.

Postergar pedido

El sistema debe permitir a un repartidor postergar un pedido deslizando la tarjeta del pedido hacia la izquierda para completarlo en otro momento. Los pedidos postergados se deben poder acceder y completar en cualquier momento. El reparto no debe finalizar hasta que todos los pedidos incluyendo los postergados se completen.

4.5.2. Requerimientos no funcionales

A continuación se presentan los requerimientos no funcionales relevados.

RNF1 - Aplicación móvil en Java

La aplicación móvil debe ser desarrollada en Java nativo de Android.

RNF2 - Back-end en Ruby On Rails

El back-end y la aplicación web deben ser desarrolladas en Ruby on Rails.

RNF3 - Funcionamiento sin conexión

Al realizar un reparto, la aplicación móvil debe ser independiente de la conectividad, no debiendo trancarse si no se cuenta con conexión a Internet, sino que siempre debe permitir a un repartidor completar un reparto en progreso. No aplica para obtener un nuevo reparto para comenzar.

RNF4 - Procesamiento de archivo de pedidos

El sistema debe ser capaz de permitir al administrador crear un reparto a partir de un archivo de pedidos con más de 130 pedidos sin que eso ocasione un *timeout* en la interfaz debido al tiempo que toma procesarlo.

RNF5 - Sincronización de los pedidos de un reparto a medida que se completan

Para permitir al administrador responder por consultas de un pedido de un reparto en transcurso, el sistema debe permitir al administrador visualizar el progreso de un reparto aunque el mismo no se haya finalizado. Los pedidos se deben sincronizar apenas luego de completados y no esperar a finalizar el reparto, suponiendo que se cuenta con conexión a Internet.

RNF6 - No se deben perder datos si el repartidor completa un pedido sin conexión a Internet

Si el repartidor no cuenta con conexión a Internet al momento de marcar un pedido como completado, el administrador deberá igualmente poder visualizar esos datos en su aplicación luego de que el repartidor recupere conectividad.

RNF7 - No se deben perder datos si el servidor está en mantenimiento o caído cuando el repartidor completa un pedido

Si el servidor se puso en mantenimiento o dejó de funcionar, el administrador deberá igualmente poder visualizar los datos de los pedidos completados en ese intervalo

luego de que el servidor vuelva a estar disponible y el repartidor continúe con el reparto.

RNF8 - No se deben perder datos si se apaga el móvil durante el reparto, o si se cierra la aplicación

Si el repartidor cierra la aplicación o apaga el móvil durante un reparto en curso antes de que algunos pedidos se sincronicen, el administrador deberá igualmente poder visualizar los datos de esos pedidos cuando el repartidor continúe con el reparto.

RNF9 - Memorización de las correcciones de direcciones

Al crear un nuevo reparto a partir de un archivo de pedidos en Excel, el sistema debe buscar o crear los destinatarios a partir de los datos del archivo. Las direcciones deben ser auto geolocalizadas, o creadas en modo pendiente notificando al administrador que debe geolocalizarlas manualmente. Si la dirección de destinatario ya había sido geolocalizada previamente, el pedido debe utilizar esas coordenadas, evitando que el administrador la vuelva a corregir.

RNF10 - Enrutamiento de pedidos

El sistema debe enrutar los pedidos de un reparto de forma de obtener el camino más rápido.

RNF11 - Reintento de enrutamiento de pedidos

El sistema debe reintentar automáticamente enviar para enrutamiento automático si pasaron más de 2 horas sin respuesta de Routeme, porque puede significar que se ha perdido el *webhook* con la respuesta.

RNF12 - Configuración de enrutamiento de pedidos

El enrutamiento automático de pedidos debe poder desactivarse para todos los futuros repartos desde una variable de entorno.

RNF13 - Memorización de enrutamientos de pedidos

El sistema debe recordar el enrutamiento de un reparto para no tener que volver a calcularlo en un reparto con los mismos destinatarios en el futuro, disminuyendo los costos del servicio de enrutamiento.

RNF14 - Sincronización con sistema externo de pedidos

El sistema debe notificar a otro sistema externo cuando un pedido es rebotado o entregado.

RNF15 - Interoperabilidad

El mecanismo para que las aplicaciones intercambien información será a través de APIs (*Application Programming Interface*) REST (*Representational State Transfer*).

RNF16 - Encriptación de los datos

Los datos sensibles como contraseñas de tanto los administradores como los repartidores deben ser encriptadas.

RNF17 - Acceso al sistema

El sistema debe contar con identificación, autenticación y control de acceso de usuarios, registrando la última vez que inició sesión, la IP desde la que lo hizo y la cantidad total de veces que inició sesión, para tanto repartidores como administradores.

RNF18 - Eficiencia en el uso de recursos

Se deben poder realizar al menos 20 repartos de 130 pedidos cada uno en un mes utilizando un plan de datos menor o igual a 4GB por mes.

RNF19 - Usabilidad de la aplicación móvil

La aplicación móvil debe poder ser utilizada por los repartidores en la calle por lo que debe ser intuitiva y rápida, debiendo respetar las heurísticas de Nielsen.

RNF20 - Escalabilidad

El sistema debe permitir ser escalado con el menor esfuerzo posible, tanto horizontal como verticalmente. El sistema debe permitir este tipo de escalamiento sin necesitar modificar el código.

RNF21 - Configuración

El sistema debe permitir que el mismo código sea desplegado a los diferentes ambientes sin necesidad de modificarse, por lo que es necesario extraer la configuración a variables de entorno.

RNF22 - Disponibilidad de la aplicación web

El sistema en producción debe estar disponible durante el horario laboral (de 9am a 6pm), no pudiendo sufrir caídas o tener defectos críticos por más de media hora en ese lapso que impidan al administrar crear o visualizar un reparto o al repartidor realizarlo.

RNF23 - Monitoreabilidad

El sistema debe permitir identificar defectos críticos en la aplicación web y móvil, alertando a los desarrolladores en el momento que sucedan. Se considera defecto crítico en la aplicación web a por ejemplo una respuesta con código 500, y en la aplicación móvil a un cierre inesperado.

5. Arquitectura y desarrollo

Este capítulo presenta las decisiones tomadas sobre la arquitectura de la solución realizada. El foco principal de la misma se basó en los requerimientos no funcionales relevados.

5.1. Desafíos de la arquitectura

A continuación se describen los desafíos enfrentados que junto con los requerimientos no funcionales relevados definieron la arquitectura.

Conectividad del repartidor

Los repartidores pueden llegar a sufrir pérdidas de conexión a internet en algunos tramos del reparto, pudiendo afectar la usabilidad o funcionalidades como alertas, sincronización, seguimiento, etc.

Plan de datos del repartidor

Los repartidores tienen un plan de datos de 4GB mensuales para transmitir todos los datos necesarios durante el reparto, incluyendo fotos, ubicaciones, etc.

Histórico de los pedidos

El administrador debe poder visualizar todos los datos de entrega de un pedido, incluyendo la marca temporal exacta en la que se realizó y no en la que llegó al servidor, pudiendo consultar la de pedidos de repartos pasados.

Seguimiento en tiempo real

El administrador debe poder visualizar el recorrido realizado por el repartidor en tiempo real por lo que la arquitectura debe definir cómo se llevará a cabo la sincronización necesaria para el seguimiento.

Geolocalización de direcciones

El archivo Excel de pedidos que se utiliza para armar el reparto contiene las direcciones de los destinatarios sin validar, las que deben ser geolocalizadas para poder enrutar los repartos. Esto implicó, además de una investigación para definir qué proveedor proveyó las mejores geolocalizaciones, definir el flujo y la arquitectura para que esto suceda.

Enrutamiento de pedidos

Los pedidos de un reparto deben ser ruteados para lograr el camino más rápido. Esto implica un procesamiento que puede tomar varias horas. Ya sea realizando un

algoritmo propio o contratando un servicio, la arquitectura debió definir responsabilidades e interacciones para que esto suceda.

Los siguientes desafíos también fueron considerados y resueltos rápidamente por los motivos descritos a continuación.

Batería limitada

La solución que planteó el equipo fue muy demandante en cuanto a uso de batería. Antes de comenzar a desarrollar el equipo sugirió a Formisur colocar cargadores de teléfono en las camionetas y la sugerencia fue aceptada.

Necesidad de almacenamiento para grandes cantidades de fotos

La solución planteada implicaba almacenar las firmas de los destinatarios digitalmente. Martín García (Encargado en Sistemas) explicó que el almacenamiento no sería una restricción para el proyecto. La empresa contó con créditos disponibles en servicios de Amazon para almacenar. De todos modos, el equipo propuso alternativas como configurar reglas de expiración de los datos en los repositorios por si esto llega a ser un problema.

5.2. Descripción de la arquitectura

A continuación, se presenta un diagrama de alto nivel de los principales elementos de la arquitectura del sistema construido, los cuales fueron diseñados a partir de los requerimientos funcionales y no funcionales relevados.

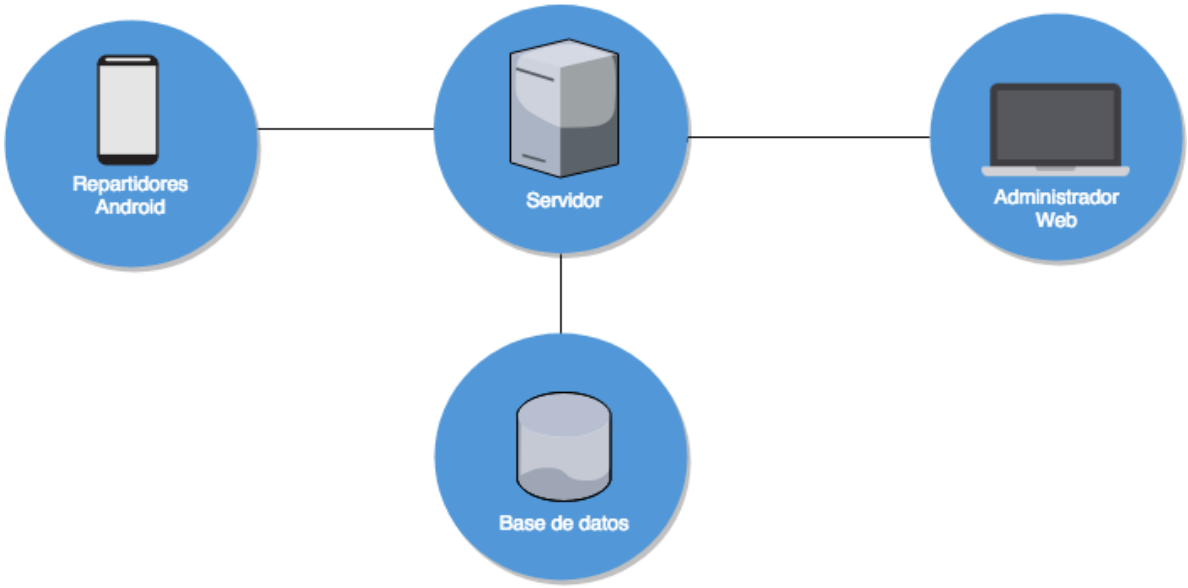


Ilustración 29 - Arquitectura alto nivel

- **Repartidores-Android:** es la aplicación móvil Android a la que acceden los repartidores para poder utilizar Geolocalizador.
- **Servidor y base de datos:** es donde se almacenan los servicios expuestos y la información generada para consumo de las diferentes aplicaciones.
- **Administradores-Web:** es la aplicación web a la que acceden los administradores del sistema Geolocalizador.

La arquitectura está dividida en back-end y front-end. El back-end se compone por el servidor y la base de datos, mientras que el front-end abarca el resto de los elementos mencionados anteriormente. A continuación se presenta un diagrama de nivel más bajo donde se puede apreciar cómo interactúan el front-end y el back-end.

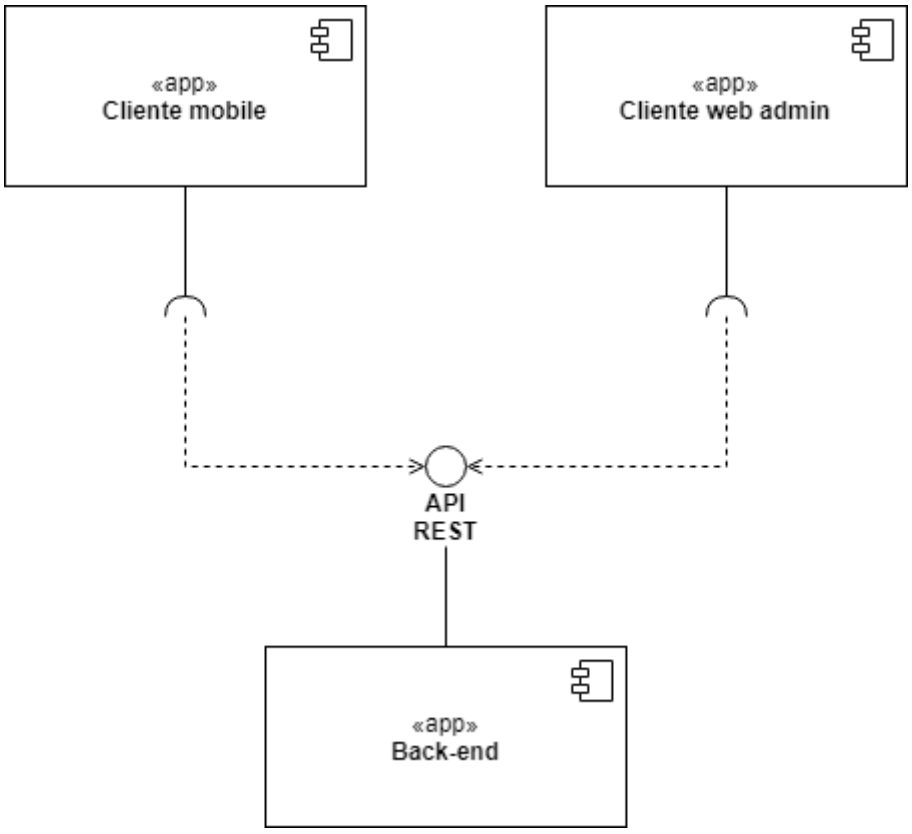


Ilustración 30 - Componentes y conectores back-end

La decisión de separar el back-end del front-end móvil estuvo dada por el requerimiento no funcional que definió implementar una aplicación móvil. Las conexiones entre las mismas se dan a través de una API REST para favorecer la portabilidad y eficiencia como se mencionará más adelante.

El servidor que expone las APIs REST es único para tanto la aplicación móvil como para el cliente web. Esto fue principalmente porque los datos a los que acceden son compartidos. Tanto el repartidor como el administrador consultan los datos de los repartos. Si bien en un futuro separar los proveedores de servicios que consume cada

aplicación otorgaría aislamiento y autonomía para escalarlas por separado, para un proyecto cuyo objetivo está en la puesta rápida a producción desarrollar las APIs en el mismo servidor agilizó el desarrollo, sin imposibilitar la opción de en un futuro separarlas.

5.3. Atributos de calidad

A continuación se describen los atributos de calidad que se desprendieron de los requerimientos no funcionales y que fueron considerados a la hora de diseñar la arquitectura del sistema, junto con las estrategias utilizadas para asegurar cumplir con los objetivos propuestos para cada uno. Esto no significa que los atributos que no se encuentren en la lista no fueron considerados, si no que se cumplieron de otra forma y que no tuvieron un impacto directo en el diseño de la arquitectura.

Los atributos de calidad que se desprendieron de los requerimientos no funcionales fueron eficiencia, disponibilidad, monitoreabilidad, modificabilidad, testeabilidad, seguridad y escalabilidad.

5.3.1. Eficiencia

Como se mencionó anteriormente, la eficiencia especialmente en cuanto a uso de recursos de red del lado de la aplicación móvil fue crítica para construir una solución que se pudo poner en producción. El equipo trabajó para cumplir con este atributo en varios aspectos diferentes del sistema. Desde el protocolo de comunicación, hasta la compresión de imágenes, este atributo de calidad dictó completamente la arquitectura.

A continuación se describen las técnicas utilizadas por el equipo para lograr los objetivos de eficiencia en cada aplicación.

Eficiencia de la aplicación móvil

El equipo tuvo como un punto de evaluación final que la aplicación construida se pueda utilizar diariamente en producción con el plan de datos actual de la empresa. Desde el 15/01/2019 el producto está en producción y no ha sido necesario aumentar el plan de datos.

Para favorecer la eficiencia en cuanto a uso de recursos de red, en primer lugar se decidió desarrollar una API REST. En el caso de la aplicación móvil, la información que se envía desde el servidor al front-end son datos planos en formato JSON. Esto significó un protocolo de comunicación liviano que requiere menos tiempo de transferencia y consumo de red, frente a otros formatos más pesados.

El equipo también procuró comprimir las fotos a 15KB. Esto se debió a un cálculo entre plan de datos disponible por repartidor, repartos por mes por repartidor y pedidos

por reparto. Se buscó un balance entre la definición de las imágenes a mostrar y el uso de red.

Por otro lado, para la funcionalidad del seguimiento del recorrido se aplicó la técnica de eficiencia llamada “*Control frequency of sampling*" [5] y se consideró reportar la ubicación del repartidor cada no más de dos minutos. Esta frecuencia se determinó considerando que es suficiente para detectar desvíos o paradas personales en el recorrido de un repartidor. Si bien reportar en intervalos más chicos permitiría una mayor precisión en el recorrido visualizado, es innecesario para el objetivo de identificar ineficiencias de los repartidores.

Eficiencia de la aplicación web y API

Para la aplicación del administrador también se tuvieron en cuenta aspectos de la eficiencia, aunque el repartidor siempre tiene acceso a *Wi-Fi*, con el fin de disminuir el tiempo de respuesta del servidor y lograr una interfaz más rápida. Del lado de la API la mayor parte de las tácticas de eficiencia utilizadas se basan en incrementar la eficiencia computacional e introducir redundancia.

El equipo se propuso que del lado del servidor no se superen los dos segundos de tiempo de respuesta en situaciones de carga de producción. Desde el 15/02/2019 el producto está en producción y no se han reportado tiempos de respuesta mayores a los dos segundos.

Para evaluar y monitorear los tiempos de respuesta el equipo instaló el add-on Logentries [6] en Heroku, herramienta de monitoreo y manejo de *logs*, tanto en ambiente de desarrollo como en producción, y se configuró para que notifique por correo electrónico a los desarrolladores si ocurre algún tiempo de respuesta mayor al establecido.

A continuación se muestra un ejemplo de una alerta en el ambiente de desarrollo debido a tiempos de respuesta altos.

Logentries Alerts Trash -...@gmail.com January 15, 2019 at 11:00 AM

[formisur] ALERT High Response Time: Heroku/Syslog drain [geolocalizador-develop]

To: Leticia Esperon, leticia@rootstrap.com, Martín García,

Reply-To: formisur@herokumanager.com

High Response Time: Heroku/Syslog drain

2019-01-15 14:00:36.279
328 <158>1 2019-01-15T14:00:36.018579+00:00 heroku router - - at=info method=PUT path="/api/v1/orders/1490" host=geolocalizador-develop.herokuapp.com request_id=1f6d7639-af7b-4320-aaa0-7c7fc3722180 fwd="152.156.222.80" dyno=web.1 connect=0ms service=5172ms status=200 bytes=803 protocol=https

Go to the log: https://api.heroku.com/myapps/app98772091@heroku.com/addons/logentries

You are receiving this email because your Logentries alarm "High Response Time" has been triggered.

In context:
2019-01-15 13:51:20.307 303 <190>1 2019-01-15T13:51:20.210278+00:00 app web.1 - - D, [2019-01-15T13:51:20.210147 #4] DEBUG -- : [1a56875e-1abd-4b48-a9fb-7ed0acf0d17f] [1m[35m (6.7ms)[0m

Ilustración 31 - Ejemplo tiempos de respuesta

Un aspecto que fue necesario tener en cuenta para asegurar esos tiempos de respuesta fue el tamaño de las imágenes a mostrar en la aplicación del administrador. Las imágenes de confirmación de los pedidos (firmas o fotos de las casas) a mostrar en los listados son *thumbnails* y no las versiones originales. Las versiones originales solo se muestran cuando se piden los detalles de confirmación de un pedido en particular y están limitadas a 500 x 500 px.

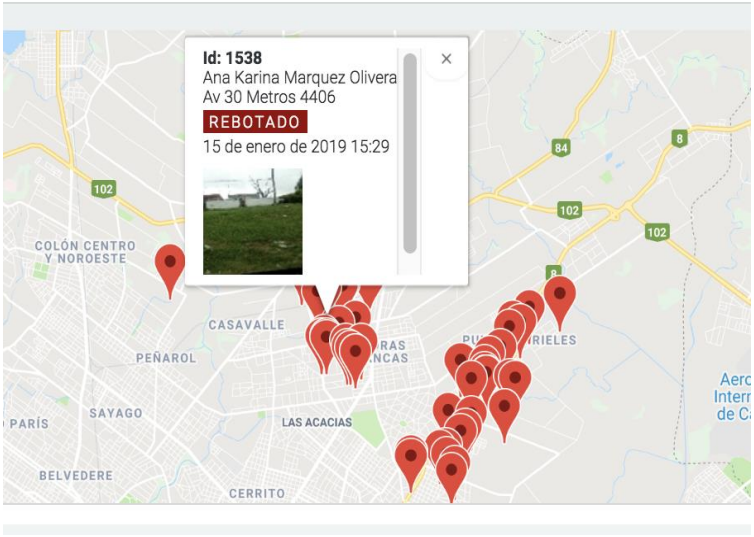


Ilustración 32 - Ejemplo foto *thumbnail*

Por otro lado, con el fin de aumentar la eficiencia computacional, se tuvieron en cuenta varios aspectos del código y se instalaron herramientas para identificar ineficiencias como problemas N + 1 [8].

5.3.2. Disponibilidad

La disponibilidad del sistema es crucial para que la empresa pueda funcionar correctamente. La aplicación web debe estar disponible sobre todo para poder programar un nuevo reparto y para poder consultar los datos de un reparto en curso, lo que sucede normalmente entre las 7 am y las 7 pm. Por otro lado, la disponibilidad de la aplicación móvil es aún más crítica, ya que si deja de funcionar media hora durante un reparto, puede llegar a implicar la pérdida de los datos de entrega de hasta 10 pedidos, o puede directamente impedir que el repartidor realice su trabajo. Es por eso que la arquitectura del sistema fue dictada por este atributo de calidad, y muchas decisiones como la plataforma de *hosting* o servicios de monitoreabilidad fueron elegidos cuidadosamente.

Disponibilidad de la aplicación web y API

La aplicación web y la API se encuentran en desplegadas en un nodo servidor de *Heroku*, denominado por este proveedor como *dyno*.

Además de permitir escalar horizontalmente con unos simples clicks, *Heroku* ofrece como parte de su *Customer Promise* un *SLA* de casi 100% de *uptime* [9]. El porcentaje de disponibilidad en los últimos 60 días fue consultado el día 27/01/2019 y mostraba un 99.999754% en la región de Estados Unidos [10], que es donde se aloja la API de Geolocalizador. *Heroku* hace uso de varias tácticas de disponibilidad para garantizar su SLA como excepciones, redundancia, *rollback* y transacciones [11].

De todos modos, el SLA de Geolocalizador es menor al garantizado por Heroku, ya que además depende del código desarrollado y de los servicios externos utilizados.

Es por ello que el equipo eligió cuidadosamente los servicios externos a utilizar e intentó asegurarse de que las funcionalidades principales de Geolocalizador no dependan de la disponibilidad de ellos. Por ejemplo, no existe ningún servicio externo crítico para las funcionalidades principales durante la entrega de un reparto, con excepción de Amazon S3 para el almacenamiento de las fotos. Sin embargo, este servicio de Amazon cuenta con un SLA de al menos 98.0% [12] por lo que el equipo no consideró que esto fuera un riesgo.

Uno de los servicios externos más importantes utilizados es Routeme, servicio en la nube que enruta los pedidos de un reparto. Para casos excepcionales donde Routeme no se encuentre funcionando, el equipo definió hacer opcional el enrutamiento de pedidos, permitiendo al administrador seleccionar si desea enviar el listado de pedidos a este servicio o no. Si por alguna razón Routeme se encuentra caído, el administrador podría repetir el reparto definiendo su propia ruta. Por otro lado, Geolocalizador guarda un histórico de los recorridos calculados en un modelo llamado Rutas Base, para no tener que enviar a Routeme un reparto que ya fue ruteado recientemente,

disminuyendo el tiempo, el costo del servicio y favoreciendo la disponibilidad. Sobre la decisión técnica de utilizar Routeme como servicio para el enrutamiento de pedidos, se describe en el ANEXO 16 – “Análisis de servicios de enrutamiento”.

Para la realización de tareas para las cuales el usuario no debe esperar su respuesta, como el enrutamiento de los repartos o el envío de emails de recuperación de contraseña, existe un *dyno*, o eventualmente más de uno, que se encarga de realizar estos trabajos, llamado *worker*. Estos *dynos* son servidores iguales que los servidores web, solo que en lugar de atender pedidos HTTP toman sus tareas de una cola de trabajos almacenada en Redis que es una base de datos en memoria, como se muestra a continuación.

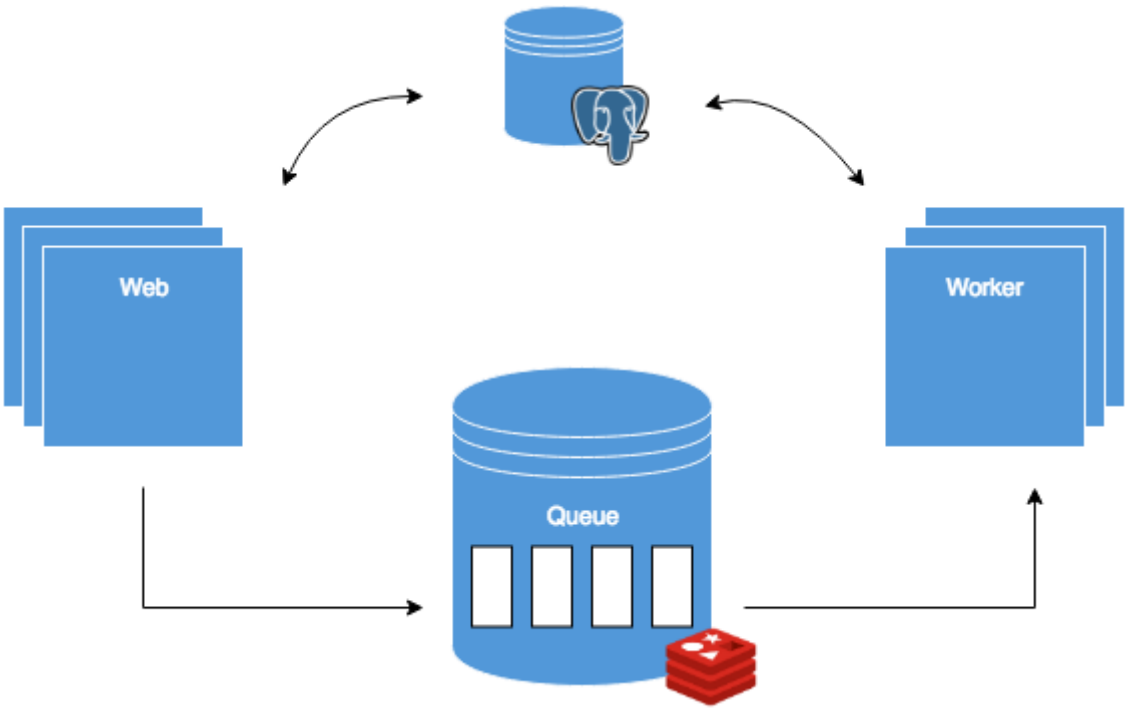


Ilustración 33 - Arquitectura de dynos web y workers

De esta manera, estas tareas se realizan de manera aislada, en servidores que pueden ser escalados independientemente. La decisión de utilizar Redis para alojar la cola de trabajos se debió a que permite una escritura más rápida que alternativas como utilizar la misma base de datos que se utiliza para guardar el resto de los datos, permitiendo que el servidor web encole el trabajo correspondiente más rápidamente y responda al usuario antes.

A continuación se muestra un ejemplo de una interacción entre el *browser* del administrador, el proceso web de la API, el proceso *worker* de la API y el servicio externo, en este caso para crear un reparto.

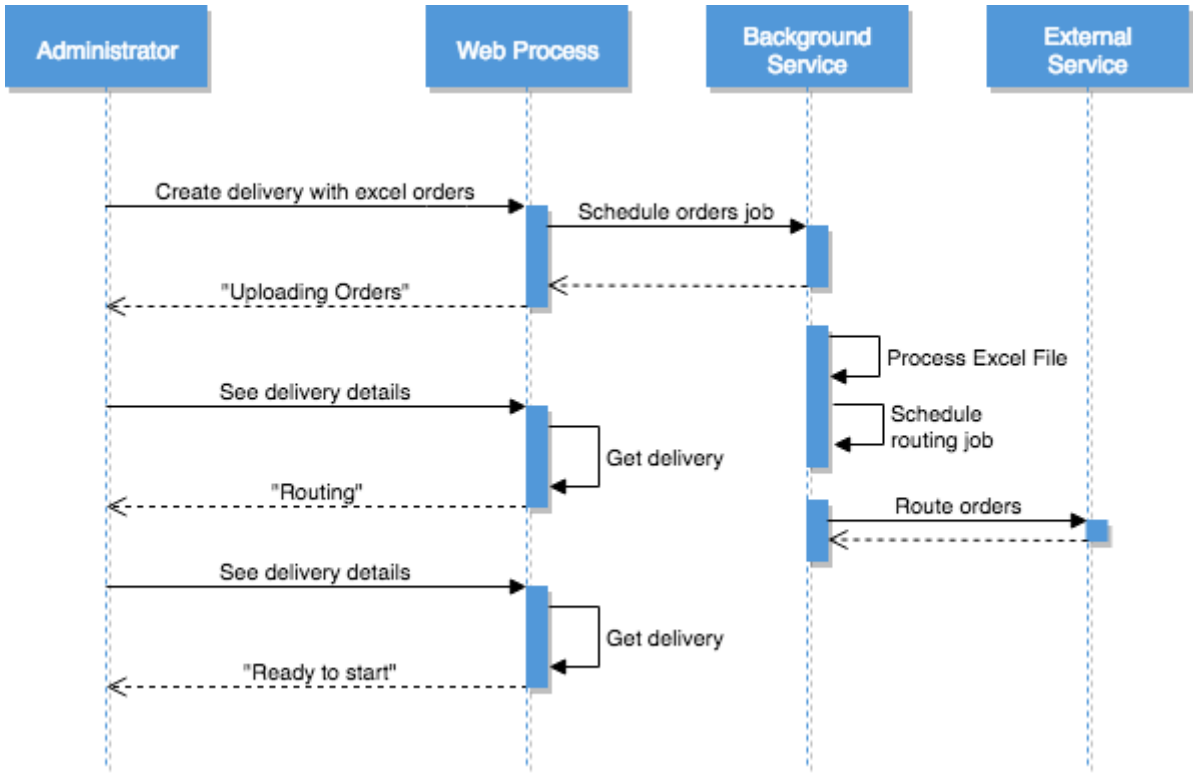


Ilustración 34 - Procesamiento asíncrono para consultas con servicios externos

Como se puede observar, el administrador obtiene una respuesta antes de que el trabajo de procesar el archivo de pedidos se realice, en lugar de tener que esperar hasta 20 segundos para que se cargue la página, lo cual permite al proceso web liberarse antes y continuar atendiendo otros pedidos.

El administrador visualiza lo descrito anteriormente de la forma a continuación:

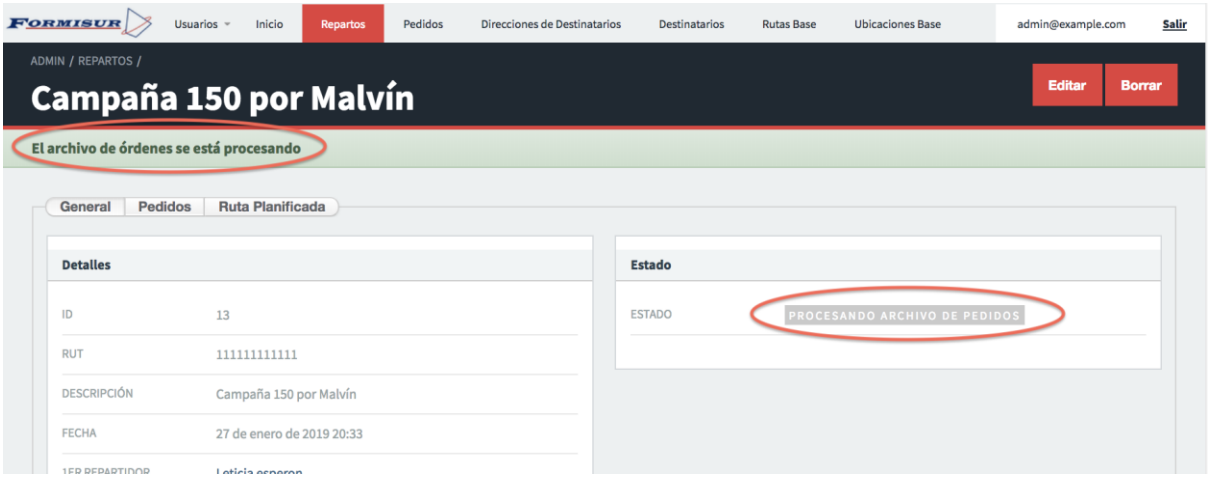


Ilustración 35 - Visualización en la interfaz al procesar archivo de pedidos

Las tareas que se realizan en forma asíncrona en el servidor son:

1. Procesamiento del archivo Excel de pedidos
2. Enrutamiento de pedidos de un reparto
3. Notificación a sistema externo de pedidos cuando un pedidos es completado
4. Envío de correo electrónico para restablecer contraseña
5. Subida de la foto a Amazon S3
6. Eventualmente comunicación con la aplicación móvil iniciada desde el servidor (Push Notifications)

Las tareas que también corren en los dynos *workers* pero que son iniciados por un *scheduler* son:

1. Expiración de rutas base
2. Recálculo de rutas de repartos cuya respuesta de Routeme no ha llegado hace más de tantas horas
3. Expiración de las matrices de costos (para la primera versión del enrutamiento de repartos)

Todos los trabajos se reintentan hasta veinte veces automáticamente si ocurre una excepción durante su ejecución, con excepción de aquellos trabajos que ya ocurren periódicamente. Ejemplos de errores en la ejecución de un trabajo podrían ser respuestas inesperadas de un servicio externo.

Disponibilidad de la aplicación móvil

La disponibilidad de la aplicación móvil fue uno de los mayores desafíos. La funcionalidad de entregar los pedidos durante un reparto debe poder hacerse sin conexión a Internet.

La arquitectura debe cumplir con los siguientes requerimientos no funcionales:

1. No se deben perder datos si el repartidor completa un pedido sin conexión a internet.
2. No se deben perder datos si el servidor está en mantenimiento o caído durante el momento de la entrega.
3. No se deben perder datos si se apaga el móvil durante el reparto, o si se cierra la aplicación.

Es por ello que el equipo definió una arquitectura inicial donde los pedidos completados se almacenan en una cola en una base de datos SQLite en al móvil, para luego enviarse al servidor de forma asíncrona al reparto. La cola se intenta sincronizar con el servidor periódicamente. Si por alguna razón el servidor dio una respuesta inesperada, como un “*500 Internal Server Error*”, los pedidos se mantienen en la cola hasta que el móvil se asegure de que el servidor recibió correctamente los datos.

A continuación se muestra un diagrama que demuestra el proceso:

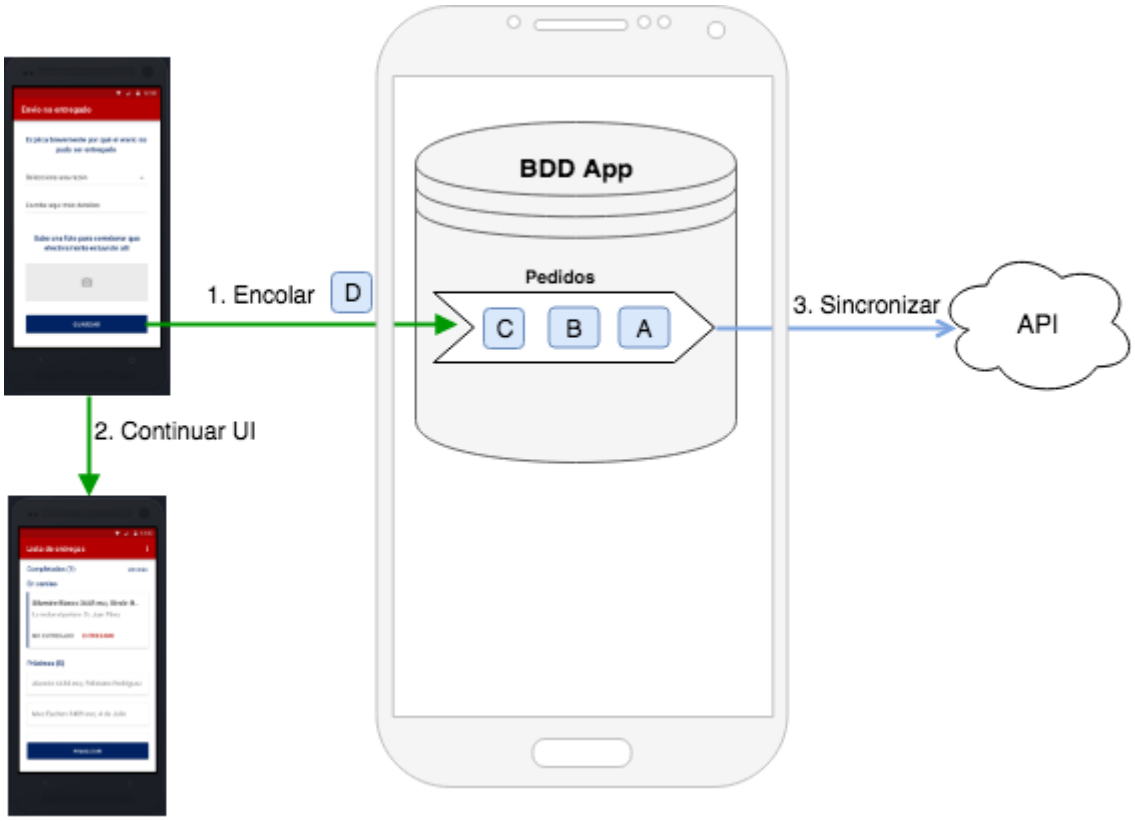


Ilustración 36 - Arquitectura de la cola de pedidos en la aplicación móvil

Esta arquitectura asegura entonces que, en síntesis, no se pierden datos a menos que se pierda el móvil antes de que se logren sincronizar, cumpliendo con los requerimientos no funcionales definidos anteriormente. Además, se favorece la usabilidad de la aplicación ya que el usuario percibe una respuesta inmediata en lugar de esperar por la respuesta del servidor.

Es importante considerar que el móvil sí precisa conexión a Internet para obtener los datos del próximo reparto a realizar, pero una vez comenzado puede completarlo sin ella. Cuando recupera conectividad los datos se sincronizan con el servidor. Si el repartidor contó con conectividad durante todo el reparto, el administrador puede monitor su recorrido en tiempo real y ver el estado de los pedidos a medida que se van completando.

La misma arquitectura se quiso utilizar para la funcionalidad de mostrar al administrador el mapa con el recorrido que ha hecho el repartidor. Se quiso aplicar para poder enviar las posiciones del repartidor, incluso si el repartidor pierde conectividad. El problema de esto es que si no cuenta con conectividad no se puede obtener la ubicación donde se encuentra en ese momento, perdiendo el sentido. Es por esto que la funcionalidad del monitoreo del recorrido requiere de conectividad al momento de enviar la posición. Se lleva a cabo con un proceso que corre en segundo

plano cada dos minutos, que intenta obtener y enviar la posición actual del repartidor al servidor.

Finalmente cabe destacar que el equipo evaluó el realizar pruebas de carga. Dada que la carga en el sistema se distribuye durante todo el horario laboral, no habiendo picos, y que para la primer versión del programa será utilizado por a lo sumo quince dispositivos simultáneamente, no se consideró necesario.

5.3.3. Monitoreabilidad

Como se mencionó anteriormente, una caída del sistema en horario laboral es crítico. Para detectar caídas y defectos, el equipo decidió instalar herramientas de monitoreo de tanto el lado de la aplicación móvil como de la API y la aplicación web.

El equipo se propuso como objetivo de la monitoreabilidad, durante los repartos enterarse de las caídas del sistema antes de que los repartidores las reporten al final del reparto.

Monitoreabilidad de la aplicación web y API

En el caso de la API y la aplicación web, se instalaron en Heroku los *add-ons* Logentries y Rollbar.

Logentries permite retener, centralizar, y filtrar el histórico de los logs del servidor. De esta manera se puede rastrear una excepción para comprender por qué sucedió.

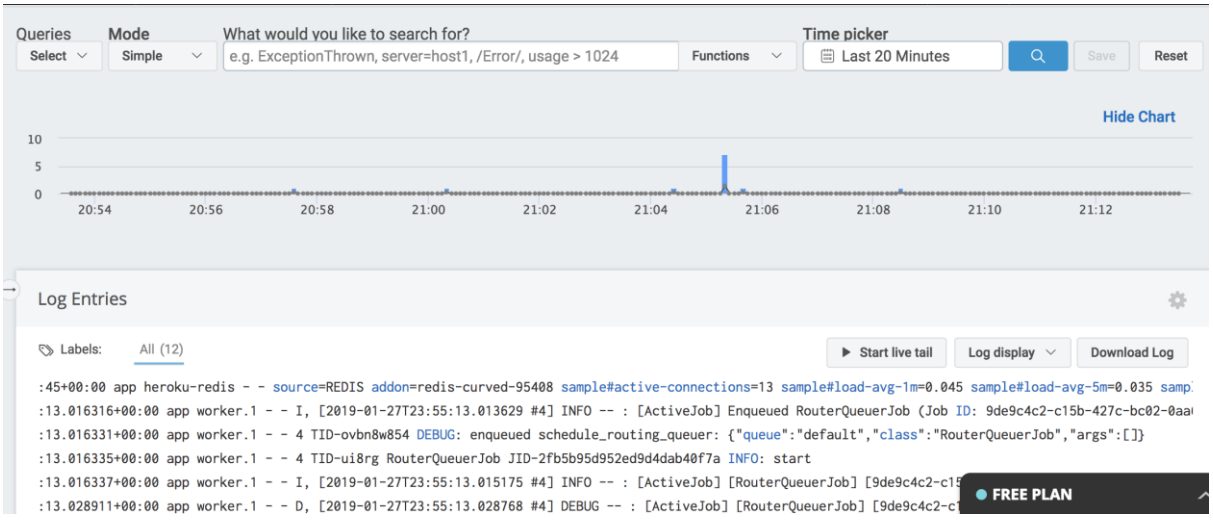


Ilustración 37 - Ejemplo de visualización de *logs* en Logentries

Por otro lado, Rollbar es un *add-on* que permite recolectar errores, monitorearlos y configurar reportes semanales o alertas por email cuando ocurre una excepción.

Durante los repartos el equipo configuró Rollbar para enviar emails a los desarrolladores si ocurría una excepción, con el fin de poder identificar y arreglar el defecto cuanto antes para evitar trancar el reparto. Afortunadamente, no hubo ningún reporte crítico del lado del servidor durante ningún reparto.

A continuación se muestra un ejemplo de cómo se visualizaron los errores en Rollbar y las notificaciones por correo.

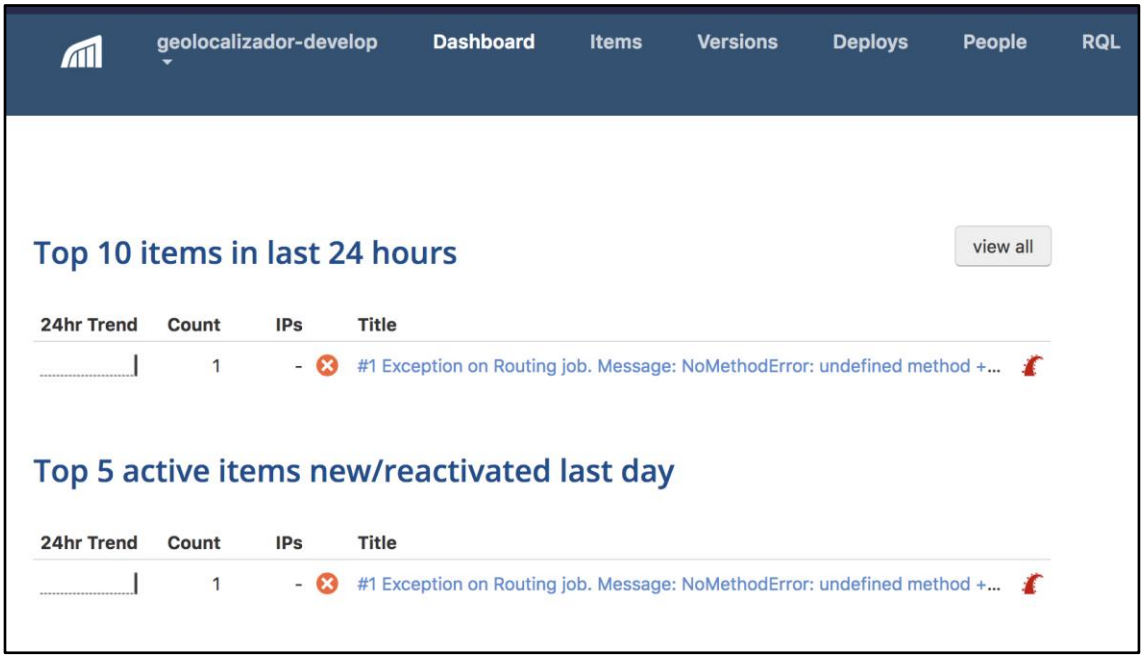


Ilustración 38 - Ejemplo de visualización de errores en Rollbar

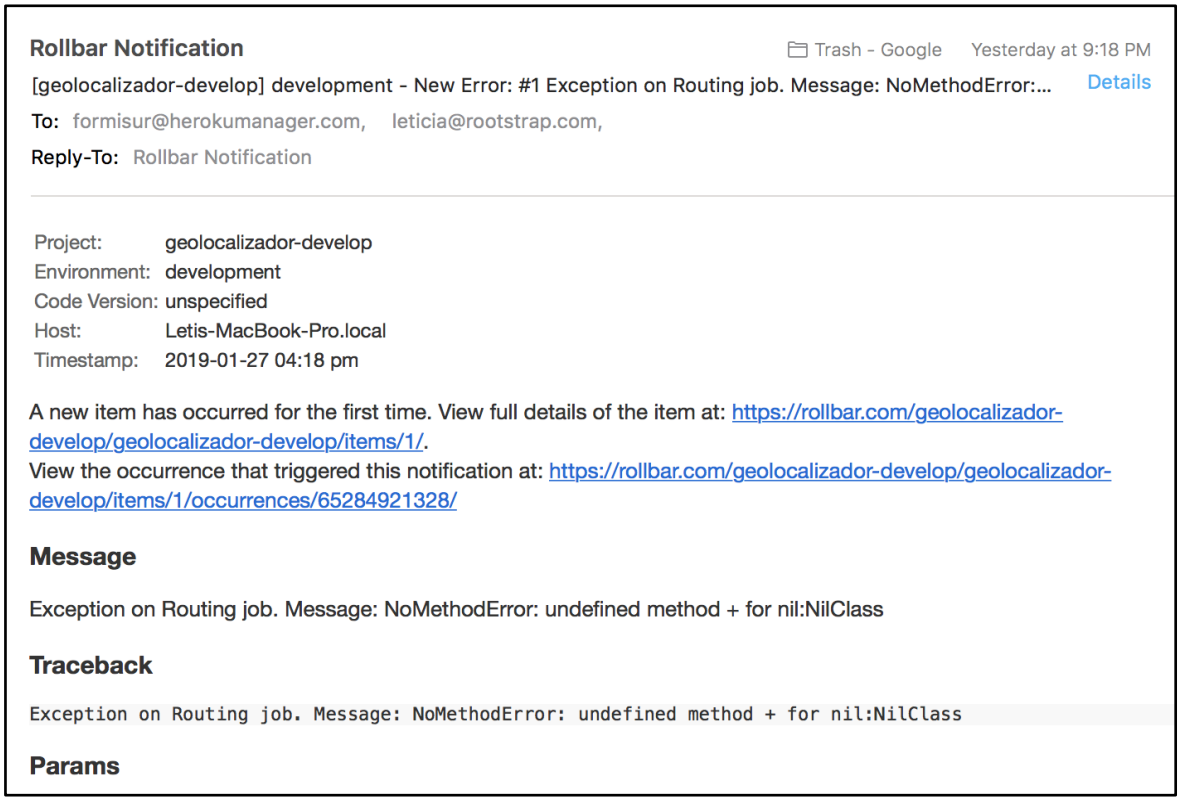


Ilustración 39 - Ejemplo de notificación por correo de errores con Rollbar

Monitoreabilidad de la aplicación móvil

Para poder identificar incidentes del lado de la aplicación móvil y tener la información necesaria para entender qué sucedió, como el *stacktrace* y estado del móvil, se instaló Fabric en la aplicación antes del primer reparto de prueba.

Fabric es un conjunto de herramientas integrables a aplicaciones móviles, entre ellas *Crashlytics* que informa de fallos en la aplicación, proporcionando un detalle de los errores que ocurrieron en la aplicación.

A continuación se muestra un ejemplo de cómo se visualizaron los errores en el tablero de Fabric y las notificaciones por correo.

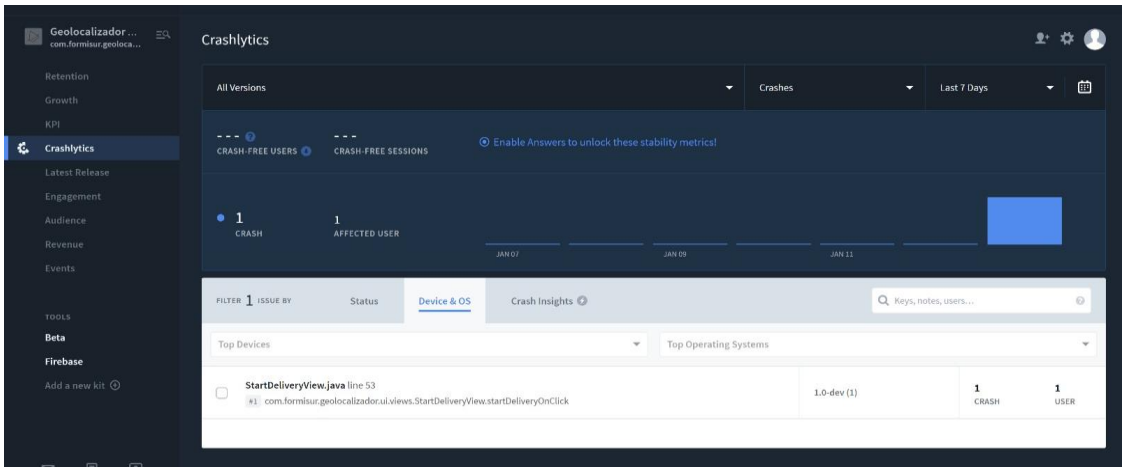


Ilustración 40 - Ejemplo de recolección de incidentes con Fabric

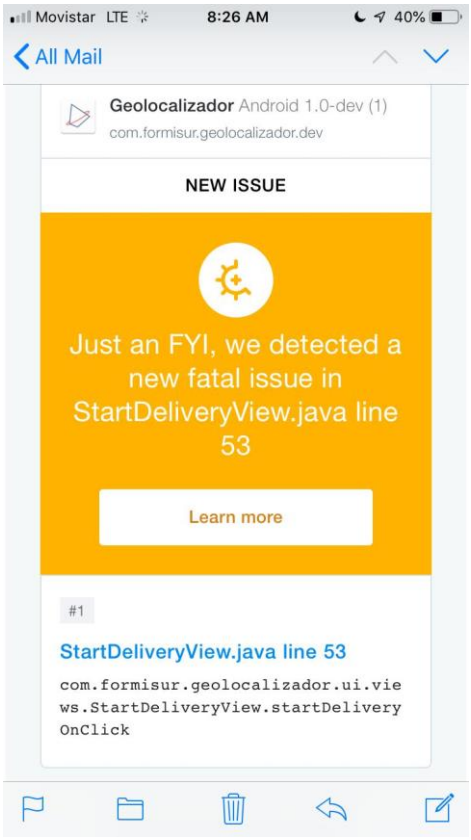


Ilustración 41 - Ejemplo de notificación de errores por correo con Fabric

Fabric permitió descubrir un defecto importante en el primer reparto de prueba, debido a un problema de mal manejo de la memoria que fue corregido para el resto de los repartos.

Actualmente Fabric se sigue utilizando durante los repartos en producción. No se han reportado nuevos incidentes en los últimos quince días al día 20/02/2019.

5.3.4. Modificabilidad

La modificabilidad fue un atributo particularmente importante en Geolocalizador ya que al ser un MVP se priorizó en todo momento simplificar las funcionalidades con el objetivo de entregar valor rápidamente, por lo que hay varias que se van a querer modificar o agregar en un futuro cercano.

Para estructurar las clases y módulos de las aplicaciones a construir, se definieron utilizar patrones de arquitectura tanto en la aplicación móvil como en la API y aplicación web.

Para el back-end se utilizó el patrón MVC (*Model View Controller*) y en la aplicación móvil el patrón MVP (*Model View Presenter*). En ambos casos el patrón correspondiente favorece la modificabilidad debido a la separación de las responsabilidades de los componentes y a la generación de un nivel de abstracción que facilita la comprensión del código.

A continuación se muestra en un diagrama el funcionamiento del patrón MVC en el lado del back-end.

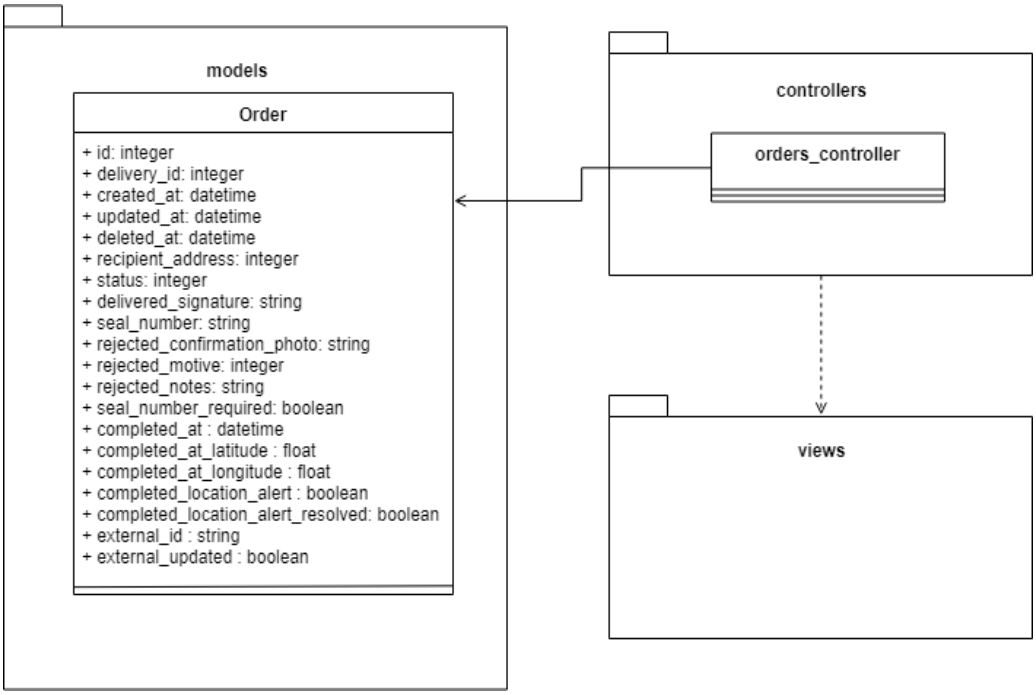


Ilustración 42 - Ejemplo MVC

El controlador actúa como mediador entre la vista y el modelo, interpretando y dando sentido a las instrucciones que realiza el usuario y actuando sobre el modelo. De esta manera se independiza al modelo de la vista y se logra que sea posible sustituir las vistas sin que impacten al mismo. Como ejemplo concreto en el caso del proyecto contamos con vistas en formato JSON para la API y vistas en formato HTML para la

web del administrador. El modelo es completamente independiente de ellas y estas solo agregan la lógica necesaria para presentar los datos.

Por otro lado, el MVP (*Model View Presente*r) es un patrón derivado del MVC (*Model View Controller*) que también permite separar la capa de presentación de la lógica de la misma, de tal forma que todo lo relacionado con cómo funciona la interfaz queda separado del cómo representarlo en pantalla. Idealmente el patrón MVP permitiría conseguir que una misma lógica pudiera tener vistas totalmente diferentes e intercambiables.

El *Presenter* se encarga de actuar de intermediario entre la vista y el modelo. Recupera los datos del modelo y se los devuelve a la vista. La vista es implementada por una *Activity*, *Fragment*, *View*, etc. El modelo es la puerta de enlace a la capa de dominio o de lógica de negocio.

En nuestra solución decidimos evitar que el modelo y la vista contengan una referencia al *Presenter* por lo que utilizamos la herramienta Otto Bus para publicar eventos e invocar métodos del *Presenter* desde el modelo y la vista sin generar una dependencia.

El siguiente diagrama ejemplifica un uso típico del patrón MVP aplicado. Este formato se repite para cada *Activity* y *Fragment*. La *Activity* hereda de *BaseActivity* y contiene una referencia a un *Presenter* que se inicializa con una *View* y un *Model*. En el caso de los *Fragments* es análogo, heredando de *BaseFragment*. Estas clases se encuentran en geolocalizador.ui.

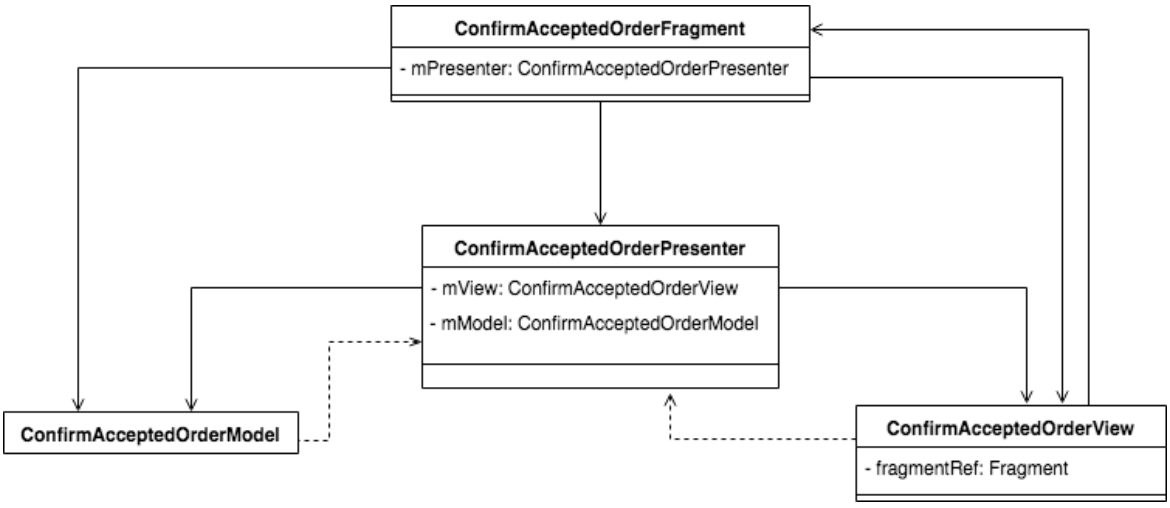


Ilustración 43 - Ejemplo MVP

Por otro lado, del lado del servidor las variables de configuración son fácilmente modificables desde el *Dashboard* de Heroku, respetando el principio número 3 - “Configuración” de “*12 Factor App*” [13]. Esto hace que no se requiera modificar el

código para cambiar una de estas configuraciones, lo que además permite tener diferentes configuraciones entre los diferentes ambientes. Ejemplos de estas variables son la ruta para acceder a la base de datos, o las rutas y claves de acceso a los servicios de Google y Routeme.

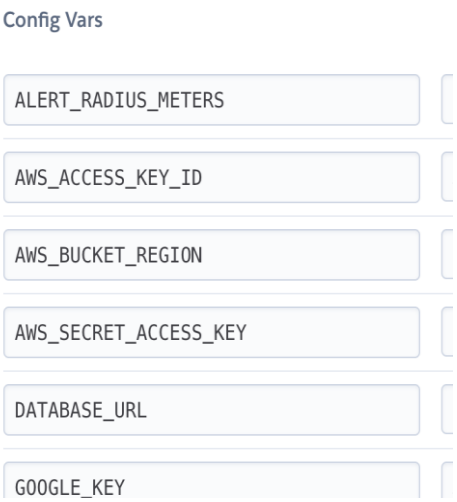


Ilustración 44 - Extracción de la configuración a través de variables de entorno

Para favorecer la modificabilidad en cuanto a los servicios externos utilizados, los servicios concretos se encuentran en un módulo con el nombre del servicio externo pero son encapsulados por un servicio genérico que permitiría cambiar la implementación sin repercutir en todos los lugares donde se llama.

Un ejemplo de esto fue el servicio que enruta los pedidos de un reparto. En la primera versión el equipo implementó un algoritmo propio llamado Intercalator. En la segunda versión, se decidió cambiar la implementación radicalmente y se integró con Routeme. Al integrar con Routeme, se creó otro módulo con lo relacionado a este, y lo único en el código que se tuvo que cambiar fue la llamada desde el servicio genérico.

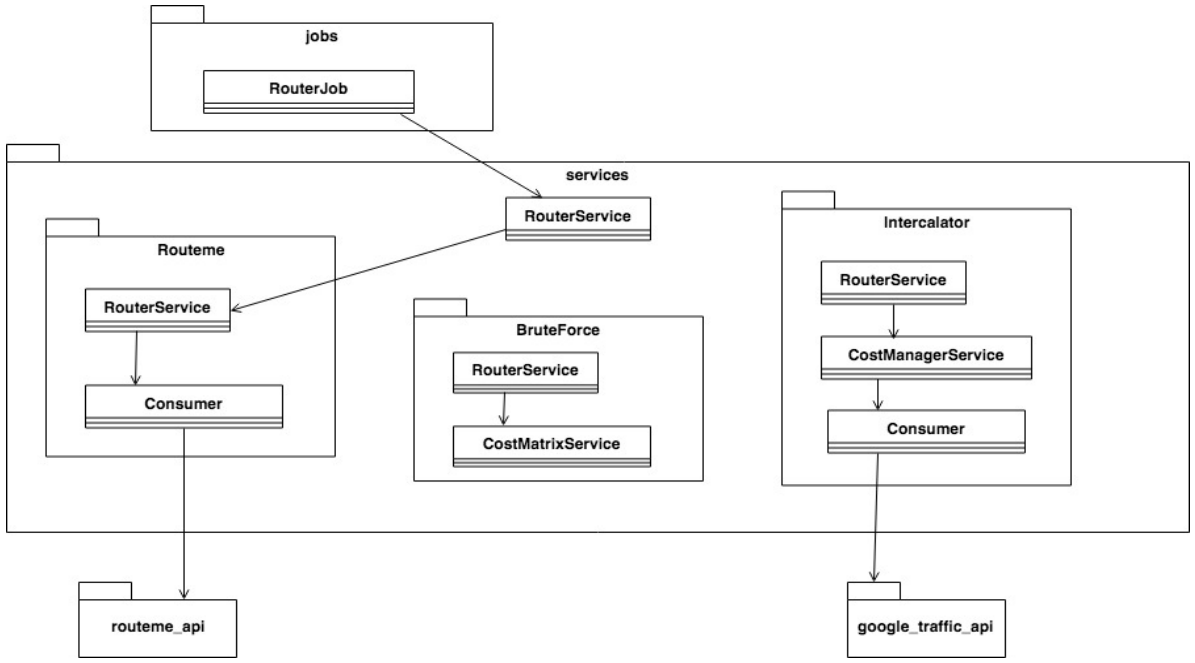


Ilustración 45 - Vista de descomposición de las implementaciones del enrutamiento

5.3.5. Testeabilidad

Para favorecer la mantenibilidad y modificabilidad del sistema, este debe ser altamente testeable. En cuanto a lo que la arquitectura respecta, el equipo decidió utilizar un Framework de testing llamado Rspec [14] del lado de la aplicación web y de la API, que permite realizar pruebas unitarias con una sintaxis sencilla y nemotécnica. Se instaló una gema llamada Webmock [15] para simular las llamadas HTTP que se realizan a servicios externos durante las pruebas, definiendo *mocks* de las mismas, de manera que las pruebas unitarias del código no dependan de estos servicios y puedan ser repetibles y rápidas.

5.3.6. Seguridad

La seguridad en el proyecto es importante ya que se manipulan datos sensibles como firmas, datos de destinatarios, ubicaciones y no se puede permitir que alguien más obtenga acceso a esos datos o los modifique sin autorización.

La comunicación con la API de Geolocalizador se realiza a través del protocolo HTTPS (*HyperText Transfer Protocol Secure*) [16], donde la información viaja encriptada previniendo los ataques *Man in the Middle* [17]. Heroku, la plataforma de hosting elegida, utiliza automáticamente este protocolo con los certificados correspondientes. En el caso de producción donde el cliente compró su propio dominio, se obtuvieron y configuraron los certificados SSL (*Secure Sockets Layer*) necesarios.

Se utilizaron las tácticas de seguridad autenticación y autorización de usuarios. Se decidió utilizar una gema muy popular llamada Devise que utiliza un manejo de sesión mediante *tokens*. Este sistema de autenticación se utilizó para administrar los *tokens* de tanto los repartidores como los administradores. También realiza el manejo de las *cookies* de sesión del lado del administrador.

Los *tokens* son generados por la API y son únicos, aleatorios y encriptados. Los llamados a la API que implican autenticación se envían firmados, es decir, se envía el *token* de la sesión que identifica al usuario y sin el mismo no es posible efectuar la acción. Los *tokens* expiran en cada llamada, retornando en los *headers* de la respuesta el *token* necesario para realizar la próxima llamada.

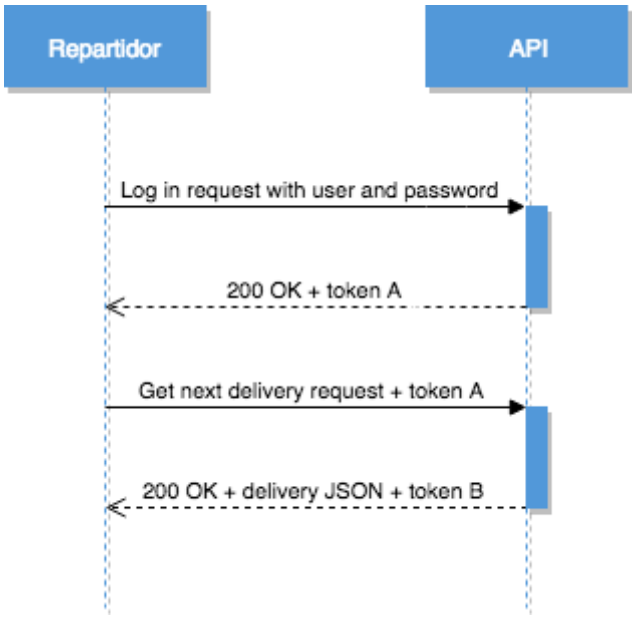


Ilustración 46 - Autenticación del repartidor a través de un *token*

El sistema permite a los usuarios mantener sesiones abiertas en diferentes dispositivos simultáneamente. El *token* de sesión del administrador expira automáticamente después de una semana sin usarse. El sistema guarda la cantidad de veces y la fecha de la última vez que cada repartidor o administrador inició sesión y la última IP desde la que lo hizo, para permitir al administrador consultarlo en caso de sospecha. En el caso de los repartidores, solo se les permite visualizar y editar datos de los repartos que tiene asignados.

En cuanto a los datos sensibles, como son las contraseñas de los usuarios, se encriptan *One Way* antes de guardarse la base de datos. El algoritmo de encriptación utilizado es *bcrypt* [18].

Los archivos de las firmas de los destinatarios se guardan en Amazon S3 [19], con quien también se utiliza HTTPS como protocolo de comunicación. Las urls de las fotos contienen un *token* randómico de 16 dígitos para esconderlas del público.

5.3.7. Escalabilidad

Formisur planteó desde un principio su intención de ampliarse, manejando una cantidad cada vez mayor de repartos cada día. Es por eso que se tuvo en cuenta la escalabilidad al momento de definir por ejemplo la plataforma de *hosting* a utilizar, inclinándose hacia aquella que simplifique este proceso. En el ANEXO 11 – "Análisis de tecnologías de hosting" puede visualizarse un análisis comparativo.

La plataforma escogida, Heroku, permite escalar fácilmente tanto horizontal como verticalmente, ya sea aumentando la cantidad de *dynos* o mejorando el poder de procesamiento de los existentes.

En el caso de la manera horizontal, Heroku se encarga de manejar la infraestructura que balancea la carga entre los diferentes *dynos* a través de un manejador de *dynos*, haciendo este proceso transparente para los desarrolladores.

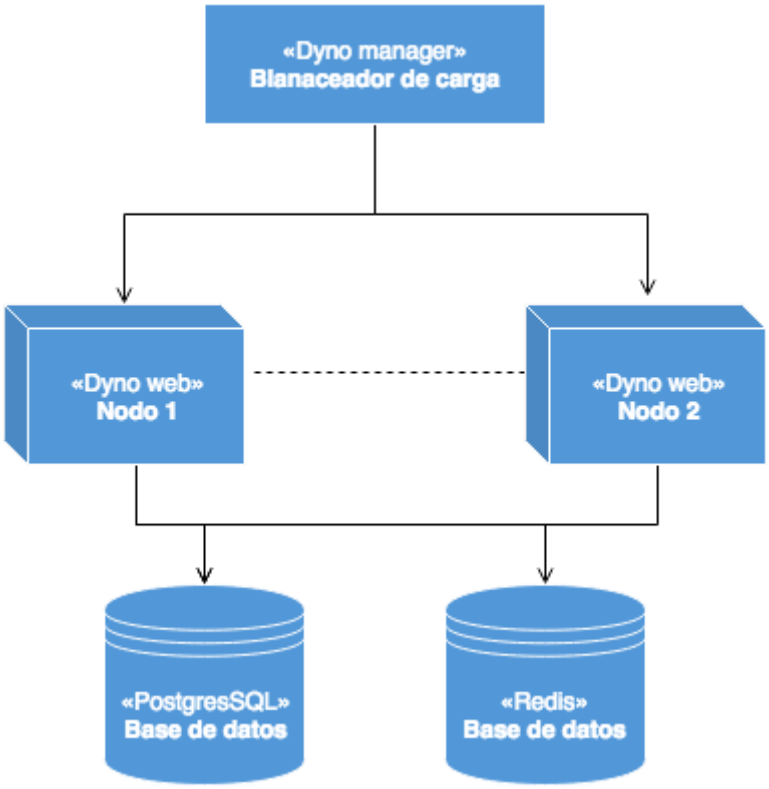


Ilustración 47 - Vista de despliegue de nodos servidores e interacción con la base de datos

Para escalar horizontalmente fue necesario implementar el servidor de forma *Dtateless*, es decir, que cada llamada sea independiente de la anterior para que cada consulta consecutiva del mismo usuario no tenga por qué ser atendida por el mismo

dyno ya que estos no retienen información de la sesión, siendo esta otra de las razones para la construcción de una API REST. Toda la sincronización se realiza utilizando las bases de datos y los *tokens* de autenticación.

Por otro lado, los *workers*, esto es, los *dynos* que no atienden llamadas web sino que obtienen sus tareas de una cola de trabajos alojada en Redis, también pueden escalarse tanto vertical como horizontalmente al igual que los *dynos web*.

De la misma manera, la base de datos puede ser escalada verticalmente si se torna necesario, ya que el *add-on* para Postgres utilizado también permite mejorar el plan contratado a cambio de un mayor costo mensual. Los planes más altos mejoran aspectos como la capacidad de procesamiento de la base, la máxima cantidad de filas permitidas y la máxima cantidad de conexiones concurrentes, sin necesidad de modificar el código.

5.4. Modelo de datos

A continuación se muestra un diagrama con los modelos principales en el back-end con el fin de mostrar cómo se relacionan los modelos.

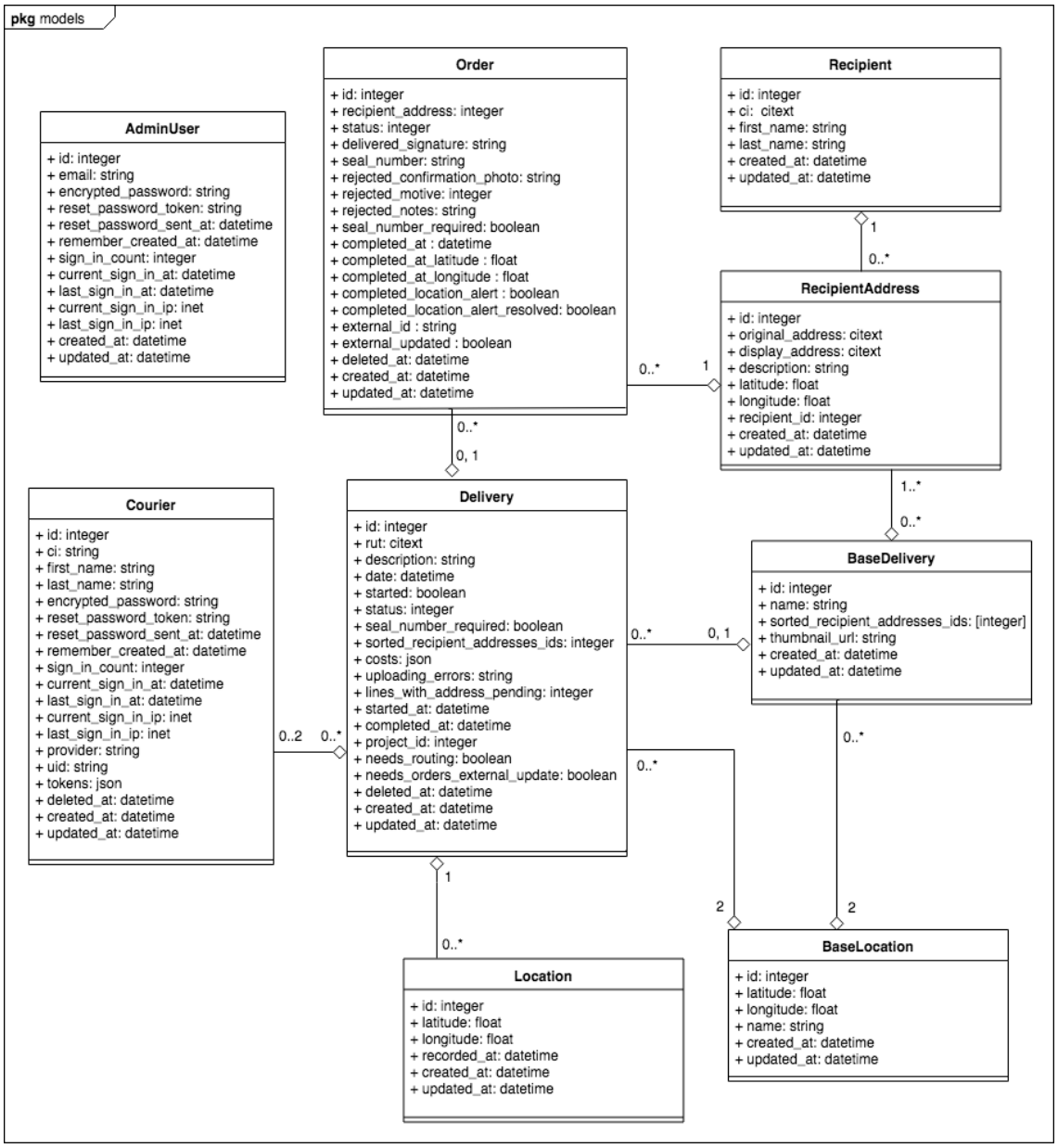


Ilustración 48 - Modelo de datos del back-end

5.5. Análisis de problemas tecnológicos

Esta sección tiene como objetivo listar y justificar las principales decisiones tecnológicas del proyecto. La mayor parte de estas decisiones fueron tomadas una vez de definidos los desafíos de la arquitectura, los principales atributos de calidad y previo a comenzar el desarrollo.

La elección de las tecnologías estuvo dada por una ponderación de los aspectos que más afectaban al proyecto, entre ellas la experiencia del equipo, la adaptabilidad al problema, la cantidad de librerías y soporte, y otras características particulares a cada tecnología.

5.5.1. Para la API

Lenguaje y Framework

El back-end se implementó en Ruby utilizando el *Framework* Rails. Si bien esta decisión estuvo dada por el cliente, el equipo la apoyó por razones que se pueden apreciar en el ANEXO 10 – “Análisis de tecnología back-end”. En resumen, este lenguaje se adapta al objetivo del equipo sobre construir un producto que esté listo para ponerse en producción. Además, existe una amplia variedad de librerías llamadas gemas y soporte disponible debido a la gran comunidad que el *Framework* sustenta. Finalmente, Leticia contaba con experiencia previa en el lenguaje y conocía su adaptabilidad para el problema a enfrentar.

Hosting

El equipo se decidió por la plataforma Heroku para desplegar la API. Esto se dio principalmente porque Heroku oculta la infraestructura, reduciendo casi a cero la necesidad de configuración, permitiendo al equipo desplegar la aplicación con muy poco esfuerzo para poder concentrarse en el desarrollo y no en la infraestructura, todo esto sin necesidad de salir del *tier* gratis. Además, Heroku permite escalar fácilmente como se explicó en la sección 5.3.7- “Escalabilidad” y provee una gran cantidad de *add-ons* útiles como herramientas de monitoreo, integración con base de datos, etc. El análisis completo se puede observar en el ANEXO 11 – “Análisis de tecnologías de hosting”.

Almacenamiento

Se decidió utilizar una base de datos relacional PostgreSQL para almacenar los datos de los repartos, pedidos, usuarios, etc. Principalmente, se eligió PostgreSQL por su fácil integración con Ruby On Rails y Heroku, en el cual se agrega como un add-on sin configuración alguna más que el plan que se quiere utilizar, donde en nuestro caso

el gratis fue más que suficiente. Se puede apreciar un análisis comparativo de tecnologías en ANEXO 12 – “Análisis de tecnologías de almacenamiento de datos”.

Se decidió utilizar el *add-on* para Postgres de Heroku para el ambiente de desarrollo, ya que Heroku configura la variable de entorno y la conexión automáticamente, permitió al equipo tener acceso a la configuración más avanzada como respaldos desde el tablero de Heroku y es gratis.

Sin embargo, al crear el ambiente de producción el cliente pidió utilizar una de sus bases de datos en AWS RDS [20], por razones propias. En ese caso, el equipo no tuvo que hacer más que modificar la variable de entorno correspondiente con la url de la base en Amazon.

Para el almacenamiento de las imágenes como los datos de confirmación de los pedidos (firmas de los destinatarios o fotos de las casas), se utilizó AWS S3. En este caso el equipo no pudo utilizar Heroku ya que este no cuenta con ningún *add-on* que encapsule este servicio y almacenarlo directamente en el *File System* de los dynos no es una opción dado que Heroku explica específicamente que no se haga [21]. Esto es porque cada vez que se despliega un nodo su *File System* se resetea, además de que no se comparten entre *dynos*.

Servicios

Los servicios se expusieron a los distintos clientes a través de REST API. Como formato de comunicación se utilizó JSON y HTML. El protocolo de comunicación fue a través de HTTPS utilizando los distintos métodos de petición como sugiere REST. La especificación de la API expuesta por Geolocalizador se encuentra en el ANEXO 13 – “API REST Geolocalizador”.

5.5.2. Para la aplicación web

Lenguaje y Hosting

La aplicación web está escrita en Ruby On Rails y se encuentra desplegada junto con la API en Heroku como se fue explicando.

Lenguaje y Framework

Para la aplicación web se utilizó un *Framework* llamado Active Admin [22]. Es una gema que permite generar tableros de administrador para las funcionalidades básicas sobre los modelos (creado, listado, filtrado, borrado, edición) muy fácilmente, además de permitir customizarlas de cualquier forma que sea necesaria. Es una de las gemas más populares de Ruby on Rails y el equipo contaba con experiencia utilizándola. Esta librería ahorró muchísimo tiempo de desarrollo en funcionalidades frecuentes que generalmente se comportan de manera similar en todos los sistemas. Permitió al

equipo no tener que escribir código HTML, sino que solamente editar los estilos para mejorar la estética y agregar los componentes personalizados utilizando la sintaxis propia que es muy fácil de comprender, o eventualmente *scripts* para los mapas.

5.5.3. Para la aplicación móvil

Lenguaje y Framework

La aplicación móvil se implementó en Java nativo de Android. Si bien esta decisión estuvo dada por el cliente, al igual que el lenguaje de back-end, el equipo la apoyó por razones que se pueden apreciar en el ANEXO 14 – “Análisis de tecnología front-end para aplicación móvil”. Principalmente fue debido a que los repartidores reciben celulares con sistema operativo Android por razones de costo, por lo que utilizar alternativas híbridas podrían comprometer la usabilidad o el rendimiento sin otorgar ganancias a cambio.

Hosting

La aplicación está alojada en una cuenta privada de Google Console [23]. Este tipo de cuentas solo permite a un grupo configurado de dispositivos acceder a la aplicación. De esta forma se restringe el acceso solamente a repartidores de Formisur y se puede configurar para que la aplicación se descargue automáticamente en esos dispositivos sin tener que figurar públicamente en la Google Play Store [24] y no permitir al repartidor eliminarla.

Almacenamiento

La aplicación utiliza SQLite [25] para el almacenamiento de los datos del reparto para no generar consumo innecesario de memoria ya que mantener esta cantidad de datos en memoria es muy costoso para la misma, además de que se correría el riesgo de perder la información.

Por otro lado se decidió guardar los datos de sesión en las *Shared Preferences* que guarda datos en el formato clave valor como variables globales para poder consultarlas rápidamente sin recurrir a consultas a la base de datos.

5.6. Análisis del problema de enrutamiento

Esta sección tiene como objetivo explicar el problema y la solución para una de las funcionalidades que requirió mayor esfuerzo durante la etapa de desarrollo. La primer solución construida fue sustituida una vez que se encontró una oportunidad que no había aparecido anteriormente. A continuación se busca describir ambas versiones y justificar el retrabajo.

5.6.1. Contexto

El requerimiento que da origen al problema es el RNF10 - El sistema debe enrutar los pedidos de un reparto de forma de obtener el camino más rápido. El desafío que se desprende del requerimiento es un problema *NP-Hard* muy popular conocido como el problema del viajero o TSP por sus siglas en inglés [26]. El problema intenta dar respuesta al siguiente problema: Dada una lista de ciudades y las distancias entre cada par de ellas, ¿cuál es la ruta más corta posible que visita cada ciudad exactamente una vez y al finalizar regresa a la ciudad origen?

En su adaptación en Geolocalizador, las ciudades a ordenar se sustituyen por direcciones de destinatarios finales. Se hablará de nodos, puntos, *waypoints* o direcciones de destinatarios finales en analogía con las ciudades que plantea el problema del viajero. Los nodos se identifican como IDs de los destinatarios finales.

Otra diferencia es que en el problema a resolver no necesariamente se regresa al nodo de origen al finalizar el recorrido.

Otro punto a tener en cuenta es que el problema supone que las distancias (costos) entre los nodos son conocidas, cuando en la realidad deben ser consultadas y no dependen solamente de la distancia euclidiana entre ellos sino que además varían según la disposición de las calles, el tráfico, etc.

El equipo consideró que las siguientes características del problema también podrían influir para diseñar la solución:

1. Cada reparto toma una jornada entera y son de 100 a 130 pedidos.
2. Todos los pedidos son en un mismo barrio.
3. Cada reparto entrega para un cliente.
4. Los repartos de un mismo cliente suelen contener siempre los mismos pedidos, difiriendo en hasta 10.
5. El lugar de salida y llegada de los repartidores suele corresponder con una de las ubicaciones bases de Formisur.
6. Hoy en día la empresa armó a mano una ruta base para cada reparto.

5.6.2. Investigación de alternativas

El equipo sabía que esta funcionalidad constituía solamente una de las funcionalidades de prioridad media en el gran *Backlog* a construir, y que debido a la alta complejidad de la misma y al reducido tamaño del equipo, se debería investigar y planear cuidadosamente para evitar retrabajo o subestimaciones. Es por eso que se decidió invertir tiempo en definir un plan de investigación para proveer una solución que entregue valor al cliente sin que implique dejar de lado funcionalidades importantes, considerando que el objetivo principal del sistema era otro.

Durante el *Spike* de investigación se realizó una investigación de alternativas actuales para resolver el problema. Se consideraron tanto contratar alternativas en la nube (servicios SaaS) como utilizar algoritmos propios.

El equipo primero intentó realizar un prototipo de un algoritmo a fuerza bruta, el cual se detalla en el ANEXO 15 – “Algoritmo fuerza bruta para problema de enrutamiento”, el cual se corrió en servidores de Heroku con un reparto de 130 pedidos y se descartó ya que no terminó a tiempo.

Por otro lado, en el ANEXO 16 – “Análisis de servicios de enrutamiento” se puede apreciar una comparación de más de diez servicios para este problema. Resumiendo los resultados del mismo, la mayoría de estos servicios encontrados solo ordenan hasta 50 *waypoints* o tienen un costo mensual mayor a USD 200 lo que se encontró por fuera de lo que la empresa estaba dispuesta a pagar por la funcionalidad.

Debido a la característica número 2, que refiere a que todos los pedidos de un reparto son generalmente en el mismo barrio, utilizar *Clustering* para subdividir el recorrido y utilizar alternativas en la nube no fue posible.

5.6.3. Solución MVP: Algoritmo propio

Habiendo descartado las alternativas anteriores por las razones descritas, el equipo decidió hacer uso de las características del contexto 3, 4, 5 y 6 mencionadas anteriormente y diseñar un algoritmo propio. La solución construida para el MVP (*Release* 1) contuvo un algoritmo que reutiliza los recorridos de los repartos o rutas base para los nuevos repartos.

La solución permite a los administradores:

- Administrar rutas base como un nuevo modelo (agregar, editar, borrar, etc.).
- Asignar una ruta base a un reparto que se está creando. (En un futuro se planeaba hacer la asignación automáticamente).

Un trabajo en segundo plano corre periódicamente para asegurarse que los costos de los puntos consecutivos de una ruta base se tengan calculados y actualizados. Para esto consulta la API Traffic de Google Maps [27]. Los costos se almacenan desnormalizados por temas de eficiencia, en formato diccionario en la base de datos junto con el registro de la ruta base correspondiente.

Al agregar un nuevo reparto el sistema inicia un trabajo asíncrono que calcula el recorrido de la siguiente manera:

1. Ignora los puntos del reparto base asignado que no están en el nuevo reparto.
2. Busca entre qué pares de puntos interpolar los nuevos puntos tal que el desvío sea el mínimo.

3. Guarda el recorrido en el reparto, como un array de IDs de direcciones de destinatarios.
4. Actualiza el reparto base agregándole los nuevos puntos en la posición que corresponden y los nuevos costos que se tuvieron que calcular.

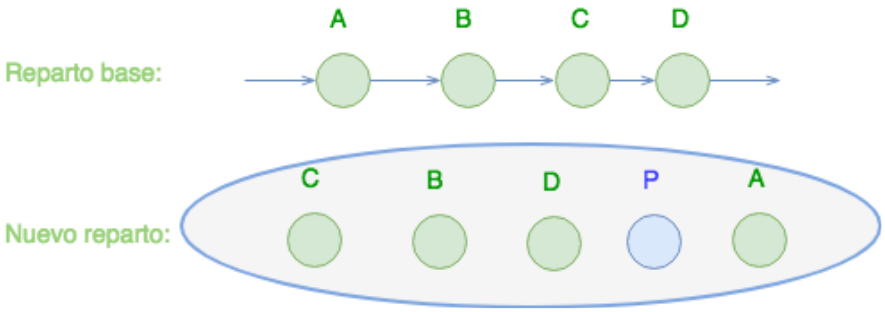


Ilustración 49 - Interpolación de nodo en algoritmo de enrutamiento

Para cada nuevo punto P que hay que ordenar que no se encuentra en el reparto base, se debe encontrar y colocarlo entre los puntos {I, J} consecutivos (incluyendo vacío) /

$$\min(\overline{IP} + \overline{PJ} - \overline{IJ})$$

La razón por la que se incluye vacío es porque los nuevos puntos podrían llegar a colocarse al principio o al final del recorrido.

A continuación se presenta un diagrama de clases de los servicios creados para esta versión del algoritmo de enrutamiento.

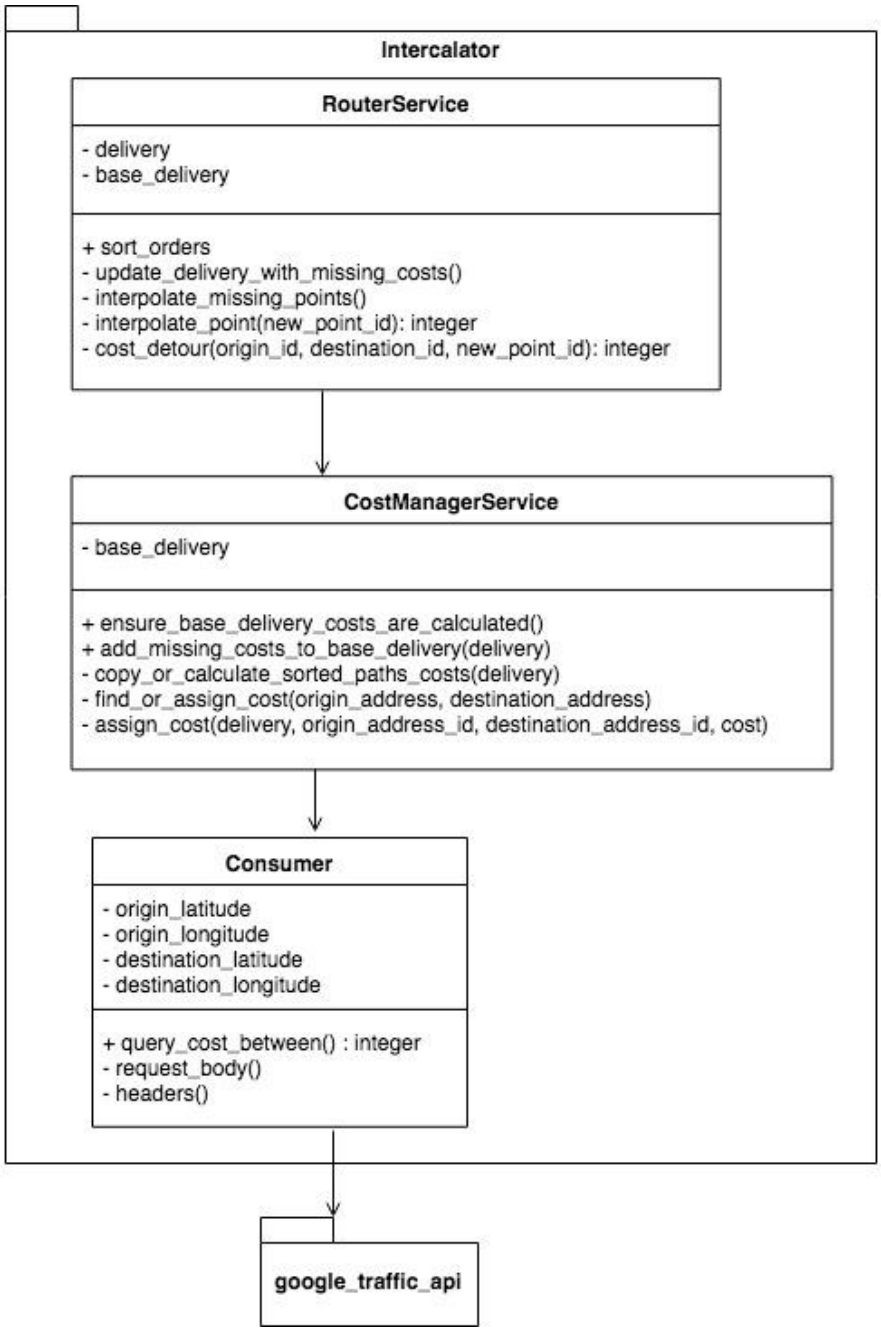


Ilustración 50 – Diagrama de servicios para el algoritmo propio de enrutamiento

Para la implementación de este algoritmo se crearon tres servicios con responsabilidades bien definidas. El primero, *RouterService*, es quien expone el método *sort_orders*.

El objetivo del método es actualizar el atributo *sorted_recipient_addresses_ids* del reparto para que sea un array con las IDs de las direcciones de los destinatarios cuyo orden representa el camino más corto.

El método *sort_orders de RouterService* primero convoca a *CostManagerService* quien se asegura de que la ruta base seleccionada para el reparto contenga calculados todos los costos necesarios para llevar a cabo el algoritmo. Esto implica calcular el costo ida y vuelta de todos los nuevos nodos que no se encontraban en la ruta base con todos los nodos ya existentes. Los costos se calculan a través de la API de Google, estando esta llamada encapsulada como para poder ser sustituida en algún momento por otro proveedor sin un mayor impacto.

Una vez que todos los costos se encuentran calculados, el algoritmo quita los nodos de la ruta base que no se encuentran en el reparto a ordenar. Luego itera sobre los nuevos nodos, buscando el par de nodos donde insertarlos provocaría el menor desvío y los inserta, en tanto el reparto como en la ruta base.

Los métodos cumplen con el principio de separación de comandos y consultas, ya que o realizan modificaciones sobre el objeto del reparto o la ruta base o responden un valor pero no ambas.

5.6.4. Solución V1: Routeme

Contacto

Luego de que el algoritmo de la versión del MVP estaba terminado el equipo encontró una oportunidad interesante. Al lugar de trabajo de Leticia ingresó un nuevo empleado, Santiago Bartesaghi, *Co-Founder* de Routeme, servicio en la nube para planificar recorridos. El algoritmo evolutivo que construyó el equipo de Santiago fue parte de su proyecto de grado para la obtención del título de Ingeniero en Sistemas en la Facultad de Ingeniería, que luego convirtieron en un servicio como emprendimiento personal.

Leticia se interesó por el servicio que ofrecían por lo que Santiago le ofreció realizar una prueba en conjunto. Leticia obtuvo del sistema Geolocalizador, con consentimiento de los clientes, una lista de las coordenadas de uno de los repartos de producción con alrededor de cien pedidos. Se introdujo la lista de coordenadas en Routeme y se generó, al cabo de pocos minutos, un recorrido y links para visualizarlo en Google Maps, donde se obtuvo el tiempo estimado del recorrido. El mismo fue comparado con el tiempo estimado del recorrido del reparto base que utilizaban los clientes, que también fue puesto en Google Maps para obtener un estimativo del tiempo.

La diferencia entre los tiempos de ambos recorridos era de más de 20 minutos a favor del de Routeme. El nuevo recorrido fue enviado a los clientes que también lo aprobaron. Allí, luego de pensar junto al tutor y con los clientes si el retrabajo valía la pena o si aportaba más valor construir nuevas funcionalidades, se llegó a la conclusión de que era un tiempo que estaban dispuestos a invertir.

Luego de tomada la decisión, el equipo puso en contacto a Martín García (Encargado de Sistemas de Formisur) y Santiago para que puedan negociar un precio.

Routeme posee muchas otras ventajas además de recorridos precisos. Cuenta con una página web para el desarrollador donde se ingresa a la cuenta y se visualizan todos los recorridos calculados, con links para exportar y visualizar en Google Maps, entre otras funcionalidades. La API es muy sencilla de integrar y se encuentra bien documentada. Además, al trabajar junto a Santiago, Leticia tuvo soporte personalizado mientras realizaba la integración.

Arquitectura

La comunicación con Routeme se realiza a través de una API REST. En la web de Routeme se generan claves de la API para identificar a la aplicación e instrucciones de cómo firmar los pedidos utilizando esta clave de autenticación. Allí también se configura la URL de Geolocalizador donde se esperan los *webhooks* con las rutas optimizados.

Al crear un reparto, Geolocalizador inicia la comunicación mediante un pedido HTTPS para crear lo que Routeme denomina un proyecto. En el pedido se incluye las coordenadas de las direcciones a ordenar, con las IDs de las respectivas direcciones de destinatarios para posterior identificación. En la respuesta se recibe una ID de proyecto que se almacena junto con los datos del reparto.

Cuando Routeme termina de calcular la ruta óptima, envía un pedido HTTPS a Geolocalizador, con la ID del proyecto y la lista de direcciones de destinatarios ordenadas. Geolocalizador guarda la lista junto con los datos del reparto correspondiente y crea un reparto base con esas direcciones ordenadas para evitar futuras consultas por las mismas direcciones.

El siguiente diagrama resume las interacciones descritas anteriormente.

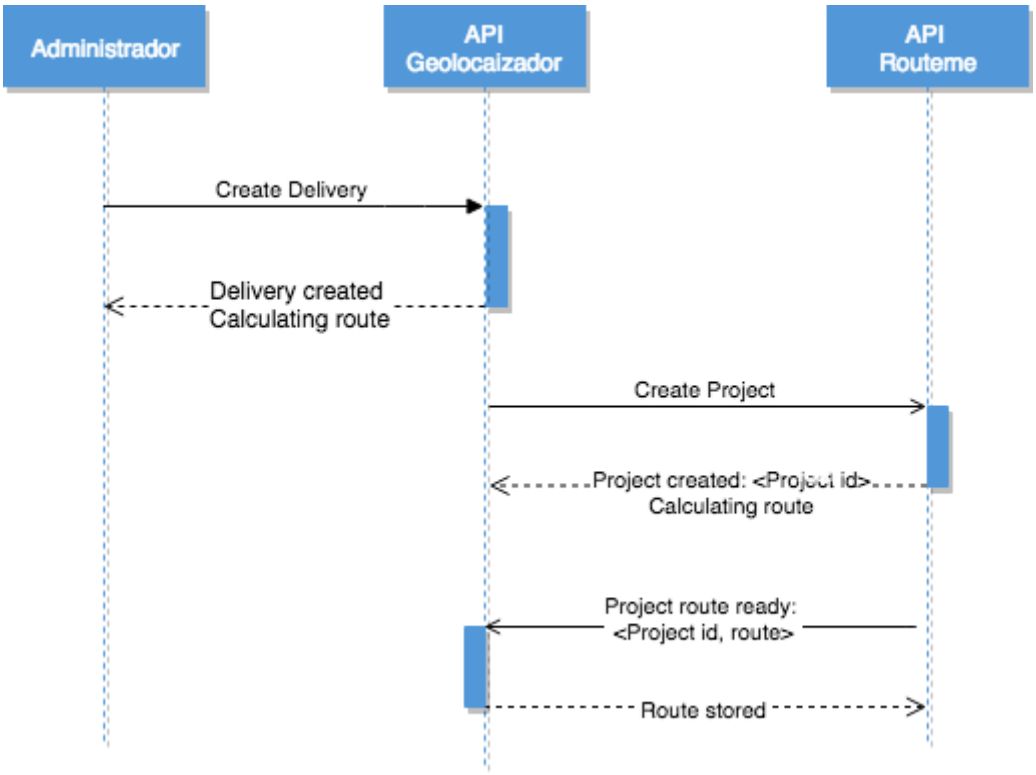


Ilustración 51 - Diagrama de secuencia de comunicación con Routeme

Observar cómo la creación del proyecto en Routeme se realiza de manera asíncrona, para que el administrador no deba esperar la respuesta de este. La llamada que inicia Routeme cuando termina de calcular la ruta permite que Geolocalizador no tenga que estar constantemente utilizando *Polling* [28] para obtener el estado de un proyecto, reduciendo el costo computacional.

Otros detalles de la implementación consisten de tareas periódicas que corren con el objetivo de expirar rutas base mayor a tantos días (configurable desde el ambiente de Heroku), o para solicitar manualmente a Routeme el estado de un proyecto cuyo *webhook* no se ha recibido.

Por otro lado, fue necesario que el administrador al crear el reparto seleccione la ubicación base de donde sale y llega el repartidor ya que estas influyen en el recorrido óptimo a realizar.

En el siguiente diagrama se representan los modelos relevantes al problema de enrutamiento.

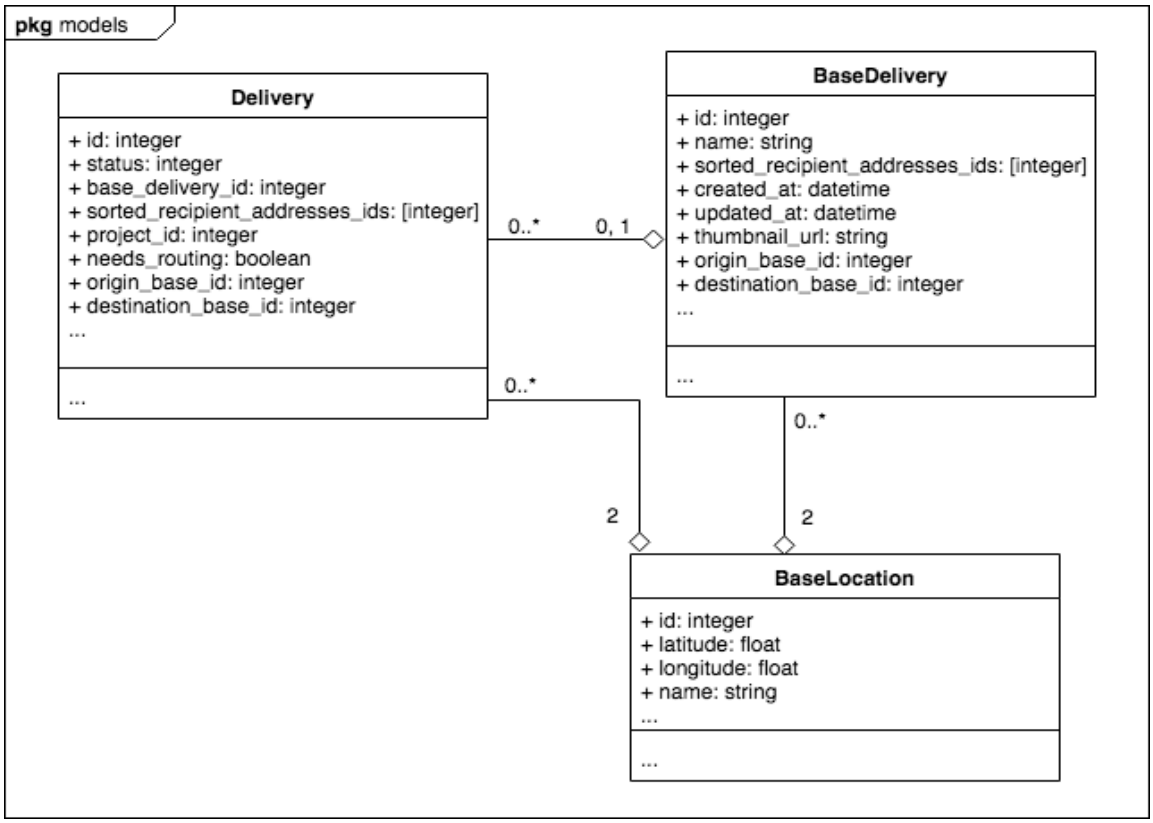


Ilustración 52 - Modelos relevantes al problema de enrutamiento

Como se puede apreciar, un reparto tiene dos ubicaciones base, una origen y una destino, y una referencia a un reparto base que permanece vacía hasta que se procesa el *webhook* de Routeme con la ruta óptima y se crea el registro del reparto base. El reparto base también guarda las dos ubicaciones base, de origen y destino, ya que estas afectan el recorrido.

6. Gestión de calidad

En este capítulo se describen los objetivos de calidad definidos para el producto y el proceso, así como las métricas utilizadas para evaluarlos y las actividades realizadas para asegurar su cumplimiento.

6.1. Objetivos de calidad

6.1.1. Del producto

Como definición de calidad del producto se utilizó la definición de Joseph Jurán, quien explica que la palabra calidad tiene múltiples significados, siendo los dos siguientes los más representativos: "la calidad de software es el conjunto de características de un producto que satisfacen las necesidades de los clientes y, en consecuencia, hacen satisfactorio el producto" y "la calidad consiste en no tener deficiencias en el producto o en el proceso" [29].

Es por ello que se definieron los siguientes objetivos del producto:

1. Lograr un producto que sea puesto en producción antes de la entrega final
2. Lograr un producto con cero defectos críticos reportados al momento de la entrega final
3. Conseguir un promedio mayor a 4 y ningún resultado menor a 3 en la encuesta de satisfacción a los interesados al terminar el proyecto

Para lograr los primeros dos a lo largo del proyecto se buscó validar y verificar las funcionalidades del producto con los interesados, midiendo su satisfacción a través de encuestas, con el objetivo final de lograr un producto aprobado por el cliente para salir a producción.

Por otro lado, se buscó lograr un producto sin defectos críticos por lo que el equipo se concentró en definir y cumplir en todo momento con el plan de gestión de incidentes y el plan de *testing*.

6.1.2. Del proceso

El equipo definió que un proceso exitoso sería aquel del que saliera un producto de calidad, mientras que se permite a los integrantes, tanto estudiantes como tutor y clientes, trabajar e interactuar de manera cómoda y eficientemente.

Buscando eso, se definieron los siguientes objetivos medibles:

1. Trabajar entre 10 y 30 horas semanales por persona a lo largo de todo el proyecto.

2. Obtener un porcentaje de retrabajo menor al 5% de las horas totales para el final del proyecto.
3. Lograr que el tiempo dedicado a tareas de gestión o de coordinación como reuniones y actas sea menor al 10% de las horas totales para el final del proyecto.

6.2. Plan de calidad

Al comenzar el proyecto el equipo definió un plan de calidad alineado con los objetivos propuestos que fue utilizado como guía a lo largo del proyecto. El plan especificó qué procedimientos se aplicarían en qué momento del proyecto y por quién. Para eso se subdividió el proyecto en fases donde se realizaron diferentes actividades de calidad. Estas fueron: análisis del negocio, ingeniería de requerimientos, arquitectura, desarrollo y pruebas. También se definieron actividades a realizar independientemente de la fase.

El plan de calidad se puede visualizar en el ANEXO 17 – “Plan de calidad”. A continuación se ejemplifica una actividad a realizar por fase.

Análisis del negocio: para obtener la información necesaria acerca del contexto del problema, fue muy importante realizar entrevistas a los interesados, como clientes, repartidores, administradores y empresas cliente. Los resultados se describen en el capítulo 2 – “Problema y solución”.

Ingeniería de requerimientos: para especificar los requerimientos se utilizó principalmente la técnica *Story Mapping*. La misma también fue de utilidad para visualizar la mejor forma de priorizar los requerimientos. La aplicación de esta técnica se encuentra en el capítulo 4 – “Ingeniería de Requerimientos”.

Arquitectura: una de las actividades relacionadas con la arquitectura, posterior a la identificación de los requerimientos, fue la realización de un primer diseño de la misma, donde se definió utilizar tácticas y patrones para los problemas a resolver. La arquitectura del sistema se describe en el capítulo 5 – “Arquitectura y desarrollo”.

Desarrollo: durante el desarrollo se utilizaron herramientas de análisis de calidad de código para asegurar la calidad del código a integrar, validando los estándares definidos al principio del proyecto. Los estándares a utilizar se especifican en el ANEXO 18 – “Estándares de codificación” y ANEXO 19 – “Aplicación de Estándares“.

Pruebas: se definió y llevó a cabo un plan de pruebas que se puede encontrar en la sección 6.4.2 – “Pruebas de software” que involucra diferentes tipos de pruebas, por ejemplo de integración y de validación.

Actividades independientes a las fases: al comienzo del proyecto se realizó un plan para gestionar los riesgos, que implicó una evaluación periódica de la evolución de los mismos a lo largo de todo el proyecto, como se detalla en la sección 7.5 - “Gestión de riesgos”.

6.3. Atributos de calidad

Además de definir objetivos del producto, el equipo identificó atributos de calidad que debe cumplir el sistema. Muchos de estos se resolvieron a través de la arquitectura, como se describe en la sección 5.3 – “Atributos de calidad”. Otros, en especial usabilidad y mantenibilidad, requirieron acciones adicionales.

A continuación se describen los atributos de calidad que no quedaron automáticamente resueltos a través de la arquitectura, explicando por qué fueron importantes y el plan seguido para asegurarlos.

6.3.1. Mantenibilidad

Este atributo de calidad recibió particular atención durante el proyecto ya que Martín García sería el encargado de mantenerlo una vez terminado el proyecto, por lo que pidió específicamente al equipo que el software sea altamente mantenible.

Se trabajó en conjunto para definir un objetivo medible para la mantenibilidad. Para eso se decidió integrar los repositorios con Code Beat [30], herramienta de uso gratuito que evalúa la calidad del código promediando una nota al mismo.

Se planteó como objetivo mantener una nota de A en todo momento. La nota se puede observar en una insignia que se colocó en el archivo *README* del proyecto o directamente en la consola de Code Beat. Para analizar el nuevo código que se integra, se configuró Code Beat como parte de los chequeos automáticos que se realizan antes de permitir la integración de un *Pull Request*, junto con las pruebas unitarias y herramientas de análisis de código.

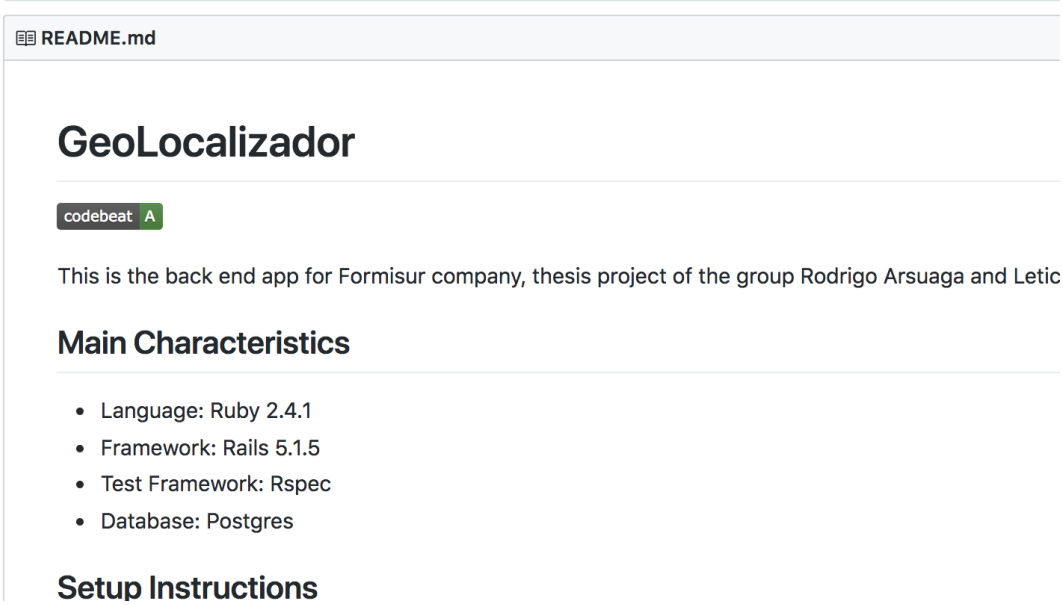


Ilustración 53 - Visualización de insignia de CodeBeat en el README del repositorio

Para asegurar la mantenibilidad del sistema se realizaron diferentes acciones que serán descritas a lo largo del capítulo, entre ellas el uso de herramientas de análisis de calidad de código, seguimiento de estándares y realización de pruebas unitarias.

6.3.2. Usabilidad

Este atributo se tuvo en cuenta para tanto la aplicación web del administrador como para la aplicación móvil del repartidor. Sin embargo, en la parte móvil tomó mucha más importancia ya que una buena usabilidad reduciría los errores cometidos y por tanto el tiempo que toma realizar una tarea. Considerando que el repartidor entrega entre 100 y 130 pedidos por reparto, cada minuto que se ahorre para completar la entrega de un pedido puede repercutir en el tiempo total del reparto. Por otro lado, se debió considerar que los repartidores de Formisur no estaban especialmente acostumbrados a trabajar con tecnología y que muchas veces la aplicación se utilizaría en la calle.

Para medir la usabilidad de la aplicación, se definió una encuesta que fue completada por los repartidores y por el administrador luego de los repartos de prueba. El equipo se propuso y alcanzó el objetivo final de obtener un promedio por arriba de 4 en el resultado de estas encuestas en la última prueba. Las encuestas y sus resultados se detallan en el ANEXO 21 – “Encuestas de satisfacción”.

Para asegurar la usabilidad de la aplicación móvil se realizaron las siguientes actividades:

Delegación de los diseños a una experta

Con el objetivo de favorecer la usabilidad desde un principio, los integrantes del equipo decidieron delegar los diseños finales a una diseñadora gráfica especializada en usabilidad de aplicaciones móviles. La diseñadora corrigió algunos aspectos de los prototipos iniciales que realizó el equipo. Procuró utilizar componentes nativos para hacer de la aplicación lo más intuitiva posible.

Con el objetivo de mantener la usabilidad a lo largo de la etapa de desarrollo, la diseñadora fue opinando sobre la usabilidad de la aplicación web y realizó recomendaciones sobre el diseño y flujos de la misma.

Análisis heurístico

Se realizó un análisis heurístico para revelar problemas de usabilidad que se describe más adelante en el capítulo.

Componentes nativos

La implementación a través de un lenguaje nativo y la correcta utilización de los estándares visuales del sistema operativo fueron necesarios para crear una experiencia de usuario intuitiva y familiar.

Inspiración en aplicaciones que los repartidores utilizan

Siguiendo con el objetivo de lograr una experiencia de usuario familiar para los repartidores, el equipo se inspiró en diseños de aplicaciones que están acostumbrados a utilizar. Por ejemplo, el cliente comentó que los repartidores utilizan la red social Badoo, por lo que a Rodrigo se le ocurrió copiar la acción de *Swipe* que se utiliza en la misma para postergar un pedido en Geolocalizador.

Desarrollar utilizando el mismo modelo del dispositivo que utilizan los repartidores

El equipo tuvo el desafío que el modelo de celular que reciben los repartidores es muy básico por lo que suele responder más lento y el desarrollador debe ser más consciente de los recursos disponibles. Para reducir diferencias entre el funcionamiento esperado por el desarrollador y el real, el equipo pidió a Formisur un celular prestado durante toda la etapa de desarrollo, para asegurar que el diseño y los recursos necesarios sean compatibles.

Interfaz no bloqueante por respuesta de servidor

Como se describe en la sección 5.3.2- “Disponibilidad”, la arquitectura del sistema asegura que la aplicación móvil no se bloquee esperando la respuesta del servidor durante un reparto, permitiendo que el repartidor perciba una respuesta inmediata lo que mejora la usabilidad.

Pruebas de usabilidad

Como se fue mencionando, se realizaron instancias de pruebas para validar la usabilidad como se detalla en la sección 4.3.2 – “Pruebas de usabilidad”.

6.4. Aseguramiento de la calidad

A continuación se describen algunas de las actividades realizadas definidas en el plan de calidad que buscan lograr los objetivos de calidad propuestos.

6.4.1. Aplicación de estándares

La especificación de los estándares a utilizar en los principales entregables del proyecto, código y documentación fue crucial para asegurar que ambos estudiantes mantuvieran un mismo criterio a la hora de trabajar.

Estándares de documentación

El equipo siguió los siguientes estándares provistos por la Universidad ORT Uruguay al momento de documentar:

Documento	Contenido
302	Conjunto de normas y estándares a seguir para el documento del proyecto final de la Facultad de Ingeniería
303	Lista de verificaciones del formato para el documento del proyecto final de la Facultad de Ingeniería
304	Proceso de entrega y correcciones para el documento del proyecto final de la Facultad de Ingeniería
306	Guías para títulos, *Abstract* e información de corrección para el documento del proyecto final de la Facultad de Ingeniería

Tabla 2 - Estándares de documentación

Estándares de codificación

Con el objetivo de lograr un código altamente mantenible y fácil de comprender, al comienzo del proyecto se definieron ciertos estándares de codificación para cada lenguaje. Estos se encuentran en el ANEXO 18 – “Estándares de codificación”. En el ANEXO 19 – “Aplicación de Estándares” se puede ver cómo se aplicaron dichos estándares.

6.4.2. Pruebas de software

El equipo consideró fundamental la elaboración de un plan de pruebas previo al comienzo de la etapa de desarrollo para saber de antemano cómo detectar y cuándo resolver los *bugs* evitando que se acumulen, que se arrastren aquellos de severidad crítica en etapas donde no deberían haberlos o que se pierda tiempo arreglando los de severidad baja en momento equivocados.

El plan de pruebas cubre aspectos como la definición de un criterio para las pruebas y categorización, seguimiento y respuesta ante defectos.

Para decidir qué tipo de pruebas se deberían realizar en qué momento el equipo consideró primero qué tipo de prueba es apropiado para cada objetivo.

Objetivo	Mejor técnica
Encontrar bugs nuevos	Pruebas manuales
Detectar regresiones (funcionalidades que solían funcionar pero que dejaron de hacerlo)	Pruebas de integración automáticas
Codificar componentes de forma robusta	Pruebas unitarias

Tabla 3 - Categorización de pruebas por objetivo

Pruebas automáticas

Para evitar introducir fallas al agregar una nueva funcionalidad o refactorizar, se decidió implementar pruebas automáticas a nivel de back-end.

Se utilizó el *Framework* Rspec para el desarrollo de pruebas automáticas. Rspec es uno de los *Frameworks* de pruebas más populares para Rails y facilita muchas configuraciones previas a cada prueba, la lectura del reporte y el análisis de cobertura.

Se decidió que cada *Pull Request* debería agregar código de pruebas de la nueva funcionalidad o corrección de *bug* y que se debería mantener una cobertura por arriba del 90% en cada nueva rama a integrar. Al finalizar el proyecto se contaba una cobertura superior a 96%.

```
Randomized with seed 18375
.....................................................................................................................
.....................................................................................................

Finished in 27.43 seconds (files took 8.25 seconds to load)
248 examples, 0 failures

Randomized with seed 18375

Coverage report generated for RSpec to /Users/leti/Desktop/Proyectos/geolocalizador-api/coverage. 1504 / 1563 LOC (96.23%) covered.
Leticias-MacBook-Pro:geolocalizador-api leti$
```

Ilustración 54 - Cobertura de pruebas

El repositorio del back-end en Github se integró con la herramienta CircleCI para que todo el conjunto de pruebas unitarias y herramientas de análisis de calidad de código sean ejecutados en cada *Pull Request* y cada vez que se integra una rama a la rama principal, imposibilitando la integración de una rama en la que haya fallado al menos una prueba o regla de calidad. Esta práctico tuvo un enorme impacto en la calidad del código integrado permitiendo agregar nuevas funcionalidades con mayor confianza y entregando código con pocos defectos.

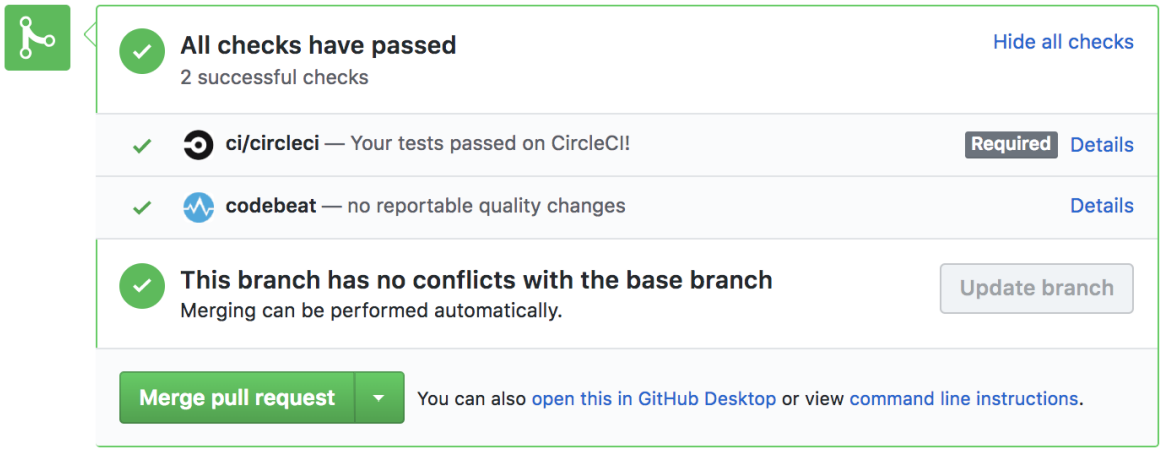
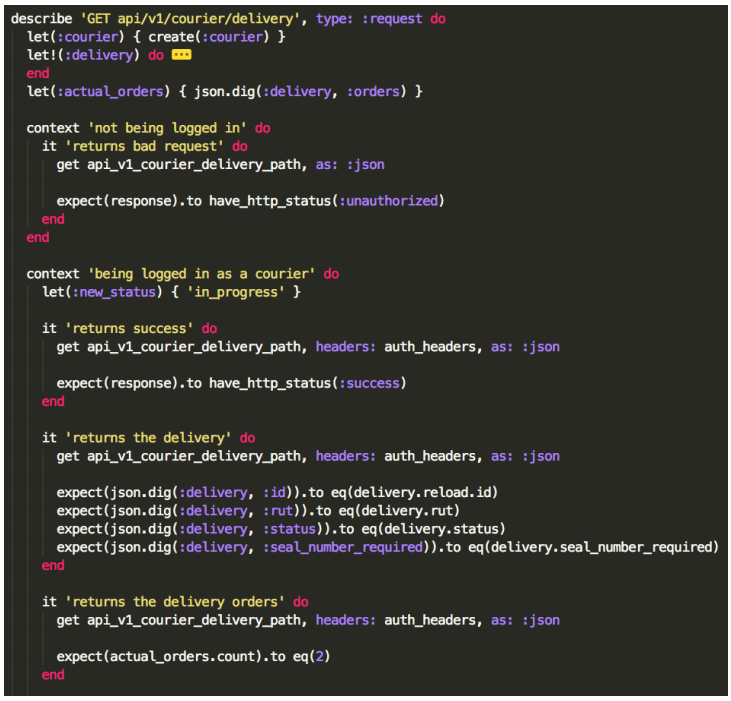


Ilustración 55 - Integración continua

Las pruebas automáticas se realizaron en diferentes niveles.

Por un lado, se realizaron pruebas automáticas de integración (a nivel de *Request* HTTP) para los servicios de la API provistos para la aplicación móvil.



```ruby
describe 'GET api/v1/courier/delivery', type: :request do
  let(:courier) { create(:courier) }
  let!(:delivery) do ···
  end
  let(:actual_orders) { json.dig(:delivery, :orders) }

  context 'not being logged in' do
    it 'returns bad request' do
      get api_v1_courier_delivery_path, as: :json

      expect(response).to have_http_status(:unauthorized)
    end
  end

  context 'being logged in as a courier' do
    let(:new_status) { 'in_progress' }

    it 'returns success' do
      get api_v1_courier_delivery_path, headers: auth_headers, as: :json

      expect(response).to have_http_status(:success)
    end

    it 'returns the delivery' do
      get api_v1_courier_delivery_path, headers: auth_headers, as: :json

      expect(json.dig(:delivery, :id)).to eq(delivery.reload.id)
      expect(json.dig(:delivery, :rut)).to eq(delivery.rut)
      expect(json.dig(:delivery, :status)).to eq(delivery.status)
      expect(json.dig(:delivery, :seal_number_required)).to eq(delivery.seal_number_required)
    end

    it 'returns the delivery orders' do
      get api_v1_courier_delivery_path, headers: auth_headers, as: :json

      expect(actual_orders.count).to eq(2)
    end
```

Ilustración 56 - Ejemplo de pruebas de integración de la API

También se realizaron pruebas automáticas de integración a nivel de controlador para los servicios de la aplicación web, utilizando una herramienta para realizar pruebas que simula a un usuario interactuando con la página web.

```ruby
RSpec.describe Admin::DeliveriesController, type: :controller do
  render_views

  let(:page) { Capybara::Node::Simple.new(response.body) }
  let(:origin_base) { create(:base_location) }
  let(:destination_base) { create(:base_location) }

  describe 'GET index' do
    context 'not being signed in' do
      it 'redirects to log in' do
        get :index

        expect(response).to have_http_status(302)
        expect(response).to redirect_to '/admin/login'
      end
    end

    context 'being signed in' do
      let!(:delivery) { create(:delivery) }

      before do
        login_admin
      end

      it 'returns http success' do
        get :index

        expect(response).to have_http_status(:success)
      end

      it 'renders the expected columns' do
        get :index

        expect(page).to have_content(delivery.rut)
        expect(page).to have_content(delivery.description)
        expect(page).to have_content(delivery.orders_count)
      end

      describe 'filters' do ···
      end
    end
  end
```

Ilustración 57 - Pruebas de integración de la aplicación web

Para métodos concretos de alta complejidad se realizaron pruebas unitarias a nivel de modelo. Los trabajos asíncronos y los servicios también fueron probados unitariamente.

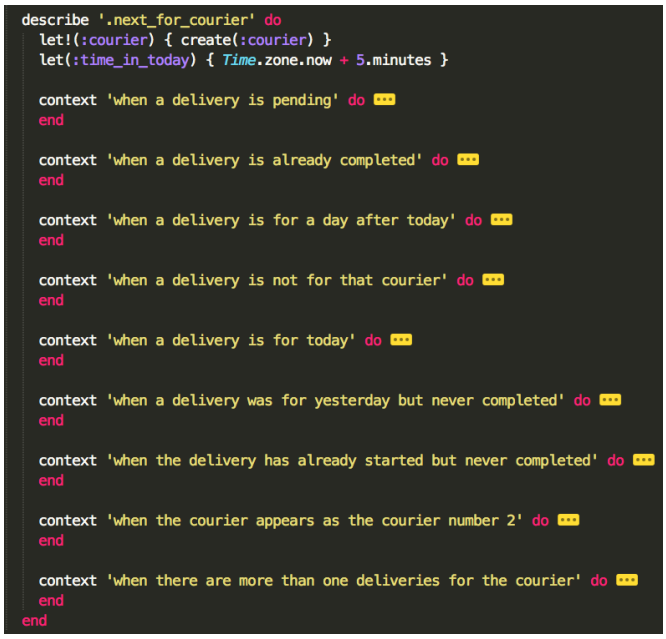


Ilustración 58 - Pruebas a nivel del modelo

Pruebas de cada desarrollador

Ambos integrantes realizaron pruebas de las funcionalidades en las que estaban trabajando durante el *Sprint*, para poder verificar que una tarea fue completada correctamente.

Pruebas de integración

El equipo fue probando manualmente todas las funcionalidades implementadas en el *Sprint* antes de la *Sprint Review* correspondiente para asegurar que no haya defectos de prioridad 1 y 2 al mostrar al cliente. En estas pruebas se integró el front-end con el back-end para verificar que funcionaban bien en conjunto. Se repasaron los criterios de aceptación con los casos comunes, particulares y bordes. Estas pruebas fueron planificadas pero no documentadas.

Pruebas de sistema

A su vez, ambos integrantes hicieron pruebas del sistema con el objetivo de encontrar nuevos defectos, regresiones o posibles mejoras al terminar cada *Milestone*. Estas fueron pruebas formales que requirieron definir previamente un conjunto de casos de prueba, seguirlos, documentar su resultado y registrar los defectos encontrados apropiadamente para su posterior manejo. Se pueden encontrar en el ANEXO 22 – “Reuniones de coordinación”

Pruebas de validación

Se realizaron pruebas de validación con los usuarios, repartidores y administradores, para poder validar el producto construido y obtener retroalimentación. Se realizaron de diferentes maneras, ya sea utilizando prototipos, funcionalidades concretas durante las *Sprint Reviews*, y todo el sistema construido hasta el momento en pruebas formales en repartos reales como se explicó en la sección 4.3 – “Validación”.

Pruebas de usabilidad con usuarios

Dado que la usabilidad fue uno de los principales atributos de calidad a tener en cuenta, se realizaron pruebas con los usuarios para identificar dificultades en el sistema. El objetivo fue obtener retroalimentación de los repartidores y del administrador.

Registro de pruebas

Se decidió seguir un formato durante la documentación de las pruebas con el fin de unificar y formalizar el proceso.

Los casos de prueba se documentaron en una tabla utilizando las siguientes columnas agrupadas por funcionalidad:

Qué se quiere probar	Entradas	Resultado Esperado	Funciona

Tabla 4 - Formato para la documentación de pruebas

De esta manera es posible visualizar los diferentes escenarios para cada funcionalidad, junto con el resultado esperado y si se encontraron defectos o no.

6.4.3. Ambientes de desarrollo

Se definió construir dos ambientes diferentes para la aplicación Web y la API. El primero, *develop*, fue el ambiente de desarrollo donde el equipo trabajó durante la etapa de desarrollo. El segundo, *prod*, fue el ambiente creado a partir de la primer prueba en repartos reales, para que la empresa lo utilizara en sus repartos de producción. Se definió así para permitir al equipo continuar modificando y agregando funcionalidades una vez que el sistema ya había sido puesto en producción, sin afectar este ambiente hasta que el cliente haya aprobado los cambios y el equipo esté seguro de que no introducirá defectos.

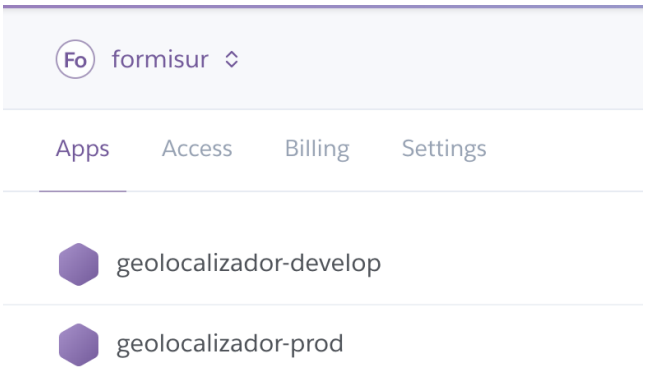


Ilustración 59 - Ambientes de desarrollo en Heroku

En la práctica el ambiente de producción solo se utilizó durante las pruebas en repartos reales, y recién regularmente a fines de la etapa de desarrollo, cuando la empresa adoptó el programa oficialmente.

Análogamente, para la aplicación móvil también existen dos versiones. Tanto la versión de producción como la beta se encuentran en una cuenta privada de la Google Cloud.

6.4.4. Gestión de incidentes

El equipo definió un plan de gestión de incidentes que tuvo como intención explicar cómo proceder ante la aparición de un defecto. Tanto el plan como ejemplos de *bugs* reportados durante el proyecto y las herramientas utilizadas durante la gestión de incidentes se detallan en el ANEXO 20 – “Plan de gestión de incidentes”.

En el plan se describe cómo se utilizó el tablero de Trello para el seguimiento de *bugs*, agregando una columna para nuevos defectos reportados por cualquier integrante. En el plan también se define la categorización de defectos, la forma de reportarlos, y una cantidad máxima de defectos de cada categoría permitidos por ambiente de desarrollo antes de que sea necesario introducir un *Sprint* de arreglo de defectos.

6.4.5. Revisiones

Durante el proyecto se realizaron dos tipos de revisiones: formales e informales. El equipo fue realizando revisiones informales de tanto el código como la documentación a lo largo del proyecto para asegurarse que cumpla con los estándares definidos. El tutor también participó en revisiones informales ya que fue recibiendo avances de la documentación y retornando comentarios y sugerencias.

Por otro lado, el equipo se juntó con expertos en los dominios para validar ciertos aspectos concretos del sistema, como por ejemplo el diseño inicial de la arquitectura con Gastón Mousqués.

Finalmente, la Software Factory de la Universidad ORT Uruguay brindó tres revisiones formales a lo largo del proyecto donde el equipo expuso en una presentación el trabajo realizado hasta el momento, y el revisor a cargo brindó retroalimentación y sugerencias. Los resultados de estas revisiones se pueden ver en el ANEXO 24 – “Conclusiones de revisiones formales de Universidad ORT Uruguay”.

6.4.6. Reuniones de coordinación

El equipo definió realizar reuniones de coordinación al finalizar cada *Milestone* del proyecto para verificar el avance hasta el momento. Los *Milestones* fueron de longitud variante, de entre dos y tres *Sprints*, comenzando con *Milestones* más cortos para mejor adaptación del equipo.

En las reuniones de coordinación se realizaron varias actividades. Estas fueron:

Recopilación de métricas

Se asegura que se hayan registrado las horas trabajadas durante las semanas del *Milestone* y la cantidad de *bugs* resueltos y arreglados, que el *Release Plan* hasta la fecha refleje el trabajo realizado, se mide el *carry-over* si lo hubo, etc.

Sprint Retrospective

Scrum es sobre mejora continua y adaptación, por lo que se utilizó el formato *START, STOP, CONTINUE* para identificar prácticas que fueron útiles, inútiles o nuevos hábitos a experimentar en el próximo *Milestone*.

Backlog Grooming

El equipo aprovechó estas instancias de reunión para organizar el tablero de Trello y replanificar el *Release Plan*.

Pruebas

Al finalizar los *Milestones* el equipo realizó pruebas documentadas de integración del sistema para detectar nuevos defectos, regresiones o posibles mejoras.

Análisis de riesgos

En las reuniones de coordinación también se hizo una reevaluación de los riesgos, creando una nueva versión del análisis de riesgos.

6.5. Métricas de calidad

"*Lo que no se mide, no se puede mejorar*". Esta frase, atribuida frecuentemente a Peter Drucker [31], es la razón por la que el equipo definió métricas de calidad sobre el proceso y el producto que permitan mejorar la toma de decisiones de manera constante y detectar errores que supongan no terminar cumpliendo con los objetivos.

El objetivo de esta sección es justificar las métricas que se utilizaron, describiendo los valores esperados, los obtenidos, el análisis y las decisiones tomadas a partir de ellas.

6.5.1. Estándar de registro de métricas

Las métricas se definieron con el siguiente formato:

Título	**Nombre de la métrica**
Objetivo	Por qué o para qué se mide
Unidad	Unidad a utilizar
Rango deseado	Rango donde se espera que se mantengan los valores
Cuándo se mide	Momentos en los que se debe medir. Números de *Sprint*
Valores medidos	Valores obtenidos durante las mediciones

Tabla 5 - Formato para la especificación de métricas

Las definiciones de las todas las métricas utilizadas según este formato se incluyen en el ANEXO 26 – "Métricas" junto con una más extensa justificación de por qué se consideró importante medirlas.

6.5.2. Métricas del producto

Las métricas del producto están estrictamente ligadas a la definición de calidad establecida anteriormente que incluye conceptos como la satisfacción de los usuarios con el producto final y la cantidad de defectos del mismo. Es por eso que las métricas del producto medidas fueron cantidad de defectos de cada categoría encontradas por *Sprint*, los resultados de las encuestas de satisfacción y resultados de análisis de las heurísticas de Nielsen.

Defectos reportados por Sprint

Se llevó un registro de defectos reportados por *Sprint* por diversos motivos, entre ellos:

1. Era necesario para seguir el criterio de arreglo de defectos especificado en el plan de gestión de incidentes.

2. Podría revelar falta de pruebas antes de liberar las funcionalidades.
3. La cantidad de defectos encontrados en un *Sprint*, unido a la cantidad de puntos desarrollados, podría sugerir al equipo ajustar la velocidad si esta daba lugar a código con muchos defectos.

Cabe destacar la definición de defecto utilizada por el equipo, quien consideró como tal a cualquier incidente, (no cambio en el criterio de aceptación), que sea reportado por cualquier persona que utilice la aplicación en los ambientes de desarrollo o producción, (no local). Esto significa que solo se consideran como defectos si fueron reportados una vez que la funcionalidad se encontró disponible a los clientes y que se dio como finalizada luego de su correspondiente *Sprint Review*. Por lo tanto si el equipo detectó y corrigió algún defecto antes de mostrarla a los clientes, no se consideró un defecto sino parte del proceso de desarrollo. Las métricas de reporte de incidentes permitirían visualizar si estas pruebas previas a liberar las funcionalidades fueron suficientes.

Por otro lado, en el ANEXO 20 – “Plan de gestión de incidentes” se explica por qué el equipo terminó no distinguiendo la cantidad máxima de defectos permitida entre los ambientes de Heroku (desarrollo y producción). En resumen, la distinción se eliminó durante el proyecto ya que el ambiente de producción no existió hasta la primera prueba y luego no se utilizó oficialmente por la empresa hasta el penúltimo *Sprint*, luego de la segunda y última prueba. Entonces el equipo decidió que no tenía sentido rastrear por separado la máxima cantidad de defectos permitidos, facilitando el seguimiento de defectos y la aplicación del plan.

A continuación se muestra la cantidad de defectos encontrados en los ambientes de Heroku, por *Sprint*. La primera gráfica muestra el total de defectos reportados y la segunda divididos según su prioridad.

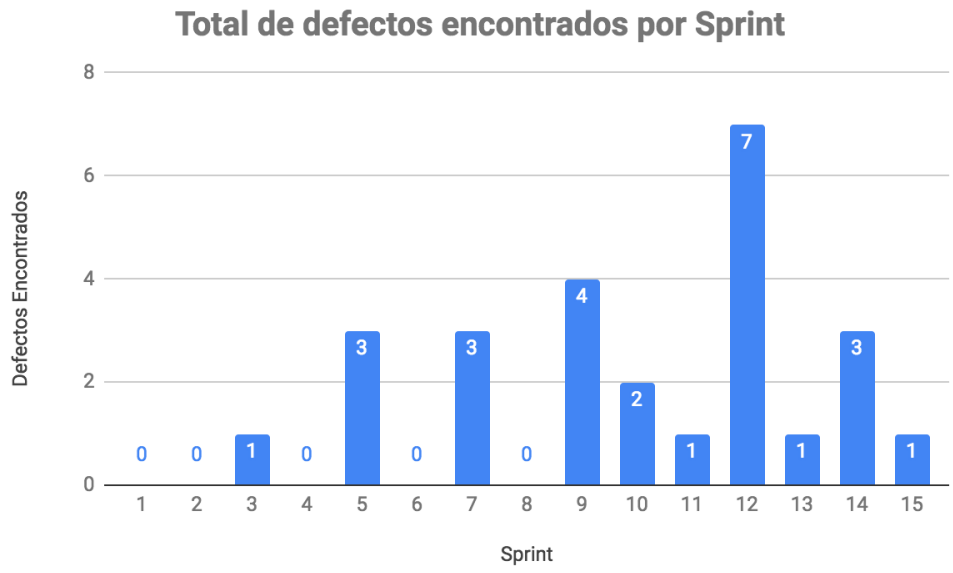


Ilustración 60 - Total de defectos encontrados por *Sprint*

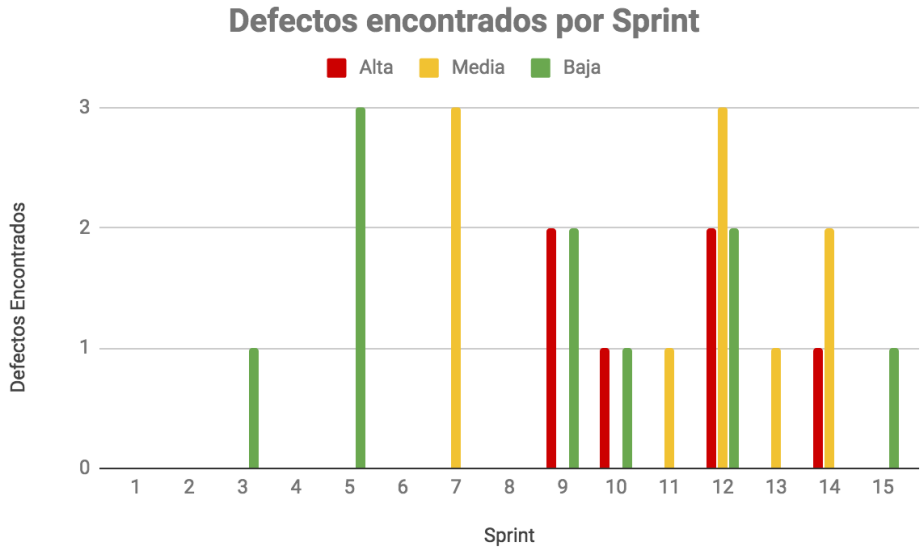


Ilustración 61 - Defectos encontrados por *Sprint*

En los primeros *Sprint*, donde la velocidad del equipo era menor, se reportaban menos defectos. Con el aumento de velocidad aumentaron también los defectos reportados. Como es esperable, luego de los *Sprints* con las funcionalidades más complejas se reportaron mayores cantidad de *bugs*.

La mayor cantidad de *bugs* se encuentran en los *Sprints* siguientes al término de cada *Milestone*, ya que en estas instancias el equipo hacía pruebas manuales de todo el sistema, detectando regresiones y nuevos defectos que no se habían detectado durante el desarrollo. Estos *Sprints* fueron el *3, 5, 9, 12 y 15*.

De la misma manera, en los *Sprints* posteriores a pruebas en repartos reales, como fueron el *Sprint* 9 y 15, se registraron todos los defectos que se reportaron durante las pruebas.

En el *Sprint* 9 se registraron defectos de prioridad alta encontrados durante la primera prueba en el *Sprint* 8. La mayoría de estos defectos estaban relacionados al funcionamiento de la aplicación en repartos con más de cien pedidos (debido a ineficiencias en el uso de la memoria), caso que el equipo no había probado exhaustivamente y que aparecieron durante la prueba. Luego de esto, el equipo decidió que debería comenzar a tomarse el tiempo de probar las funcionalidades internamente con más de 100 pedidos, para encontrar los defectos antes y no esperar a que aparezcan durante un reparto con los repartidores. A partir de eso los defectos de prioridad alta registrados fueron reportados por el equipo y no durante repartos lo cual se consideró una gran mejora.

El equipo incrementó la cantidad de esfuerzo en pruebas hacia el final de la etapa de desarrollo, antes de la prueba final, para asegurarse de no culminar la etapa de desarrollo con muchos defectos pendientes y cumplir así con el objetivo de un producto sin defectos críticos. Para el *Sprint* 15 ya no se reportaron casi defectos.

Se reportaron solo 6 defectos de prioridad alta del total de 96 historias sobre las que se trabajó. Esto fue logrado gracias a la alta cobertura de código mantenida en todo momento y a las meticulosas pruebas de integración realizadas antes de cada *Sprint Review,* que minimizaron los defectos encontrados luego de liberada la funcionalidad. Por otro lado, el equipo concluyó que las pruebas durante las reuniones de coordinación fueron de gran utilidad ya que otorgaron al equipo tiempo para realizar instancias formales de *testing* donde se examinaba todo el producto.

Resultado de las encuestas de satisfacción

Se realizaron dos instancias de encuestas, una al finalizar cada *Release*. Se separó la encuesta para los diferentes interesados: administrador y repartidores. Al finalizar el proyecto también se les pidió completar una encuesta a los clientes que evaluaba otros aspectos del trabajo del equipo. Las encuestas y sus resultados se pueden observar en el ANEXO 21 – “Encuestas de satisfacción”.

Las encuestas fueron diseñadas utilizando la escala de Likert [32], siendo 1 muy en desacuerdo y 5 muy de acuerdo con un enunciado positivo. Es por esto que se buscó como objetivo un promedio de todas las respuestas por arriba de 4 y ningún promedio en ningún enunciado en particular menor a 3.

A continuación se puede visualizar una gráfica que promedia los resultados.

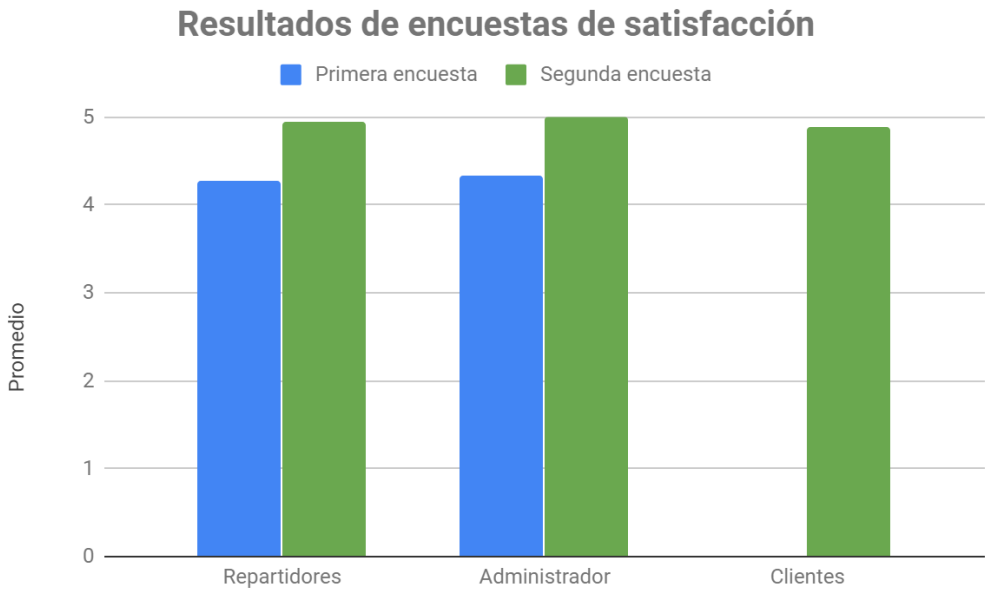


Ilustración 62 - Resultados de encuesta de satisfacción

Las encuestas alcanzan el valor promedio mínimo esperado, y en la segunda se aumenta el valor obtenido. Esto es porque el equipo tuvo en cuenta la retroalimentación de la primer instancia de pruebas en repartos reales, haciendo las modificaciones necesarias en la segunda lo que aumentó el grado de satisfacción.

Por ejemplo, una de las preguntas a los repartidores fue: “La nueva aplicación me eliminó la necesidad de manejar boletas en papel durante el reparto”, promediando 3 en la primera instancia. Luego de indagar el por qué, el equipo descubrió que era ya que en algunos pedidos se tenía que registrar un número de precinto que la aplicación no permitía en el momento. Entonces el equipo lo agregó como funcionalidad a la segunda versión aumentando el nivel de satisfacción.

No se visualiza en la gráfica pero también se consiguió el objetivo de ningún promedio menor a 3 en ninguna pregunta en concreto.

En conclusión, se consiguió el nivel de satisfacción esperado en la última instancia de encuestas que el equipo se propuso como objetivo, el cual era un promedio por arriba de 4 y ningún resultado menor a 3. Esto se debió al constante intercambio y retroalimentación que el equipo buscó conseguir de los interesados para incrementar progresivamente su satisfacción con el producto.

Heurísticas de Nielsen

Se realizó un análisis heurístico de tanto la aplicación móvil como la web para el administrador, en el cual se analizaron las pantallas y se detectaron problemas con diferentes niveles de severidad. El análisis completo se encuentra en el ANEXO 27 – “Análisis heurístico”. El análisis reveló alrededor de 12 problemas de usabilidad en cada aplicación, ninguna de severidad crítica, que fueron corregidas.

Se concluye que las heurísticas se estaban respetando debido a las actividades realizadas previamente al desarrollo para asegurar la usabilidad del sistema, como fueron el delegar los diseños a una diseñadora gráfica, utilizar componentes nativos para la aplicación móvil y un *Framework* para la interfaz del administrador. De todos modos, el análisis permitió realizar algunas mejoras.

6.5.3. Métricas del proceso

Las métricas del proceso se monitorearon para controlar la ejecución de los procesos utilizados durante el proyecto, permitiendo evaluar y mejorar su calidad *Sprint* a *Sprint*.

Dentro de las métricas del proceso se incluyen aquellas de gestión que se pueden visualizar junto con su análisis en la sección 7.4 - “Métricas de gestión”, como son las

que comparan los puntos realizados versus los planificados, las horas por semana y por área, etc.

Por otro lado, se consideraron ciertas métricas sobre defectos como métricas del proceso. Se consideró que para lograr los objetivos de calidad propuestos, el proceso debe hacer posible el seguimiento de los planes que se definieron, entre ellos, el de gestión de incidentes, que define un criterio para cuándo se debe arreglar un defecto antes de superar la cantidad máxima de defectos permitidos. Es por ello que parte de la calidad del proceso está dada por la medida en la que favorece el arreglo de *bugs*.

Defectos corregidos por *Sprint*

Además de la gráfica de la cantidad de defectos reportados por *Sprint*, es importante poder visualizar cómo se distribuyó el arreglo de defectos a lo largo del proyecto por las razones mencionadas anteriormente.

A continuación se visualiza la cantidad de defectos de cada prioridad corregidos en cada *Sprint*.

Ilustración 63 - Defectos resueltos por *Sprint*

El arreglo de *bugs* se concentró luego de *Sprints* donde se realizaron pruebas, ya sea durante las pruebas del equipo al finalizar un *Milestone*, o durante repartos de prueba, ya que en estas instancias se reportaron más defectos.

Algunos defectos fueron encontrados por el equipo durante un *Sprint* en progreso. En esos casos, siempre y cuando el equipo tuviese tiempo libre al finalizar todas las tareas del *Sprint* actual, procedía a arreglarlos si la prioridad del defecto lo ameritaba.

El *Sprint* con mayor cantidad de arreglo de *bugs* fue el último, en el cual no se planificó ninguna funcionalidad sino que se dejó vacío para realizar las últimas pruebas y corregir defectos y sugerencias finales. En este *Sprint* se solucionaron todos los defectos de prioridad baja que se fueron postergando de los *Sprints* anteriores.

En conclusión, el arreglo de defectos se distribuyó a lo largo del proceso. La flexibilidad de un *Release Plan* que no estaba fijo sino que se fue adaptando *Sprint a Sprin*t permitió planificar tarjetas de arreglo de defectos como parte de los *Sprints*, fomentando un producto sin defectos en todo momento.

Defectos acumulados sin corregir por *Sprint*

De los datos utilizados para las gráficas de defectos descubiertos y resueltos por *Sprint* se pudo generar la gráfica de defectos acumulados de cada categoría que se puede observar a continuación.

Ilustración 64 - Gráfica de defectos acumulados sin resolver por *Sprint*

En la gráfica se puede visualizar como nunca se sobrepasó la máxima cantidad de defectos de cada categoría permitidos especificada en el plan de incidentes durante dos *Sprints* seguidos. Esto es porque en los momentos donde se alcanzaba esta cantidad se planeó inmediatamente un *Sprint* con mayor cantidad de arreglo de *bugs*, que disminuyó la cantidad de defectos acumulada hasta el momento.

Los defectos con prioridad alta se intentaron solucionar en el mismo *Sprint* en el que fueron reportados, o sino en el siguiente, haciendo que la acumulación de defectos de prioridad alta casi no suceda. El *Sprint* 9 fue el único *Sprint* donde se reportaron defectos de prioridad alta que no se solucionaron en el mismo *Sprint* porque el equipo ya estaba comprometido con otras funcionalidades y no tuvo tiempo para arreglarlos hasta el *Sprint* siguiente.

Se puede notar cierta acumulación de defectos de prioridad baja que sin embargo siempre estuvo controlada dentro de los límites permitidos. Esto demuestra que el equipo supo priorizar correctamente el arreglo de defectos, prestando inmediata atención a aquellos con mayor prioridad, y arreglando en la medida que fuese necesario los de baja prioridad con el objetivo de que no se acumulen.

En conclusión, el equipo respetó el plan de gestión de incidentes para el arreglo de defectos, lo que resultó en un producto sin defectos acumulados. El proceso permitió balancear el arreglo de defectos con nuevos requerimientos, permitiendo que en todo *Sprint* se agregue valor sin perder calidad.

6.6. Gestión de la configuración

Esta sección tiene como objetivo presentar el plan de gestión de la configuración.

La gestión de la configuración es el conjunto de procesos destinados a asegurar la calidad del producto a través del control de cambios y de la disponibilidad constante de una versión estable de cada elemento de la configuración del software (ECS).

Identificación de los elementos de configuración

Antes de comenzar el equipo identificó aquellos elementos que entrarían en el plan, esto es, aquellos que deban estar versionados o susceptibles a la introducción de cambios que deban gestionarse.

Tipo	Elementos
Desarrollo de Software	Código Fuente: • geolocalizador-api (Ruby on Rails) • geolocalizador-android (Android)
Documentación	• Notas de reuniones y entrevistas • Prototipos y resultados • Estándares de documentación y codificación • Informes de revisión y avance • Documentación de ceremonias de Scrum • Product Backlog

	• Plan de calidad • Plan de gestión y análisis de riesgos • Análisis de riesgos • Documentación de arquitectura • Bibliografía

Tabla 6 - Elementos de configuración

6.6.1. Elección de herramientas

A continuación se describen las herramientas seleccionadas para gestionar los elementos de configuración durante el proyecto.

Desarrollo de Software

Al momento de elegir la herramienta para la gestión del software se consideraron las dos principales arquitecturas: centralizada y distribuida. La arquitectura tradicional es la centralizada, donde un solo servidor remoto contiene las versiones de todos los archivos. Los clientes toman los archivos del servidor, los editan y los vuelven a subir. En cambio, en los sistemas distribuidos, cada cliente tiene un repositorio completamente funcional en el cual trabajar que puede ser combinado con el repositorio en el servidor remoto.

Esto trae consigo ventajas que hizo que el equipo se decidiera por una arquitectura distribuida como:

- Mayor autonomía y rapidez dada por una menor necesidad de conexión para operar
- Fácil colaboración entre los miembros del equipo
- Operaciones rápidas y sencillas para la integración de código
- Resolución de conflictos de forma sencilla
- Eliminación del servidor como la única copia de los archivos
- Posibilidad de trabajar privadamente (útil para probar cosas)

Por las ventajas de esta arquitectura y la experiencia previa de ambos miembros del equipo con la herramienta, se decidió utilizar la herramienta Git. Además, se escogió Github como plataforma para alojar el proyecto. Para ver más detalles de estas elecciones ver ANEXO 28 – “Análisis de herramientas de repositorio”.

Documentación

El equipo seleccionó Google Drive para el mantenimiento de la documentación principalmente por su carácter colaborativo y posibilidad de versionado e historial.

Para la recopilación final se utilizó Microsoft Word ya que era necesario para cumplir con todas las pautas de formato requeridas por la Universidad ORT Uruguay.

6.6.2. Organización del repositorio

Desarrollo de Software

El software se alojó en dos repositorios de Git privados:

- **geolocalizador-api:** Repositorio utilizado para el desarrollo del back-end incluyendo la lógica de negocio y la aplicación web del administrador.
- **geolocalizador-android:** Repositorio utilizado para el desarrollo del front-end Android de la aplicación utilizada por los repartidores.

En la siguiente ilustración se observan los repositorios en Github.

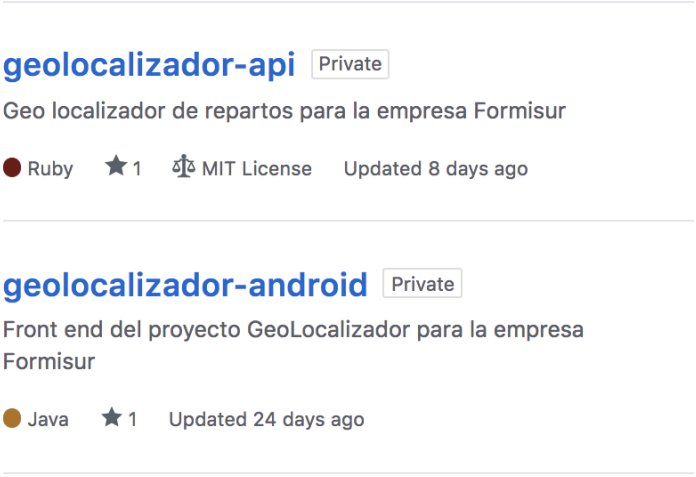


Ilustración 65 - Repositorios del código fuente

Documentación

El repositorio se organizó en diferentes carpetas con subdivisiones intuitivas de forma de facilitar la búsqueda de un determinado archivo. A continuación se presentan las carpetas del primer nivel.

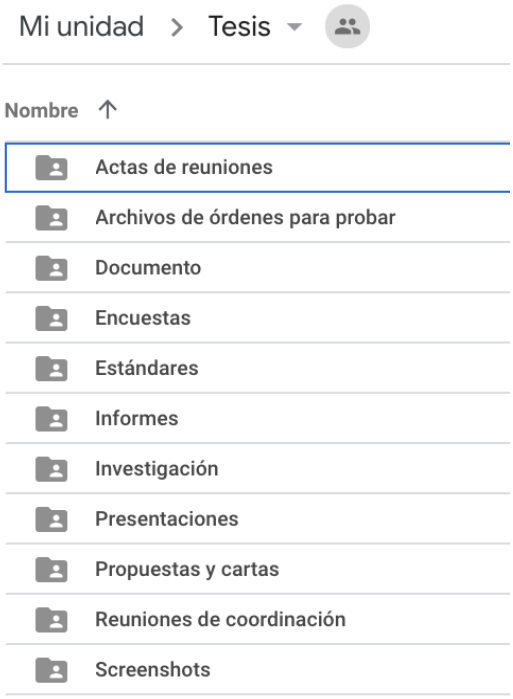


Ilustración 66 - Ejemplo de divisiones de carpetas para la documentación

La carpeta Investigación agrupó los documentos de investigación realizados especialmente para el problema del enrutamiento.

En la carpeta Documento se encontró la mayor parte del contenido ya que ahí se almacenaron los capítulos, los planes y otras partes de secciones que finalmente se integraron para conformar el documento final. Se organizó en dos principales subcarpetas. El primer nivel separó según el estado de los documentos: En proceso, A revisar y Revisadas por el tutor. El tutor tuvo acceso a las últimas dos y era notificado a medida que nuevos planes, anexos o capítulos se encontraban disponibles. El equipo movía los documentos de En Proceso a A revisar cuando consideraba que estaban listos y luego a Revisadas una vez que el tutor las revisó.

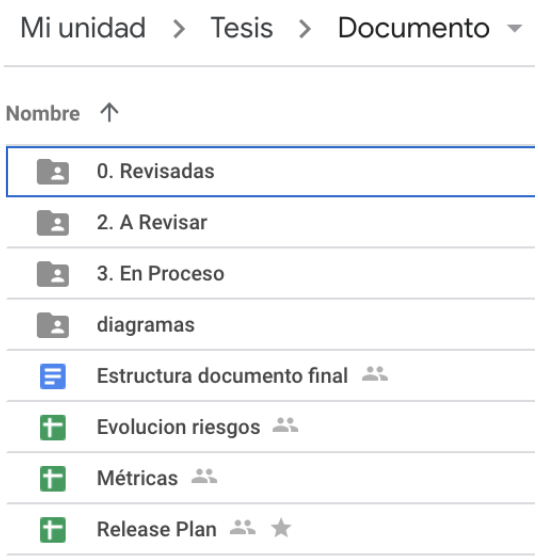


Ilustración 67 - Ejemplo de carpetas que marcan el estado de los documentos

Documentos sobre los que se trabajó constantemente como son las hojas de Excel con las métricas obtenidas a lo largo de todo el proceso, la evolución de los riesgos y el *Release Plan* se colocaron fuera de las carpetas de estado.

Durante la fase inicial, la carpeta Revisadas se subdivisión según áreas que agrupaban los planes definidos.

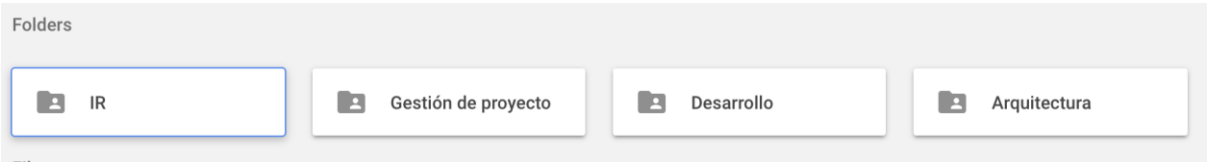


Ilustración 68 - Ejemplo de la subdivisión de los planes

Arquitectura: Almacena todo lo relacionado a la arquitectura del sistema, incluyendo diagramas y documentos sobre la misma.

Desarrollo: Almacena justificaciones de tecnologías utilizadas en el desarrollo como herramientas de hosting, de almacenamiento, servicios externos, el plan de testing, etc.

Gestión: Almacena los documentos sobre el plan de proyecto, la metodología, reglamentos, cronograma, plan de calidad, plan de gestión de la configuración y plan de gestión de riesgos.

IR: Almacena los documentos relacionados con el análisis del problema y solución, la especificación de requerimientos, el plan de prototipos, etc.

Durante la etapa final, la carpeta Revisadas contuvo los capítulos ya integrados y revisados por lo que la subdivisión explicada anteriormente se eliminó.

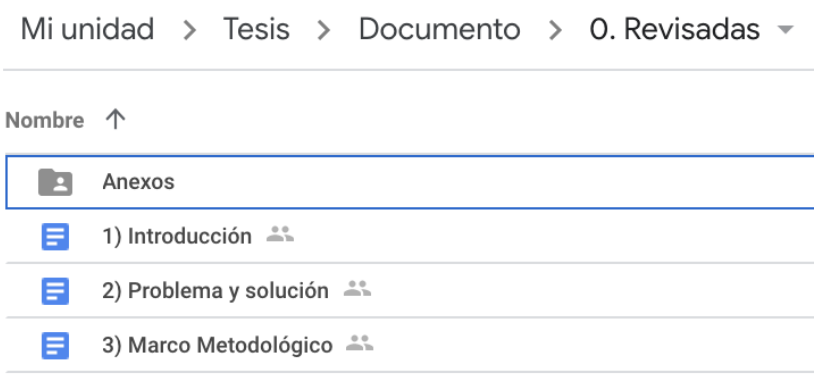


Ilustración 69 - Ejemplo del estado final de la carpeta “Revisadas”

6.6.3. Política de *branching*

Desarrollo de Software

Se trabajó bajo el modelo GitFlow [33] para el control de versiones del código fuente. GitFlow permitió al equipo desarrollar en paralelo de una manera más fácil separando el código estable del que estaba en desarrollo al separar en *branches* por *features* o *bugs*.

El modelo define la utilización de las siguientes ramas:

master: Almacena todo el historial de liberaciones a producción.

develop: Rama sobre la cual se trabaja localmente y en la cual se integran todas las funcionalidades, *bugs* y *hotfixes* del desarrollo.

***Feature branches*:** Al comenzar a trabajar en una nueva funcionalidad, se crea una rama desde la rama *develop*. Al terminarla, se crea un *Pull Request* y al ser aprobado se unen los *commits* de la rama en uno solo que se integra a la rama *develop*.

***hotfix branches*:** Son creadas exclusivamente para los casos en los que hay un error en producción (error en la rama *master*) y se quiere subir su arreglo sin tener que subir las nuevas funcionalidades en la rama *develop* que quizás no han sido probadas. De esta manera cuando se encuentra un error en *master*, se crea una rama de *hotfix* desde *master*, se integra luego de aprobada, se sube master nuevamente arreglando el error, se integra la rama de *hotfix* a la rama *develop* y se continúa con el curso normal.

Ilustración 70 - Ejemplo GitFlow

En la infraestructura definida, el servidor de producción siempre está basado en la rama *master*, mientras que el servidor de desarrollo se sincroniza con la rama *develop*.

6.6.4. Gestión de versiones

Desarrollo de Software

Como se mencionó, los ambientes de producción estaban centrados en la rama *master* y el ambiente de desarrollo en *develop*. Para identificar las versiones del ambiente de desarrollo del de producción se utilizó la funcionalidad de etiquetas que provee Git.

Para definir las versiones del software se utilizó la numeración internacional para el versionado de la semántica: **Major.Minor.Patch**.

Dígito	Descripción
Major	Incrementa cuando existe un cambio significativo y que puede hacer incompatible previas versiones de la API o de la app.
Minor	Incrementa cuando se agregan funcionalidades de manera que el sistema siga siendo compatibles con antiguas versiones.
Patch	Incrementa cuando se hacen correcciones de manera que el sistema siga siendo compatibles con antiguas versiones.

Tabla 7 - Numeración para el versionado del software

Se creó un *Release* para la primera prueba (v0.0.0), un *Release* para la segunda, (v1.0.0), y luego un *Release* para cada vez que se subió un conjunto nuevo de mejoras o arreglos de defectos a producción.

A continuación se muestra la utilización de las etiquetas de Git para el versionado. La siguiente imagen fue tomada al finalizar la segunda prueba con los repartidores. Luego de esta fecha se realizaron algunos otros *Releases*.

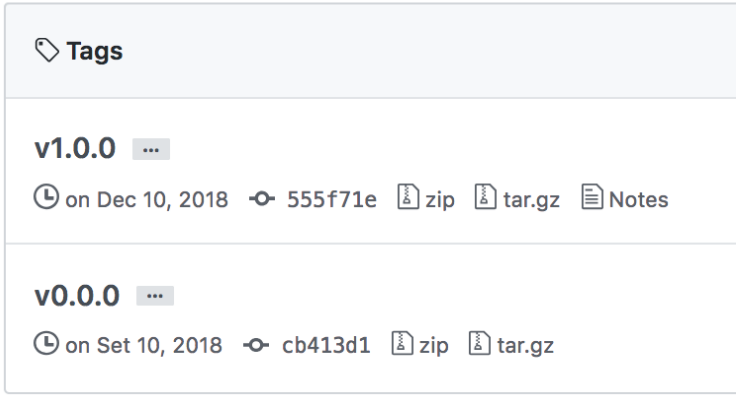


Ilustración 71 - Ejemplo de *Releases* durante el proyecto

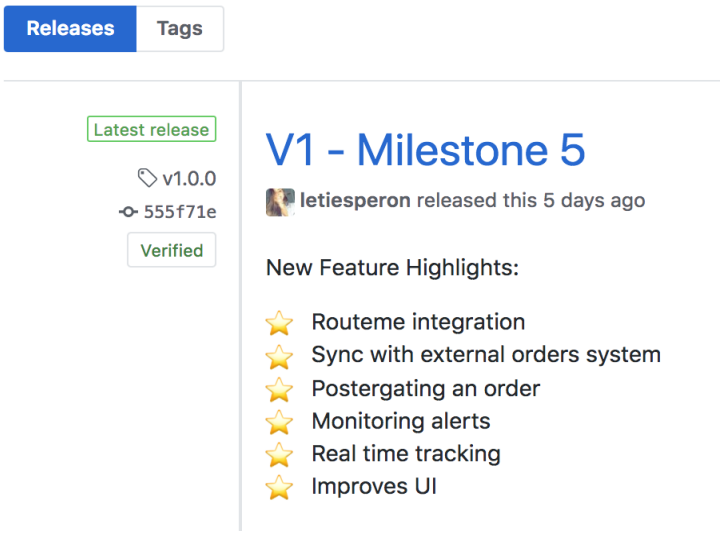


Ilustración 72 - Ejemplo de especificación de un *Release* en Github

Documentación

Se consideró innecesario el uso de una numeración para el versionado de los documentos en el *Google Drive*. Cuando fue necesario se recurrió al historial de *Google Drive* para la visualización de cambios. Los elementos de la documentación que sí se versionaron fueron principalmente las revisiones del documento final y los análisis de riesgos.

6.6.5. Control de cambios

El equipo definió un plan de gestión de cambios para actuar cuando se enfrenta a una solicitud de cambio de los requerimientos, esto es, una sugerencia para cambiar o agregar uno o más requerimientos ya sea su funcionalidad en sí, diseño o flujo. No se consideraron los *bugs* como cambios para los cuales existe la gestión de incidentes.

Por otro lado, los cambios menores sobre funcionalidades demostradas en la *Sprint Review* pueden considerarse *feedback* y ser tratados de forma distinta que los cambios. El equipo evaluó en la misma reunión si se trataba de un *feedback* (si es algo muy rápido de corregir) en cuyo caso se anotaba en Trello en la misma historia de usuario, o si se trata a de un cambio que modificaba la funcionalidad, el diseño o el flujo en mayor medida.

Se utilizó Trello para realizar el seguimiento de las solicitudes de cambio. Para ello se creó una columna *Icebox* para contener las solicitudes que aún no han sido agregadas definitivamente en el alcance estimado del producto.

El diagrama a continuación demuestra el proceso que siguen las solicitudes de cambio.

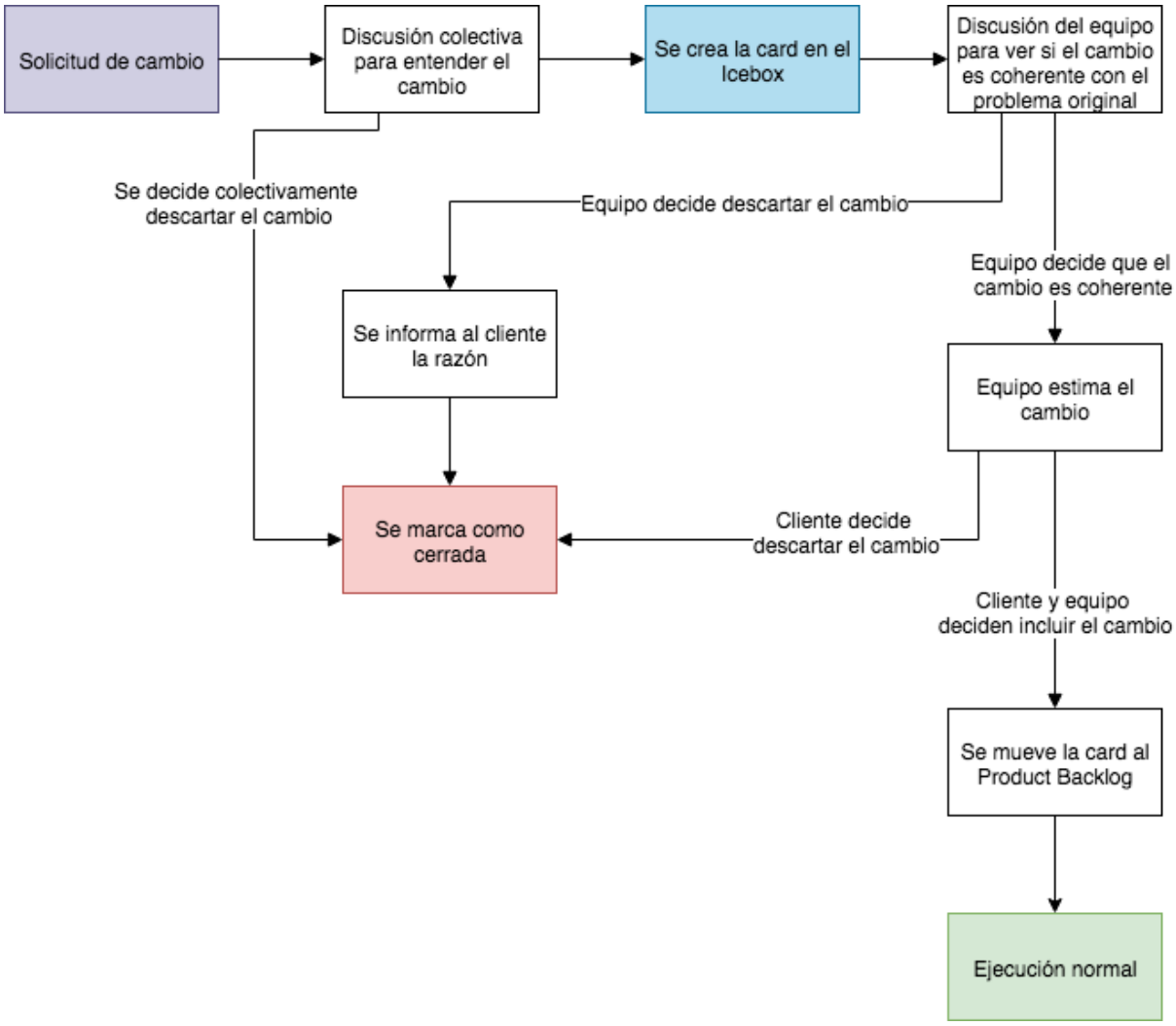


Ilustración 73 - Diagrama de gestión de solicitudes cambios

Como se puede visualizar, cuando hay una solicitud de cambio se agrega como historia de usuario en el *Icebox* como abierta, de forma de documentar la solicitud.

Cuando el equipo la discute, evalúan si es coherente con el problema original a resolver, determinando si se cierra en el momento o si queda abierta en la columna *Icebox* para resolver después.

Se estima y, si en una re priorización del producto se decide que dada su prioridad entra en el alcance, se agrega al *Product Backlog* siguiendo desde allí el flujo común de todas las funcionalidades.

A continuación se muestra una captura del Trello, para visualizar la utilización de la columna *Icebox*.

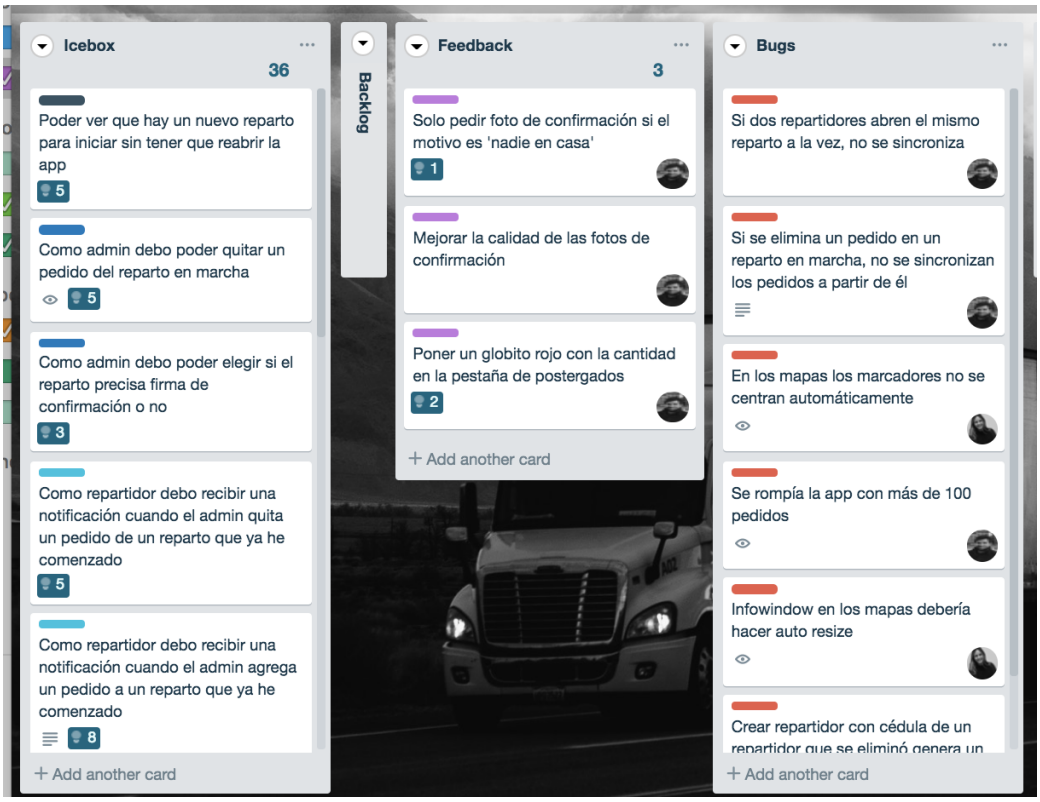


Ilustración 74 - Ejemplo del Icebox

La gestión de cambios especificada fue correctamente seguida durante todo el proceso, y fue particularmente importante dada la gran cantidad de ideas que surgían de todas las reuniones. De hecho, el *Product Backlog* final incorporó varias solicitudes de cambios que no estaban planificadas en el *Product Backlog* inicial.

7. Gestión de proyecto

En este capítulo se describen los procesos de gestión utilizados a lo largo del proyecto, mostrando las tareas, planes, estimaciones, métricas y análisis realizados.

7.1. Adaptación del marco de trabajo

Como se ha mencionado anteriormente, se utilizó la metodología Scrum como marco de trabajo durante la fase de desarrollo pero se realizaron algunas adaptaciones. El siguiente diagrama muestra un ciclo completo de esta metodología.

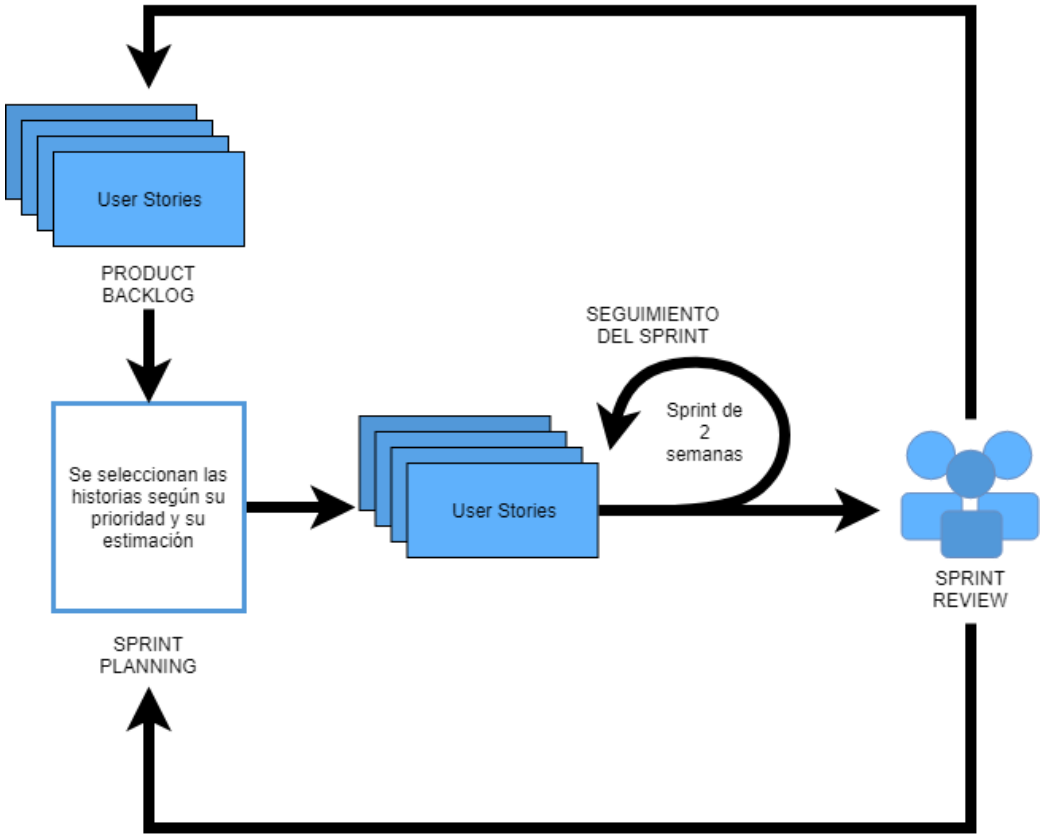


Ilustración 75 - Marco de trabajo

A continuación se describe la instanciación de Scrum en el proyecto.

Product Owner

Se designó como *Product Owner* a Martín García por ser uno de los expertos en el dominio que acompañó y guió al equipo durante todo el proyecto. Fue con quien más se validaron los requerimientos y la priorización de los mismos.

Scrum Master

El *Scrum Master* es el encargado de que se aplique la metodología correctamente, además de guiar al equipo hacia la auto organización y multifuncionalidad. Dada la experiencia previa, se designó a Leticia para ocupar este rol pero en la práctica fue compartido.

Product Backlog

Se denominó *Product Backlog* a la lista priorizada de los requerimientos escritos en forma de historias de usuario. El *Product Backlog* no fue estático sino que evolucionó a lo largo de todo el proyecto. Las principales historias de usuario del sistema se desarrollaron según el plan pero las funcionalidades complementarias se modificaron, eliminaron y agregaron nuevas. Las funcionalidades del *Product Backlog* implementadas para el final del proyecto se encuentran en el ANEXO 9 – “Product Backlog”.

Estimación

La estimación de historias se realizó a través de *Story Points* o puntos de esfuerzo por las razones mencionadas en la sección 4.4 - “Estimación”.

Sprint Planning

Antes de comenzar un *Sprint* se realizaba una *Sprint Planning* con la intención de organizar el trabajo del mismo. La participación de los clientes durante esta reunión era esencial ya que ellos eran quienes confirmaban la prioridad de las funcionalidades a implementar y opinaban sobre cómo debería ser su funcionamiento.

Previa a la *Sprint Planning*, el equipo ya había realizado un *Backlog Grooming* considerando las prioridades de los clientes, donde las historias de usuario ya fueron priorizadas y las tareas a realizar en el *Sprint* ya fueron pre planificadas y su criterio de aceptación escrito. De esta manera se agilizaban las *Sprint Plannings* con los clientes ya que el equipo no iba en blanco a las reuniones sino que ya tenía una idea y simplemente se validaba y se discutían los detalles.

Sprint Review

La *Sprint Review* tuvo como objetivo verificar el incremento creado durante el *Sprint*, mostrándolo a los clientes para obtener retroalimentación. Para medir la usabilidad, muchas veces se les pidió a los clientes que intentaran llevar a cabo la tarea sin haber visto previamente el sistema más que en los diseños en la *Sprint Planning* anterior. De la *Sprint Review* el equipo se llevaba consigo una lista de *feedback* o sugerencias de cambio a agregar al *Icebox*.

La *Sprint Planning* y la *Sprint Review* se realizaban en el mismo día, donde luego de finalizada la *Sprint Review* se comenzaba a planificar el próximo *Sprint*. Luego de estas reuniones, el equipo se quedaba un tiempo más a redactar el acta, documentar las métricas obtenidas durante el *Sprint* y organizarse para comenzar a desarrollar las funcionalidades.

Sprint Retrospective

Como se fue mencionando anteriormente, las retrospectivas de Scrum se realizaron formalmente durante las reuniones de coordinación al finalizar cada uno de los cinco *Milestones* del proyecto. Se utilizó el formato *START, STOP, CONTINUE* para identificar prácticas que fueron útiles, inútiles o nuevos hábitos a experimentar en el próximo *Milestone*. Se pueden visualizar ejemplos en el ANEXO 22 – “Reuniones de coordinación”.

7.2. Planificación de *Milestones* y *Releases*

El equipo realizó un *Release Plan* para proyectar el trabajo a realizar a lo largo del proyecto. El *Release Plan* permitió a tanto el equipo como a los clientes tener mayor visibilidad de las fechas estimadas para el desarrollo de las funcionalidades para permitir la planificación de las pruebas en repartos, la definición del alcance final y la planificación en las *Sprint Plannings*.

El *Release Plan* se realizó colocando las funcionalidades según se habían priorizado en orden, y se especificaba en qué *Sprint* se desarrollarían con ayuda de la estimación de la velocidad del equipo. Fue evolucionando constantemente a lo largo de todo el proyecto, especialmente durante las reuniones de coordinación donde el equipo rearmaba el plan según su nueva velocidad, las prioridades del cliente, etc.

En el *Release Plan* se agruparon los *Sprints* en *Milestones*. Los *Milestones* agrupaban objetivos del incremento del producto. Se definieron cinco *Milestones* con duración de entre dos y cuatro *Sprints*, comenzando con *Milestones* más cortos para mejor adaptación del equipo ya que al finalizar cada *Milestone* el equipo realizaba una reunión de coordinación con varias actividades de gestión e introspectiva.

Los *Milestones* del proyecto fueron:

1. Administración de repartidores
2. Administración de repartos
3. Viaje del repartidor
4. Monitoreo de los repartos
5. Funcionalidades complementarias

A continuación se presenta una captura de pantalla del *Release Plan,* donde se pueden visualizar los primos *Milestones* y la asignación de las funcionalidades de sus *Sprints*.

		Sprint:	SPRINT 1		SPRINT 2		SPRINT 3		
		FUNCIONALIDAD / SEMANA	S1	S2	S3	S4	S5	S6	S7
Puntos estimados	Velocidad		28/05 - 04/06	04/06 - 11/06	11/06 - 18/06	18/06 - 25/06	25/06 - 02/07	02/07 - 09/07	09/07 -
		Milestone 1: Administración de repartidores							
3		Setup repositorio Ruby		X					
3		Setup repositorio Andoird		X					
3		Setup Heroku		X					
3	12	Setup Google Play Store o equivalente		X					
3		Ícono de la app y landing			X				
1		Como admin debo poder loguearme a la web con usuario y contraseña			X				
2		Como admin debo poder crear usuarios de la app para los repartidores			X				
1		Como admin debo poder filtrar los repartidores			X				
1		Como admin debo poder ver repartidores			X				
3		Como repartidor debo poder loguearme			X				
2	13	Como repartido debo poder cerrar sesión			X				
		Milestone 2: Administración de pedidos y repartos							
8		Como admin debo poder crear un reparto					X		
1		Como admin debo poder ver la lista de repartos					X		
1		Como admin debo poder ver los datos de un reparto					X		
1		Como admin debo poder ver los datos de un pedido					X		
1		Como admin debo poder ver la lista de pedidos					X		
1		Como admin debo poder filtrar repartos					X		
1	14	Como admin debo poder filtrar pedidos					X		
5		Como admin debo poder administrar direcciones de destinatarios							
1		Como admin debo poder filtrar las correcciones de direcciones							
3		Como admin debo poder borrar y recuperar un reparto							
5	14	El reparto debe crearse con las direcciones de destinatarios arregladas							
		Milestone 3: Viaje del repartidor							
8		Como repartidor debo poder iniciar el próximo reparto							
3		Como repartidor en un reparto debo poder ver la próxima dirección en mi pedido							
5		Como repartidor debo poder navegar hacia la próxima dirección							

Ilustración 76 - Release Plan

Se definió realizar dos *Releases* del producto a lo largo del proyecto. Se priorizaron las tareas de forma de que el primer *Release* incluyera una versión mínima utilizable del producto. Esto se consiguió a partir del *Milestone* 3. La buena priorización de las tareas permitió que el primer *Release* estuviese listo para el 10 de Setiembre, permitiendo validar el producto construido muy temprano en el proyecto.

La siguiente imagen tiene como objetivo reflejar la duración proporcional de cada *Milestone* y las fechas donde se completaron los *Releases*.

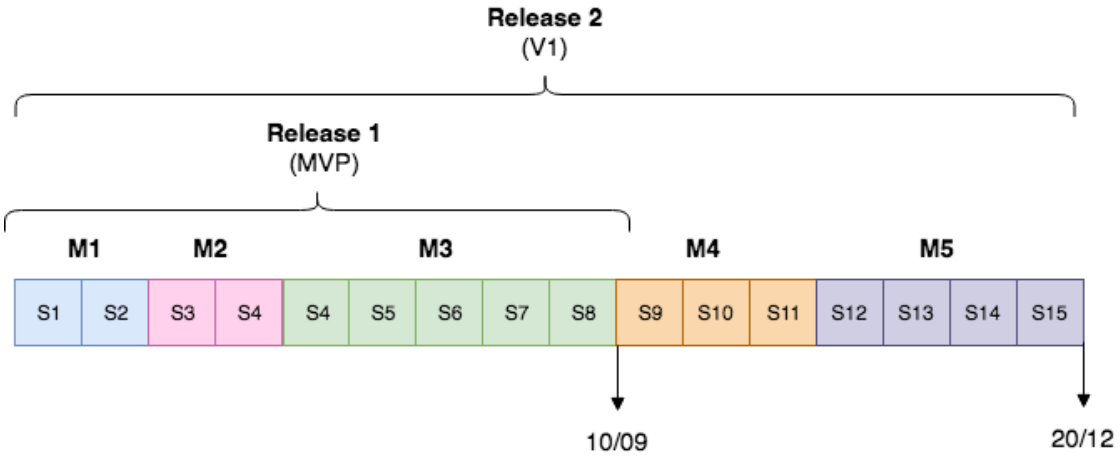


Ilustración 77 - Duración y fechas de los Milestones

El detalle del desarrollo de los *Sprints* se puede visualizar en el ANEXO 30 – “Release Plan”.

7.3. Seguimiento de las tareas

Como se fue mencionando, se utilizó Trello para almacenar la lista de requerimientos, defectos y *feedback* y realizar el seguimiento del estado de los mismos. Para visualizar las razones por la que se eligió Trello referirse al ANEXO 29 – “Análisis de herramientas de gestión”.

Se crearon diferentes columnas en Trello que fueron *Icebox*, *Product Backlog*, *Bugs*, *Feedback*, *Sprint Backlog*, En proceso, Terminadas y Aprobadas. A continuación se muestra una captura del tablero de Trello. Para permitir que entren en la pantalla algunas de las columnas fueron contraídas.

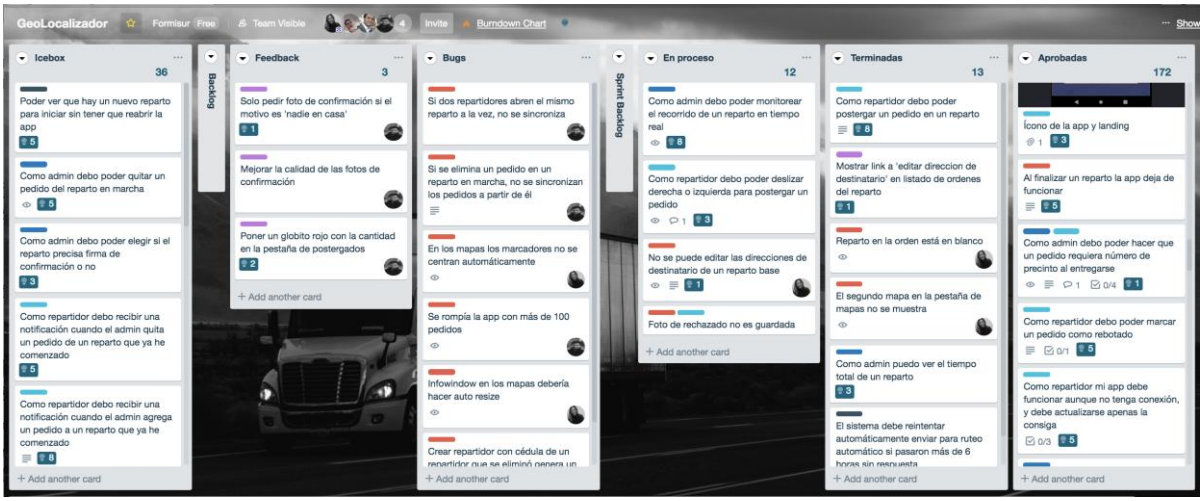


Tabla 8 - Seguimiento de tareas en Trello

Para visualizar una descripción completa de cómo se utilizaron las distintas columnas de Trello referirse a ANEXO 25 – “Utilización del Trello para seguimiento de tareas”

Sprint Burndown Chart

Durante el proyecto se fue realizando un *Sprint Burndown Chart* para visualizar cómo disminuía la cantidad de trabajo restante a medida que se avanzaba en los *Sprints*. Esto permitió estimar si la velocidad del equipo proyectaba ser suficiente para cubrir la versión del *Product Backlog* del momento.

A continuación se muestra el *Sprint Burndown Chart* considerando el tamaño final del proyecto que fue mayor al planificado inicialmente.

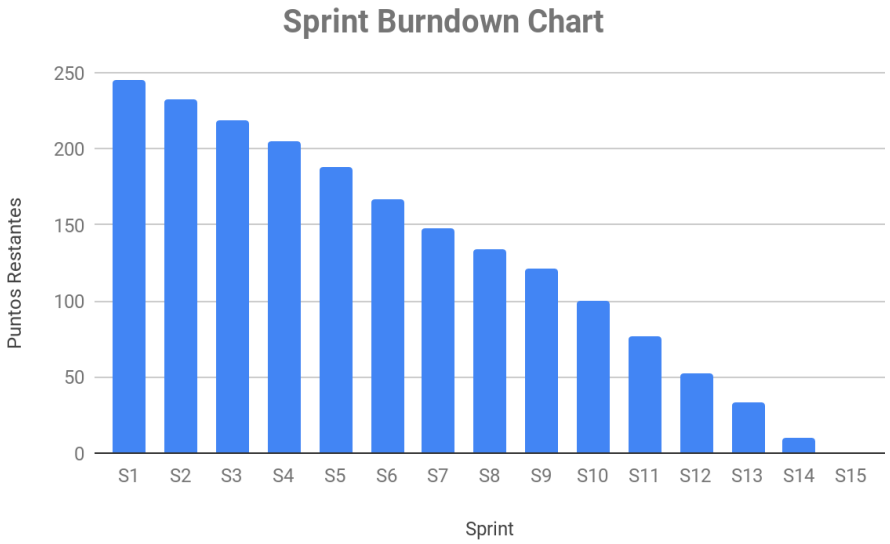


Ilustración 78 - *Sprint Burndown Chart*

7.4. Métricas de gestión

Las métricas de gestión permitieron al equipo evaluar y adaptar el proceso para lograr cumplir con los objetivos del proceso que se propuso. En esta sección se muestran y analizan las principales métricas de gestión controladas durante el proyecto que fueron permitiendo tomar decisiones para maximizar la eficiencia del proceso sobre el que se trabajó.

7.4.1. Velocidad del equipo

La velocidad del equipo fue una métrica importante por varias razones. Principalmente permitió estimar la cantidad de puntos que el equipo puede tomar en una semana, mejorando la planeación de trabajo en las *Sprints Plannings* ya que disminuía la probabilidad de tener *carry-overs* para el siguiente *Sprint*. Por la misma razón, poder estimar la velocidad del equipo facilitó la planeación a largo plazo del *Release Plan*, ya que el mismo contemplaba la velocidad del equipo para distribuir las funcionalidades entre los *Sprints* y recortar el alcance final.

A continuación se muestra la gráfica de la evolución de los puntos realizados por *Sprint*. También se incluye la que compara estos puntos realizados con los puntos planificados para analizarlas en conjunto.

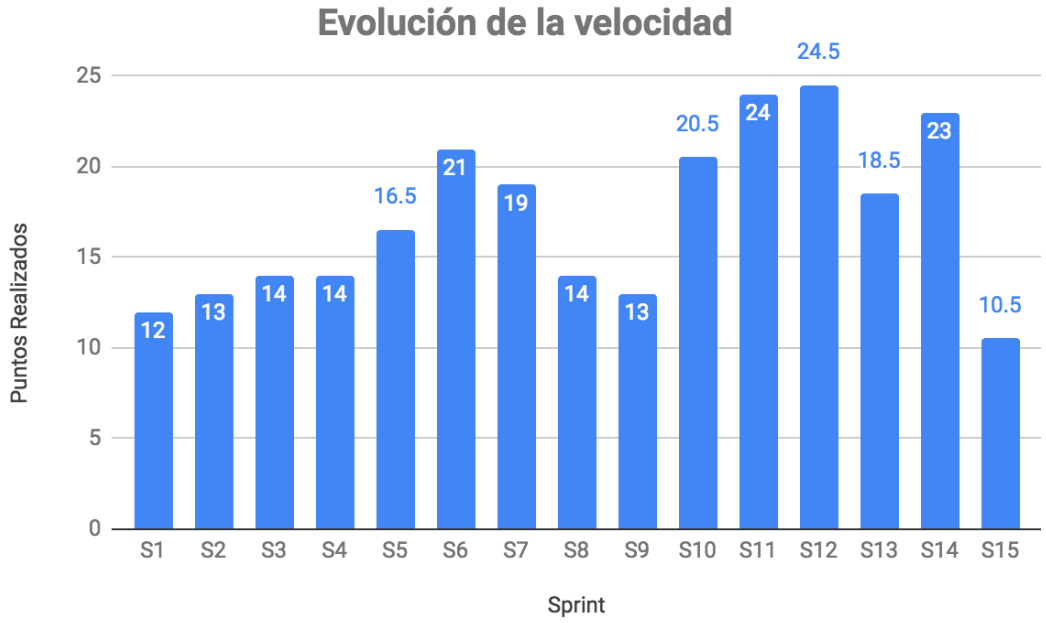


Ilustración 79 - Seguimiento de la velocidad

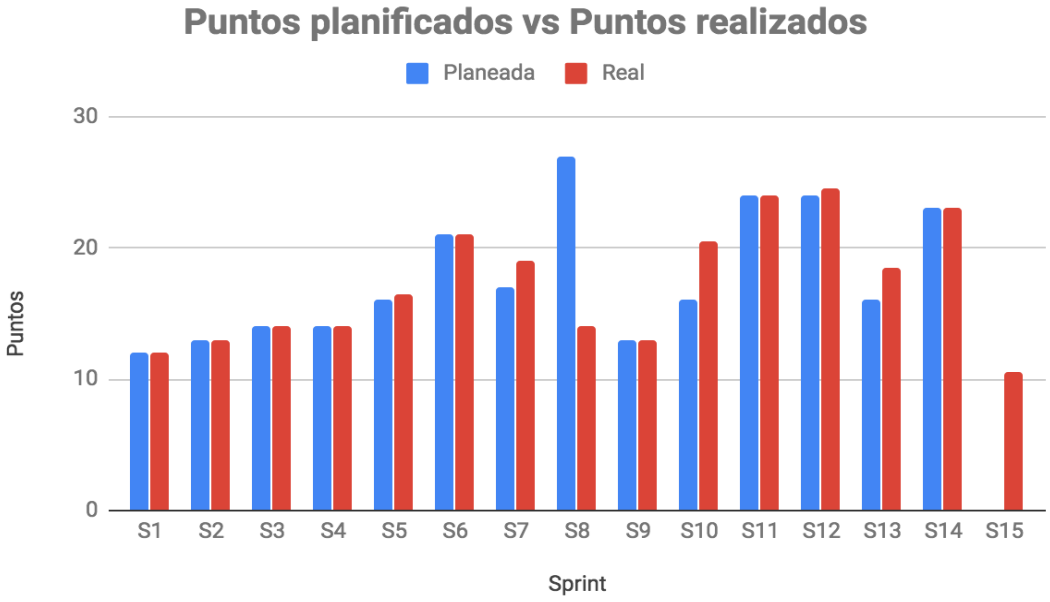


Ilustración 80 - Puntos planificados vs puntos realizados por *Sprint*

Los integrantes decidieron comenzar realizando una cantidad segura de puntos por *Sprint* e ir incrementándola a medida que fuesen ganando confianza en las tecnologías y en el ritmo de trabajo, como se especificó en la estrategia para evitar el riesgo de atraso por sobre estimación de la velocidad del equipo en el ANEXO 32 – “Listado de riesgos y planes”.

Es por eso que se comenzó con 12 puntos en el primer *Sprint*, que luego se incrementaron porque los integrantes notaron que llegaban tranquilos a las *Sprint Reviews* sin defectos y sin tener que dedicar demasiadas horas. En los siguientes *Sprints* el equipo buscó su velocidad óptima incrementando de a poco. En algunos casos donde se terminaron todas las funcionalidades planeadas también se adelantaron puntos de arreglos de *bugs*.

En el *Sprint* 8 se decidió comenzar con dos funcionalidades muy complejas, estas fueron la del enrutamiento de pedidos del lado del back-end y la de funcionar sin conexión a internet del lado del front-end. Estas funcionalidades, junto a otra pequeña, sumaban 27 puntos, un 42% más de lo realizados en el *Sprint* anterior. Sin embargo, como el equipo no supo cómo dividirlas, decidió comenzarlas de todos modos y al no poder terminarlas las tuvo como *carry-overs* al *Sprint* siguiente. Sumado a una mala estimación, estas historias terminaron tomando todo el *Sprint* 8 y el *Sprint* 9.

El equipo concluyó del *carry-over* del *Sprint* 8 que debían bajar la velocidad por lo que en el *Sprint* 10 ya volvieron a planear una cantidad de puntos más pequeña, además de aprender del error de estimación de esas historias que resultaron más complejas de lo esperado.

Sin embargo, viendo el lado positivo, esas dos historias fueron las únicas que el equipo consideró que subestimó suficiente como para no llegar a la *Sprint Review*. Afortunadamente ese fue el único *Sprint* en todo el proyecto donde el equipo tuvo *carry-overs*, por lo que concluyeron que en general hicieron un buen trabajo estimando las historias. Esto se debió principalmente a que ambos integrantes contaban con algo de experiencia previa en el lenguaje, experiencia estimando este tipo de historias de usuario utilizando puntos de esfuerzo y además fueron mejorando las estimaciones en el proceso.

La razón por la disminución de puntos planificados y realizados en el *Sprint* 13 es porque el equipo se fue de viaje algunos días durante el *Sprint*.

El *Sprint* 15 fue el último de la fase de desarrollo y originalmente se planificó vacío. Esto fue porque en el primer día de este *Sprint* se realizó la última prueba que tendrían antes de terminar oficialmente el programa, por lo que el equipo quería tener al menos un *Sprint* para arreglar los posibles defectos que se reportaran y corregir mejoras en usabilidad que pudieran surgir. La realidad es que se encontraron pocos *bugs* en la última prueba y solo algunas sugerencias que fueron producto de retroalimentación de los repartidores y del administrador, por lo que solo se realizaron 15 puntos de *bugs* y *feedback*. El equipo también aprovechó a hacer otras mejoras gráficas o de código. Disfrutaron de un último *Sprint* tranquilo durante el verano antes de comenzar con la etapa final de documentación.

La primera versión del *Release Plan* estaba diseñada para que con una velocidad promedio de 16 puntos por *Sprint* se llegara a terminar la versión inicial del *Backlog* para mediados de diciembre. El equipo logró una velocidad promedio de 18 y terminó agregando más funcionalidades de lo que originalmente se había pactado con el cliente. Además, solo fue un caso en el que el equipo no cumplió con el trabajo que se propuso para el *Sprint* por lo que se concluye que en general las estimaciones y planificaciones fueron buenas.

7.4.2. Distribución de horas

Como se mencionó, uno de los objetivos del equipo fue mantener la distribución de horas pareja durante todo el proyecto para evitar el caos y trabajar tranquilos, proponiéndose trabajar entre 10 y 30 horas semanales por persona.

Para monitorear este objetivo, el equipo decidió monitorear las horas utilizando la herramienta Toggle.

La siguiente gráfica refleja las horas trabajadas por semana por persona a lo largo de todo el proyecto. Para facilitar la lectura de las semanas se agregó uno de los prefijos: *inicial*, *dev*, *final*, para identificar la etapa. La numeración de la semana se reinicia al cambiar de fase para facilitar el cálculo del *Sprint* durante las semanas de desarrollo.

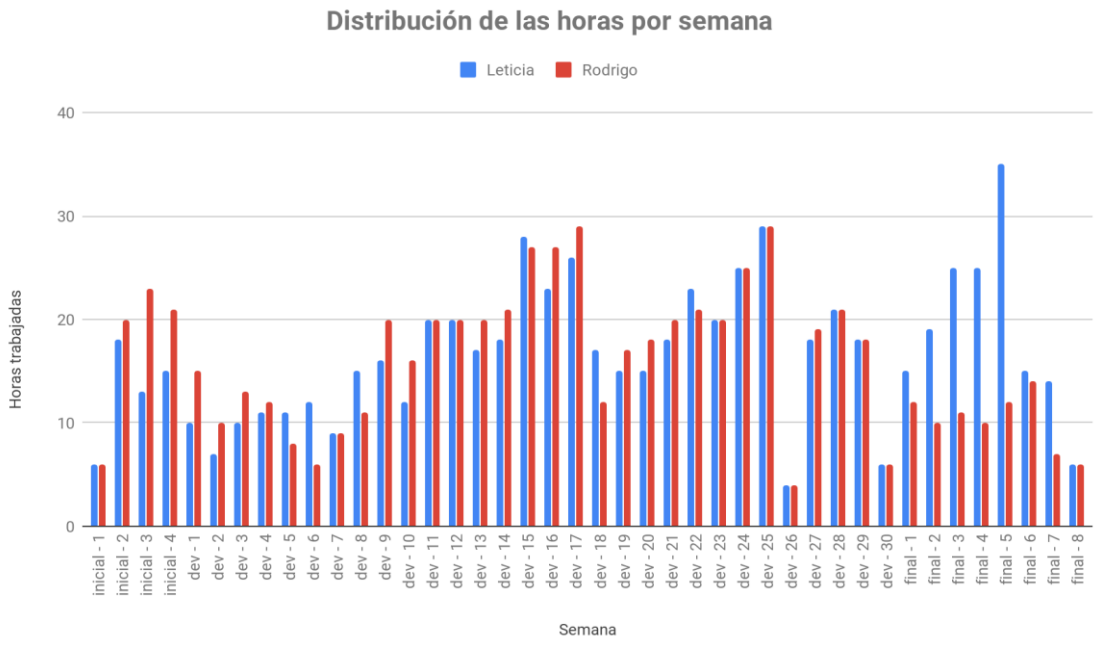


Ilustración 81 - Seguimiento de horas por semana

Se puede observar que las horas trabajadas por semana variaron entre semana y semana.

La primera semana del proyecto el equipo no estaba acostumbrado al ritmo de trabajo y no estaba seguro de cómo organizar el trabajo. Esto enseguida cambió y el equipo decidió invertir mucho esfuerzo en las semanas de la fase inicial para poder comenzar cuanto antes el desarrollo.

Durante las primeras semanas de desarrollo ocurrieron varias cosas. En primer lugar el equipo se dio cuenta que estaban llegando tranquilos al final del *Sprint* y que no habían tenido que invertir muchas horas. Entonces decidieron aumentar la velocidad.

Junto con el aumento de velocidad se aumentaron las horas de trabajo. En la segunda retrospectiva el equipo concordó que las funcionalidades realizadas en el segundo *Milestone* concentraban el trabajo en el back-end y no en el front-end, por lo que definieron reacomodar el *Release Plan* cuidadosamente para que las semanas balancearan el trabajo de Rodrigo y de Leticia.

Durante las próximas semanas de desarrollo Rodrigo tuvo algunas horas más de trabajo que Leticia. Esto se puede haber debido a que Leticia contaba con más experiencia en el lenguaje de back-end que Rodrigo en el de front-end. Se plantearon varias soluciones a esto, como redefinir los roles para que ambos trabajen en tanto back-end como front-end. Sin embargo, acordaron que esto podría ser contraproducente ya que implicaba capitaciones. Acordaron entonces no hacer ningún ajuste al respecto y que las horas sean las que se precisan para terminar las funcionalidades, aunque implique desbalances.

En la fase final, Leticia recuperó las horas de diferencia que tenía con Rodrigo en la fase de desarrollo. Esto se debió a que el equipo designó a Leticia para redactar la mayor parte de los documentos aunque se planeaban y revisaban en conjunto. Rodrigo realizó tareas como recopilación, revisión de formato, diagramas, etc., que no requirieron tanto tiempo. En la quinta semana de la fase final, Leticia se pidió licencia para dedicar a la documentación, sobrepasando el límite de 30 horas semanales planificado.

En conclusión, aunque las situaciones donde las horas trabajadas por semana quedaron fuera del rango de 10 a 30 horas fueron situaciones específicas, el equipo no logró cumplir con el objetivo planteado de mantenerse dentro del rango. De todos modos, el objetivo final del mismo era evitar acumular trabajo en ciertos momentos debido a atrasos, lo que no sucedió. En la gráfica se puede observar que si bien hubo semanas fuera del rango, en general el esfuerzo estuvo distribuido durante todo el año, permitiendo al equipo llegar tranquilos a las fechas de pruebas en repartos y a la entrega final.

En retrospectiva, la formulación del objetivo no fue la más apropiada. De todos modos, intentar cumplir con los rangos de horas por semana contribuyó a que el equipo distribuya el esfuerzo y logre trabajar a un ritmo sostenible en el tiempo.

7.4.3. Distribución de las horas por fase

A continuación se visualiza la gráfica de distribución de las horas por fase.

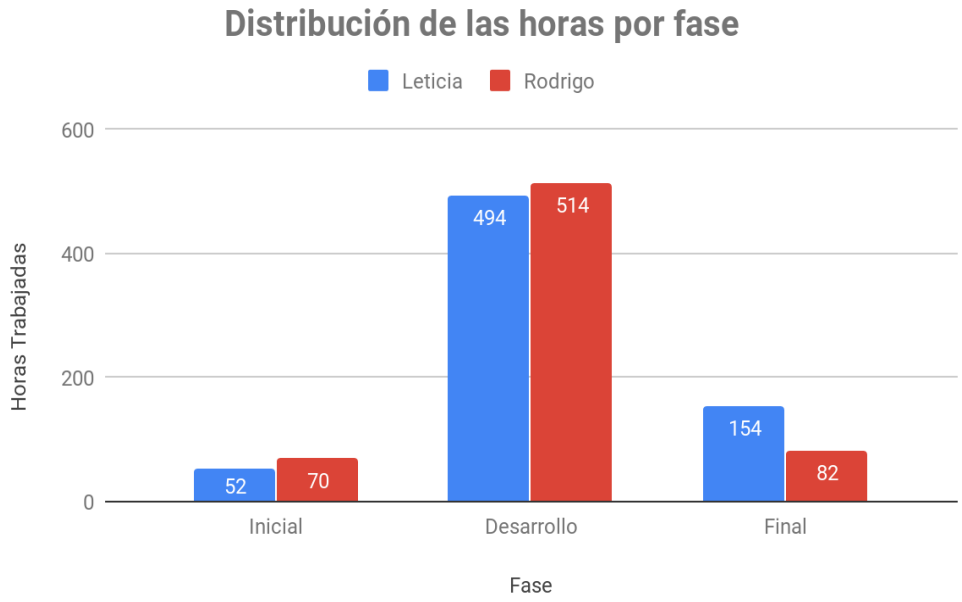


Ilustración 82 - Seguimiento de horas por fase

La fase inicial duró cuatro semanas, la de desarrollo treinta y la final ocho. En la fase inicial se invirtieron muchas horas en el armado de planes que luego se recopilaron en la fase final, ahorrando al equipo mucho tiempo de trabajo al final del proyecto. En la fase de desarrollo también se tuvieron horas en investigación, tareas de gestión, reuniones, revisiones, actas y pruebas por lo que no todas estas horas fueron directamente de desarrollo como se verá en la próxima gráfica.

Al finalizar el proyecto el equipo concluyó que la distribución de las fases fue apropiada para el proyecto, ya que permitió comenzar a desarrollar tempranamente, pero habiendo previamente realizado un correcto relevamiento y definido los planes necesarios.

7.4.4. Distribución del esfuerzo por área

Se definió realizar un seguimiento de las horas categorizadas según áreas. Toggle permite agregar etiquetas a las horas rastreadas por lo que se definió una etiqueta por área a rastrear.

Las áreas registradas fueron las siguientes:

Área	Descripción
Conocimiento del negocio	Involucra las entrevistas realizadas a los clientes y otros interesados para obtener el conocimiento del negocio durante la fase inicial.
Ingeniería de Requerimientos	Involucra las actividades realizadas durante la fase inicial para definir el problema y la solución, bajar a tierra el conocimiento de negocio obtenido, especificar y estimar los requerimientos, realizar los mock-ups y prototipos iniciales y las instancias de validación de los mismos.
Arquitectura	Involucra la identificación de los RNF y atributos de calidad, la definición de un diseño inicial de la arquitectura y la validación de la misma con expertos.
Definición de Planes	Involucra la definición del proceso y de los diferentes planes a utilizar a lo largo del resto de las etapas del proyecto. Por ejemplo el plan de gestión de riesgos, de calidad, SCM, seguimiento de tareas, control de cambios, control de defectos y marco metodológico.
Investigación	Involucra la investigación durante la fase inicial de las herramientas a utilizar a lo largo del proyecto y la investigación en la fase de desarrollo de las posibles soluciones al problema de enrutamiento. No involucra investigaciones de otros problemas tecnológicos durante la fase de desarrollo.
Reuniones, revisiones y actas	Involucra las horas de revisiones, reuniones con el tutor en la fase inicial, y con el cliente a partir de la fase de desarrollo. Incluye las horas de preparación para esas reuniones o revisiones, y las horas de armado de actas o informes. No incluye las horas de entrevistas para conocer el negocio.
Gestión	Involucra las tareas de gestión como recopilación de horas, registro de métricas luego de cada *Sprint*, reuniones de coordinación entre el equipo y todo lo que eso implica (Backlog Grooming, rearmado del Release Plan, retrospectiva, etc.).
Desarrollo	Involucra las actividades exclusivamente relacionadas a la construcción del software: codificación, pruebas, actividades de preparación de ambiente y despliegue. No involucra las horas de arreglo de defectos.
Retrabajo	Involucra exclusivamente la codificación para arreglar defectos en funcionalidades que ya se han liberado a los clientes, o codificación de funcionalidades por segunda vez por no haber entendido correctamente lo que el cliente quería. No involucra pequeñas mejoras (*feedbacks*).
Documentación final	Involucra las actividades realizadas durante la fase final sin contar las reuniones con el tutor: recopilación de los planes, redacción de nuevos, corrección de formato, diagramas, etc.

Tabla 9 - Áreas para el seguimiento de horas

Originalmente el equipo se planteó como objetivo tener un porcentaje de retrabajo menor al 5% de las horas totales, y de gestión y reuniones menor al 10%, por lo que medir las horas por áreas fue necesario para identificar si se cumplió o no.

Al finalizar el proyecto el equipo recopiló las horas dedicadas por ambos integrantes en conjunto durante todo el proyecto lo que le permitió realizar el siguiente gráfico de horas dedicada por área.

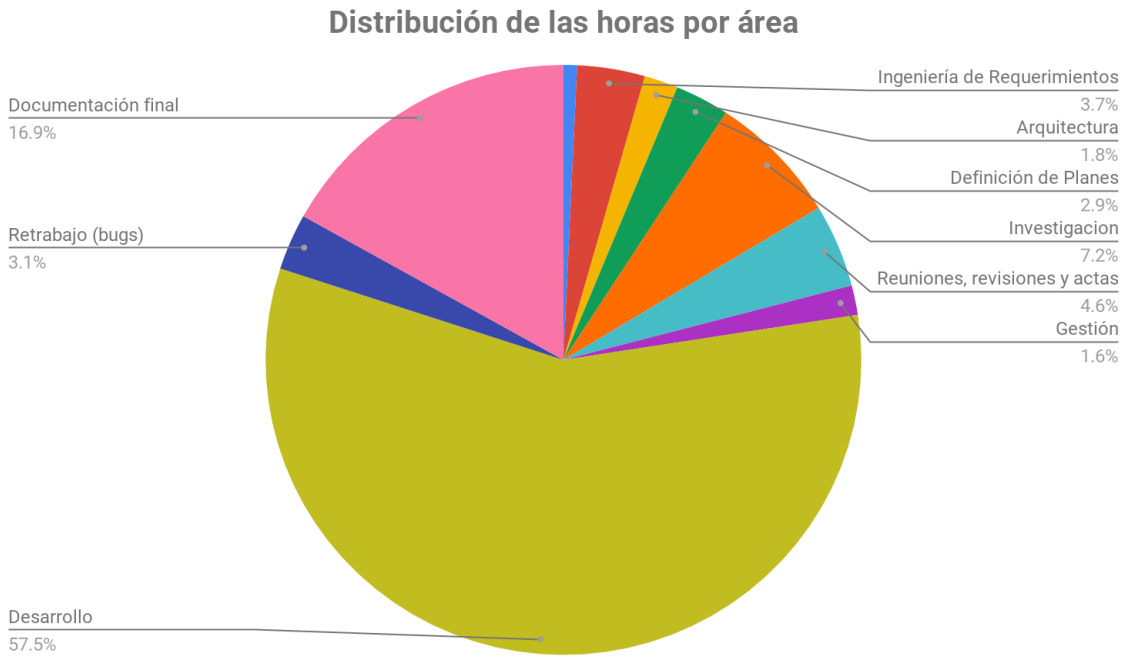


Ilustración 83 - Seguimiento de horas por área

Como se describe mejor en la sección 6.5.2 – “Métricas del producto”, el equipo registró menos de 30 defectos a lo largo del proyecto, producto de todo el esfuerzo invertido en pruebas antes de liberar las funcionalidades, a los altos estándares de calidad de código y a la alta cobertura mantenidos en cada *Sprint*. Las estimaciones de los arreglos de esos *bugs* sumadas conformaron alrededor de un 8% de los puntos del *Backlog* sin contar los *bugs*. De todos modos, del total de horas en todo el proyecto representaron el 3.1%. El equipo cumplió entonces con el objetivo propuesto respecto a mantener el retrabajo por debajo del 5%.

Cabe destacar que la funcionalidad de integración con Routeme podría haberse incluido como retrabajo ya que implicó rehacer totalmente la funcionalidad de enrutamiento de los pedidos que ya se había dado por terminada. Sin embargo, se definió que no se contaría dentro del porcentaje de retrabajo porque no había forma de haberlo previsto. El equipo pudo haber dejado la funcionalidad funcionando con la primera implementación, pero priorizó maximizar la escalabilidad del producto y darles más valor a los clientes.

Las horas dedicadas a gestión, reuniones y actas no superan el 7% del total, cumpliendo entonces también el objetivo de mantener este porcentaje menor a 10%.

En general, se concluye que el proceso que siguió el equipo fue exitoso en la que medida que permitió optimizar el tiempo, maximizando las horas de desarrollo, documentación e investigación, y presentando porcentajes mucho menores de retrabajo, tareas de gestión, reuniones, etc.

7.5. Gestión de riesgos

Por riesgo se entiende a “un evento o condición incierta que, si se produce, tiene un efecto positivo o negativo sobre al menos un objetivo del proyecto como tiempo, costo, calidad o alcance” [34]. No existe proyecto libre de riesgos e ignorarlos puede resultar en el fracaso del mismo. Como una manera de manejar esta incertidumbre bajo un enfoque estructurado el equipo definió un plan de gestión de riesgos.

El propósito de la gestión de riesgos fue definir estrategias para minimizar las probabilidades de ocurrencia de los riesgos negativos y tener un plan en caso de que ocurran para no sufrir consecuencias graves en el proyecto.

La utilización de una metodología ágil permitió reducir la incertidumbre de los riesgos ya que estos se evaluaban continuamente permitiendo cubrir nuevos riesgos que fueran apareciendo o reconsiderando los existentes.

Se consideró que las etapas de los riesgos son la identificación, análisis, plan de respuesta y contingencia.

7.5.1. Identificación

La identificación y seguimiento de los riesgos se realizó durante todo el proyecto. Al finalizar cada *Milestone*, durante la reunión de coordinación, el equipo se juntaría para evaluar una nueva versión de los riesgos.

Los riesgos identificados se especificaron de la siguiente manera:

Columna	Descripción
Riesgo	Nombre del riesgo identificado.
Fecha de identificación	Fecha o *Milestone* en el que se realizó la versión de los riesgos donde se identificó el riesgo por primera vez.
Fecha de desaparición	Fecha o *Milestone* en el que se realizó la versión de los riesgos donde se concluyó que el riesgo no tiene más probabilidad de ocurrencia y por ende desaparece.

Probabilidad	Probabilidad de ocurrencia del riesgo.
Impacto	Impacto que genera la ocurrencia del riesgo en el proyecto.
Magnitud	El producto entre la probabilidad y el impacto del riesgo, lo que nos permite clasificarlos y tratarlos de distinta manera.
Plan de respuesta	Plan de respuesta definido para el riesgo identificado.
Alerta	Alerta definida para el riesgo identificado.
Plan de contención	Plan de contención definido para el riesgo identificado.

Tabla 10 - Formato para la especificación de riesgos

Los riesgos identificados pertenecieron a diferentes áreas. La clasificación se realizó con el fin de tener en cuenta todas las áreas al momento de pensar en los riesgos del proyecto. Estas fueron:

Categoría	**Descripción**
Tecnológicos	Estos riesgos son los ocurridos en base a una falta de conocimiento en alguna tecnología o algún cambio de la misma que pudo generar un retraso o impedimento de una solución.
Del cliente	Involucra a los riesgos que no dependen del equipo sino del cliente, como su abandono, cambios en los requerimientos, descontento con alguna decisión del equipo, etc.
De gestión	Involucra a los atrasos en el cronograma, malas estimaciones y otros riesgos relacionados con el proceso.
De RRHH	Se basaron en la posible inhabilitación de integrantes del equipo por enfermedades, conflicto internos entre los integrantes del equipo, desbalance en la asignación de trabajo o falta de tiempo.

Tabla 11 - Categorías de riesgos

Cuando un riesgo era identificado se le asignaban valores para las siguientes variables:

- Probabilidad de ocurrencia (P)
- Impacto sobre los objetivos del proyecto (I)

Las escalas utilizadas fueron del 0 al 1 para la probabilidad y del 0 al 5 el impacto.

La magnitud fue la medida que se utilizó como indicador de la prioridad de cada riesgo. Esta se calculó como el producto entre la probabilidad de ocurrencia y el impacto.

Magnitud (M) = Probabilidad (P) * Impacto (I)

Los posibles valores de magnitudes se dividieron en tres franjas que dieron lugar a la clasificación de los riesgos según su prioridad: alta, media o baja, lo que permitió resolver qué acciones tomar con cada riesgo de manera estandarizada. Las escalas utilizadas para la definición de impacto y probabilidad de ocurrencia y la delimitación de franjas de los valores de magnitud se detalla en el ANEXO 31 – “Escalas para el análisis de riesgos”.

7.5.2. Plan de respuestas y contingencias

El plan de respuesta define las tareas a realizar a lo largo del proyecto para disminuir la probabilidad de la ocurrencia e impacto de un riesgo.

Por otro lado, el plan de contingencia especifica acciones para reducir el impacto de un riesgo cuando ocurra o cuando está por ocurrir para minimizar sus consecuencias.

7.5.3. Evolución de los riesgos

Como se fue mencionando el equipo decidió reevaluar los riesgos de forma periódica una vez al comenzar el proyecto y luego al finalizar cada *Milestone*. Durante estas revisiones el equipo verificó para cada riesgo si seguía existiendo, si cambió la probabilidad o magnitud, o si aparecieron nuevos riesgos. El listado de los riesgos gestionados con sus correspondientes planes se encuentra en el ANEXO 32 – “Listado de riesgos y planes”. Las versiones de los análisis de riesgos se pueden visualizar en el ANEXO 22 – “Reuniones de coordinación”.

A continuación se listan los principales riesgos gestionados.

Código	Nombre
R1	Escasez de tiempo para dedicarle la atención necesaria al proyecto
R2	Mala estimación de la velocidad del equipo
R3	Demorar más de lo estimado por falta de dominio en desarrollo Android
R4	Mala estimación de una o varias historias
R5	Dificultad de implementación del problema del enrutamiento
R6	Inhabilitación por más de una semana de un integrante del equipo por enfermedad
R7	Choque de roles
R8	Cambios grandes en los requerimientos ya implementados

R9	Conflictos entre los integrantes del equipo
R10	Dificultad de integración de tecnología para navegación
R11	Pérdida de los archivos de elaboración del proyecto
R12	Cancelación del proyecto por parte del cliente
R13	Repartidores no se acostumbran a utilizar el software
R14	Cliente insatisfecho con las rutas planificadas por Routeme
R15	Routeme decide interrumpir sus servicios
R17	Se reportan defectos durante repartos de prueba que el equipo no puede reproducir localmente

Tabla 12 - Principales riesgos

La siguiente gráfica describe cómo evolucionaron los riesgos mencionados anteriormente. Para mejorar la visualización se dividieron los riesgos en dos gráficas.

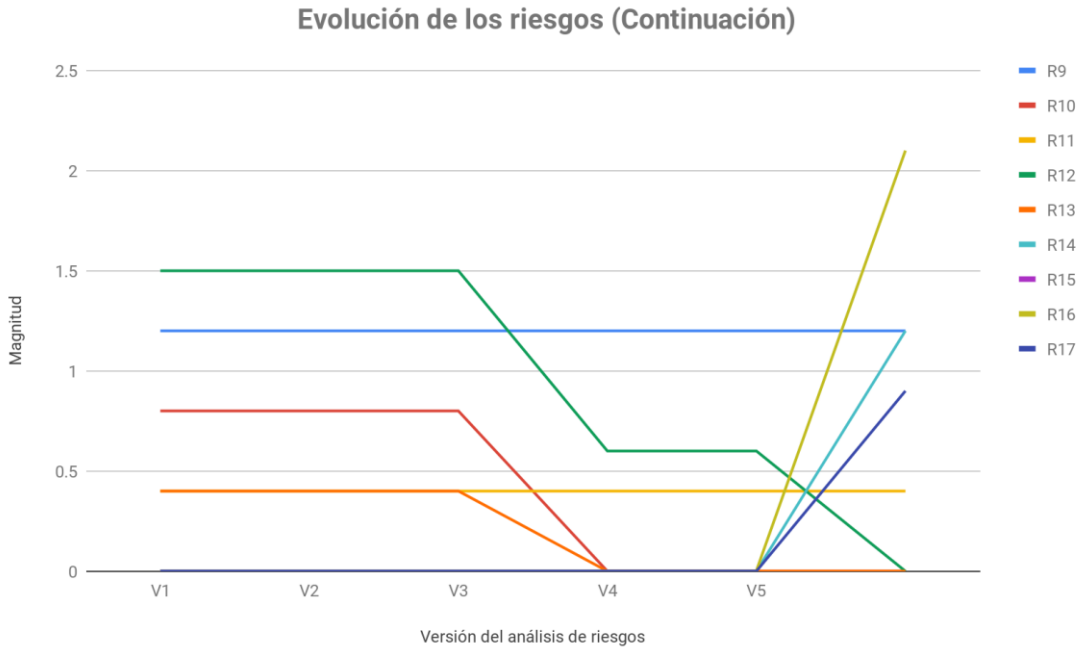


Ilustración 84 - Evolución de los riesgos

A continuación se explica la evolución de algunos de los riesgos.

R1 – Escasez de tiempo para dedicarle la atención necesaria al proyecto

Al trabajar y dar más materias de lo normal el equipo identificó este riesgo porque realmente se encontró una preocupación continua por escasez de tiempo para dedicarle al proyecto. La probabilidad de este riesgo empeoró cuando Leticia se pasó a 8 horas en el trabajo lo que le quitó horas de dedicación durante la semana. Volvió a disminuir al terminar las materias y enfocarse solamente en el proyecto y en el trabajo.

R4 – Mala estimación de una o varias historias

Dado que las fechas de los Releases se fijaron con anticipación porque era necesario acordar las fechas de las pruebas de validación de antemano, el hecho de mal estimar una historia podría suponer un retraso en la fecha del Release y perder la posibilidad de la prueba. De hecho, este riesgo sucedió ya que el equipo subestimó la historia del enrutamiento y fue necesario aplicar el plan de contención y no incluir esta funcionalidad en el Release con el fin de poder llevar a cabo la prueba de todas maneras.

La probabilidad de este riesgo fue disminuyendo conforme el equipo obtenía experiencia en la estimación. Una evolución similar se presentó en los riesgos R3 –

“Demorar más de lo estimado por falta de dominio en desarrollo Android” y R2 – “Mala estimación de la velocidad del equipo”.

R7 – Choque de roles

Durante las etapas iniciales se consideró que podría ser difícil distinguir las tareas de cada integrante y que la solapamiento de las mismas no resulte en retrabajo o tareas sin realizar. Durante la etapa de desarrollo la probabilidad de este riesgo bajó a cero ya que los roles front-end y back-end estaban muy bien diferenciados. Volvió a aparecer en la fase final de documentación.

R8 – Cambios grandes en los requerimientos ya implementados

En un principio se planteó la posibilidad de tener mal entendidos con los clientes durante la *Sprint* Planning e implementar las funcionalidades de otra forma a como las esperaban, o de descubrir en pruebas de validación que la solución implementada no es la correcta. Los casos anteriores resultarían en retrabajo que se buscó minimizar durante el proyecto. Para evitar este riesgo el equipo utilizó criterios de aceptación claros durante las reuniones, prototipos y realizó pruebas de validación a lo largo de todo el proyecto, reduciendo progresivamente la probabilidad de que esto suceda.

R9 – Conflictos entre los integrantes del equipo

Dada la naturaleza de la relación del equipo fue sumamente importante mantener una buena dinámica y convivencia para evitar conflictos que puedan disminuir la productividad. Es por eso que se evaluaron los mejores métodos de comunicación para evitar conflictos y se definió un reglamento interno. Se consideró que el riesgo tuvo el mismo impacto y probabilidad de ocurrencia durante todo el proceso.

R12 – Cancelación del proyecto por parte del cliente

El impacto de la cancelación del proyecto por parte del cliente se consideró muy alto al principio. Disminuyó después del Milestone 3, cuando el equipo ya había realizado una validación formal con el cliente y que el mismo abandonase el proyecto ya no hubiese tenido tal impacto para el equipo ya que se contaba con evidencia suficiente para la documentación final.

Por otro lado, la probabilidad de que el cliente abandone siempre fue baja ya que se le hizo firmar una carta de compromiso con el equipo de manera de evitar este riesgo y se le transmitió el impacto que tendría. Al igual que el impacto, la probabilidad de este riesgo también disminuyó luego de la primer validación ya que el cliente vio que contaba con un software que le era de valor y que servía en repartos reales, haciendo más improbable que abandone.

R13 - Repartidores no se acostumbran a utilizar el software

Este riesgo hubiese tenido una magnitud mayor, pero como se definió en la estrategia de prevención, el equipo realizó pruebas de usabilidad utilizando los prototipos antes de comenzar a desarrollar, disminuyendo la probabilidad de que suceda. Luego del Milestone 3, donde se realizó la primer prueba en repartos reales, se vio que los repartidores pudieron utilizar la aplicación por lo que el riesgo se eliminó.

8. Conclusiones

8.1. Estado Actual

Hoy en día la aplicación móvil se está utilizando por la empresa diariamente en sus repartos.

La aplicación web administradora se encuentra disponibles de forma pública, aunque no es posible acceder sin un usuario y contraseña de administrador.

La aplicación está alojada en una cuenta privada de Google Console, accesible al usuario de Martín García, donde se configuró que la aplicación se instale automáticamente en el celular de los repartidores y que los mismos no la puedan desinstalar.

8.2. Conclusiones generales

A continuación se listan los objetivos que el equipo se propuso al inicio del comienzo enunciando si se cumplieron o no.

Objetivo	Se cumplió
Lograr un producto que sea puesto en producción antes de la entrega final	Sí. El producto se comenzó a utilizar en producción desde la segunda prueba, en enero.
Lograr un producto con cero defectos críticos reportados al momento de la entrega final	Sí. El producto no tiene defectos de severidad media o alta reportados.
Conseguir un promedio mayor a 4 y ningún resultado menor a 3 en la encuesta de satisfacción a los interesados al terminar el proyecto	Sí. El equipo consiguió un promedio de 4.9 en la segunda instancia de encuestas a los repartidores, administrador y clientes.
Que al finalizar el proyecto ambos integrantes respondan afirmativamente a la pregunta ¿Volverían a trabajar en un proyecto juntos?	Sí. Ambos integrantes acuerdan que volverían a repetir la experiencia en otro proyecto.
Trabajar entre 10 y 30 horas semanales por persona a lo largo de todo el proyecto.	No. Sin embargo se concluyó que la intención del objetivo de distribuir el trabajo a lo largo de todo el proyecto se logró, pero

	que la formulación del criterio de aceptación del objetivo no fue el más apropiado.
Obtener un porcentaje de retrabajo menor al 5% de las horas totales.	Sí, el porcentaje de retrabajo fue el 3%.
Lograr que el tiempo dedicado a tareas de gestión o de coordinación como reuniones y actas sea menor al 10% de las horas totales para el final del proyecto.	Sí, el porcentaje de estas tareas fue el 6%.

Ilustración 85 - Evaluación del cumplimiento con los objetivos

8.3. Lecciones aprendidas

A continuación se describen las principales lecciones aprendidas de las grandes decisiones o actividades del proyecto.

Ciclo de vida incremental iterativo

La elección de un ciclo vida incremental e iterativo fue fundamental para poder contemplar las sugerencias que iban surgiendo, así como las posibilidades de mejora que el equipo detectó ya a mediados de la etapa de desarrollo. Brindó al cliente la tranquilidad de ver el producto funcionando en etapas tempranas y al equipo la flexibilidad que construir en incrementos trajo consigo.

Las iteraciones de dos semanas se adaptaron perfecto. Fueron el tiempo suficiente para no tener sobrecarga de reuniones y para poder distribuir el trabajo como el equipo quiso dentro de un período que sigue siendo corto y manejable.

Scrum

Se consideró que Scrum fue una buena decisión. El foco que pone en lograr un equipo autogestionado, que aprende y adapta el proceso continuamente fue de mucha ayuda. No todos los proyectos y equipos son iguales y es necesario que cada uno aprenda a adaptar el marco de trabajo. El equipo realizó ciertas adaptaciones que hicieron que el proceso no fuese tan tedioso de seguir, como el concatenar las reuniones *Sprint Planning* y *Sprint Review*, e incluir la retrospectiva en la reunión de coordinación una vez por *Milestone*, permitiendo dar un cierre adecuado al mismo y asegurar un progreso firme. A su vez, el equipo supo identificar prácticas que solo estaban quitando tiempo y decidieron descontinuarlas, como las actas de reuniones con el tutor.

Se aprendió que para que el equipo pueda efectivamente autogestionarse es fundamental respetar las instancias que se definieron cumplir, para lograr la confianza del cliente y entre los integrantes del equipo. El esfuerzo del equipo de preparar las reuniones previamente hizo que estas fueran eficientes y profesionales, además de haber asegurado que las funcionalidades a mostrar se prueben antes de las reuniones minimizando la cantidad de defectos reportados luego.

Scrum además permitió postergar ciertas decisiones hasta el momento en el que se precisaron, evitando al equipo invertir mucho tiempo al principio escribiendo todos los criterios de aceptación de funcionalidades que muchas se terminaron eliminando o modificando radicalmente.

Release Plan

Se terminó de comprobar la tranquilidad que brinda el trabajar bajo un plan. Si bien se fue modificando todo el tiempo, el *Release Plan* fue el esqueleto del camino a seguir durante todo el proyecto. Se lo cuidó y mantuvo con mucha disciplina, actualizándolo con el trabajo realizado en los *Sprints* anteriores y replanteando los que se vienen según las nuevas prioridades. Para ello también fue necesario tener el *Product Backlog* completo, priorizado y estimado, en lo cual los clientes fueron de gran ayuda.

***Spike* de investigación**

El *Spike* permitió tener un plan de la solución antes de realizarla, para poder estimarla más precisamente y evitar retrabajo. El equipo recomendaría la aplicación de un *Spike* previo a la realización de cualquier funcionalidad de alta complejidad o riesgo en este o cualquier otro proyecto que trabaje en el contexto ágil.

Se verificó también la importancia de tener un plan para organizar el *Spike* de manera de evitar el caos y maximizar la eficiencia.

Ingeniería de requerimientos

El equipo se vio parado frente a un problema en un negocio del cual no sabía absolutamente nada, y tuvo que aprender a obtener el conocimiento necesario para poder entender el contexto, bajar a tierra el problema y proponer una solución que realmente sirva.

Se entendió por qué los prototipos son la mejor herramienta para validar un producto antes de construirlo. El tiempo que se invierte en un prototipo se recupera de retrabajo luego. Además, los clientes se quedaron muy satisfechos con los prototipos y manifestaron su tranquilidad.

Por otro lado, la herramienta *Story Map* resultó muy valiosa ya que fue una forma visual de detectar la forma más rápida de cumplir el objetivo del programa para poder

validar el producto cuanto antes. De no haberse usado quizás el equipo hubiese gastado tiempo en construir requerimientos que parecían necesarios para el flujo pero que en realidad no lo eran, postergando la primer prueba en un reparto real. Se aprendió entonces que la priorización es mucho más importante de lo que parece, especialmente en un grupo de dos personas trabajando bajo una metodología ágil.

Finalmente, se consideró que las técnicas de estimación utilizadas fueron apropiadas para el equipo, ya que en general se tuvo buenas estimaciones que se reflejan en la casi inexistente diferencia entre lo que se planeó y lo que se realizó *Sprint* a *Sprint*.

Arquitectura y Desarrollo

Se verificó la importancia de identificar los atributos de calidad en base a los requerimientos no funcionales de modo de cumplir con las funcionalidades del sistema antes de comenzar con el desarrollo. Se comprendió la utilidad de la aplicación de tácticas y patrones conocidos para los problemas que frecuentan en la mayoría de los sistemas, ya que estos permitieron al equipo diseñar rápidamente una arquitectura que cumplió con todos los requerimientos no funcionales.

Se destaca como una decisión acertada la selección de lenguajes y otras tecnologías ya que el equipo logró minimizar el tiempo de configuración enfocándose en el desarrollo y pudiendo entregar funcionalidad muy rápidamente. Se verificó especialmente la importancia de herramientas de reporte de incidentes para poder descubrir y entender las caídas de la aplicación sin depender de que el usuario las explique.

Problema del enrutamiento

Incluso habiendo realizado un *Spike* de investigación previamente, equipo subestimó el esfuerzo de la solución del problema de enrutamiento ya que terminó llevando el doble del tiempo planificado debido a todos los casos de prueba y detalles del algoritmo diseñado. Se entendió la alta complejidad del problema que se había subestimado al comenzar el proyecto.

Sobre el cambio de implementación, el equipo se llevó el aprendizaje de que si bien algunas veces puede implicar retrabajo, es parte del proceso saber aprovechar las oportunidades y no apegarse a un algoritmo o diseño solo porque se invirtió mucho tiempo en ellos. Es parte de dirigir un proyecto el tener que poner en la balanza casos como este y evaluar si el valor versus el retrabajo lo vale.

Gestión de Calidad

Se comprobó que todo el esfuerzo invertido, en tanto la definición como el seguimiento de planes para asegurar la calidad del producto y del proceso, dieron resultados.

Como consecuencia se logró un producto que funciona en producción sin defectos críticos reportados, con excelente grado de mantenibilidad debido a las herramientas de análisis de calidad de código y a la alta cobertura que se incorporaron como forma habitual de trabajo.

Por otro lado se siguió un proceso organizado que permitió trabajar tranquilos y motivados, sin *overhead* de reuniones o tareas de gestión, y que dio lugar al arreglo de defectos y a la incorporación de las sugerencias y nuevas funcionalidades que fueron surgiendo.

Se aprendió la importancia de definir cuidadosamente las métricas a utilizar en el proyecto, ya que las mismas deben permitir comprobar la verificación de los objetivos. De la misma forma, realizar el seguimiento de las métricas definidas, por más tedioso que sea, es fundamental para tomar decisiones inteligentes que permitan mejorar en varios aspectos. El equipo se sorprendió el observar que las métricas fueron realmente brindando información útil para mejorar el proceso y la planeación.

Gestión de cambios

Se aprendió que la gestión de cambios es particularmente útil en proyectos como este donde las reuniones son fuente de tormentas de ideas y mucha retroalimentación que debe ser filtrada. El equipo aprendió que en esos casos definir un plan de gestión de cambios que balancee el incorporar feedback y nuevas funcionalidades que van surgiendo con el no perder el foco en el problema original es fundamental.

Pausa

Durante el desarrollo el equipo se tomó una semana de licencia. Previo a eso el equipo se mostraba completamente inmerso en el desarrollo comenzando a perder la perspectiva y a sentirse cansados. Esto se dio principalmente porque los integrantes venían de completar muchas horas durante los *Sprint*s anteriores, de las entregas de facultad, de mucho trabajo, y además se encontraban tomando la difícil decisión de si integrar con Routeme para el problema de enrutamiento o no.

Tanto el tutor como un revisor recomendaron pausar para tomar un respiro. El equipo se tomó unos días para viajar y despejarse, y al volver se encontraban más motivados y seguros del trabajo que venían realizando. Realizaron entonces una reunión de coordinación más extensa, donde se aseguraron de no perder de vista los objetivos.

Se entendió entonces la importancia de pausar para tomar distancia, analizar los problemas, juntar aire y continuar avanzando.

8.4. Conclusiones personales

Rodrigo

“Este proyecto fue el mayor desafío que tuvimos como equipo en estos cinco años de carrera. Lo manejamos con un alto grado de profesionalidad y mantuvimos una excelente relación entre el equipo y con los clientes, lo que considero un logro personal.

A diferencia de otros proyectos, en este trabajamos en un producto real que hoy en día está en uso oficialmente. La realidad del producto nos llevó a poner lo mejor de nosotros para entregar incrementalmente a los clientes una solución eficiente a su problema y adaptada a su realidad.

Por otro lado, el proyecto mejoró mis habilidades como desarrollador *mobile*, sobre todo en el lenguaje Java para Android. Fue un desafío para mí el haber sido el único desarrollador front-end en el proyecto utilizando una tecnología que no dominaba del todo. Durante el paso del año pude pulir mis habilidades hasta lograr sentirme realmente cómodo.

En cinco años de trabajo en conjunto en numerosos proyectos académicos y personales nunca habíamos seguido un proceso tan riguroso. Consideramos que aunque fue algo cansador al comienzo, fue lo que nos permitió completar un proyecto de esta magnitud sin perder el control, y aprender juntos como equipo durante todo su recorrido.”

Leticia

“Creo que este proyecto fue un excelente cierre a nuestra carrera. Fue sin dudas el mayor desafío por el que pasamos. El tener que llevar a cabo un proyecto de ingeniería con todo lo que eso implica, siendo solamente dos personas para cubrir todos los roles necesarios fue una experiencia difícil al principio pero muy satisfactoria al final.

Por otro lado, siento que luego de cinco años de realización de obligatorios que solamente se archivan, el haber construido un sistema que se esté realmente utilizando, que sirva de algo y tenga valor para alguien, es una sensación increíble. Comprueba nuestro aprendizaje durante la carrera y nuestro valor en la sociedad.

Sobre la empresa, la realidad es que tuvimos mucha suerte en tener clientes con ese grado de interés en el proyecto, que además confiaron plenamente en el equipo y que cumplieron ampliamente con su parte. Lo menciono porque sé que no es habitual y tuvo un gran peso en el éxito del proyecto.

En general creo que hicimos un excelente trabajo. Con mucha disciplina para aplicar los planes que definimos y, por más tonto que suene, mucho cariño en todo lo que hicimos, logramos cumplir con lo que nos propusimos.”

8.5. Próximos pasos

Luego de la entrega académica , el equipo y el cliente tienen acordado juntarse para coordinar cómo continuará el vínculo entre ambas partes.

Corregir últimos defectos

Los nuevos defectos reportados en producción durante la fase final de documentación se siguieron registrando para posterior manejo. Luego de la entrega final el equipo y el cliente tienen pendiente reunirse para coordinar su arreglo.

Nuevas funcionalidades

Al finalizar el proyecto, muchas funcionalidades que fueron surgiendo como ideas pero que no entraron en el alcance final se fueron registrando en la columna Icebox en el Trello. Esta lista puede encontrar en el ANEXO 33 – “Icebox”. El equipo y el cliente se reunirán para conversar acerca de su implementación.

Adaptación para ofrecer como SaaS a otras empresas

La empresa implantó oficialmente en sus repartos el producto construido y manifestó al equipo su intención de ofrecerlo como servicio en la nube a otras empresas del rubro de logística. El equipo y la empresa acordaron tener una reunión luego de la entrega para negociar un acuerdo y de llegar a uno el software deberá ser adaptado para permitirlo.

8.6. Carta de conformidad escrita por el cliente

Al finalizar el proyecto Martín García tuvo la gentileza de redactar la siguiente carta avalando el trabajo del equipo.

En la misma se lee:

> *Señores Comité de Proyectos, Universidad ORT Uruguay.*
>
> *Por intermedio de la presente, quien suscribe, Martín García, en representación de la empresa FORMISUR S.A., me pongo en contacto con ustedes para manifestar nuestra total conformidad con Rodrigo Arsuaga y Leticia Esperón, estudiantes ellos, quienes llevaron adelante nuestro proyecto "Geolocalizador de repartos" como su tesis de grado.*
>
> *Quiero destacar el grado de compromiso y dedicación que demostraron durante el último año, cumpliendo con creces los objetivos planteados durante cada etapa del desarrollo.*
>
> *Con total agrado, transmito en nombre de la empresa, la satisfacción de haber colaborado con el equipo, cuyas ideas y soluciones hicieron de este un proyecto exitoso.*
>
> *Por último, dejo constancia de que el producto ya se encuentra en producción.*
>
> *Atentamente,*
>
> *Martín García*

Formisur S.A.

Luis Batlle Berres 8580
Montevideo, Uruguay
(2) 313 8941
MARTINGARCIA@FORMISUR.COM.UY

28 de febrero de 2019

Señores Comité de Proyectos, Universidad Ort Uruguay.

Presente.

Por intermedio de la presente, quien suscribe, Martín García, en representación de la empresa Formisur S.A., me pongo en contacto con ustedes para manifestar nuestra total conformidad con Rodrigo Arsuaga y Leticia Esperón, estudiantes ellos, quienes llevaron adelante nuestro proyecto "Geolocalizador de Repartos" como su tesis de grado.

Quiero destacar el grado de compromiso y dedicación que demostraron durante el último año, cumpliendo con creces los objetivos planteados durante cada etapa del desarrollo.

Con total agrado, transmito en nombre de la empresa la satisfacción de haber colaborado con el equipo, cuyas ideas y soluciones hicieron de este un proyecto exitoso.

Por último, dejo constancia de que el producto ya se encuentra producción.

Atentamente,

Ilustración 86 - Carta de conformidad

9. Glosario

Aplicación del administrador: Aplicación web diseñada por el equipo para la empresa que permitirá a los administradores realizar tareas como ingreso de los pedidos, tracking de los repartos, visualización de los datos de un pedido, etc.

Aplicación del repartidor: Aplicación móvil que utilizarán los repartidores de la empresa para visualizar los próximos pedidos de un reparto, el camino hacia un destinatario de un pedido, etc.

Bug: Defecto del software a construir.

Clientes: Representantes de Formisur, en particular Martín García y Nicolás Benítez.

Destinatario (final): Cliente de un cliente de la empresa Formisur, a quien se le debe entregar un pedido.

Empresas cliente: Clientes de la empresa Formisur que contratan a la empresa para que reparta pedidos a sus respectivos clientes, los destinatarios finales.

Estado de un pedido: Situación de un pedido durante un reparto: pendiente, rebotado o entregado (aceptado).

Equipo: Los dos estudiantes que realizaron este proyecto.

Formisur: Empresa dedicada a la logística empresarial para la cual se realiza el software. Dentro de los servicios que provee se encuentran la recepción, almacenaje, etiquetado, control y distribución de mercadería a todo el país.

Motivo de rebote: Razón por la cual un pedido fue marcado como rebotado, por ejemplo “no había nadie en casa”.

Pedido: Envío, orden, paquete. Unidad que la empresa de logística debe entregar a un destinatario en una fecha dados por un cliente de la empresa.

Pedido completado: Pedido que fue marcado como rebotado o entregado durante un reparto.

Recorrido: Camino que realizó el repartidor durante un reparto.

Repartidor: Cadete de la empresa. Normalmente a cada reparto se le asignan dos repartidores.

Reparto: Agrupación de pedidos que serán distribuidos a sus respectivos destinatarios por uno o dos repartidores en una camioneta en un mismo día.
Campaña: Pedidos de un cliente para los próximos 30 días.

Ruta: Orden de los pedidos de un reparto.

Ruta base: Orden predefinido para un conjunto de pedidos.

Software a construir: Producto construido por el equipo en la duración del proyecto

Tracking: Actividad de seguir o monitorear en tiempo real el estado de un pedido o reparto.

Ubicación Base: Local de Formisur de donde salen o llegan los repartidores.

10. Referencias bibliográficas

[1] BEEVA, “User Story Mapping, el product backlog de un vistazo” [En línea]. Disponible:https://www.beeva.com/beeva-view/metodologiasagiles/user-story-mapping-el-produc-backlog-de-un-vistazo/. [Último acceso: Febrero 2019]

[2] Trello, “Trello” [En línea]. Disponible: https://trello.com/ . [Último acceso: Febrero 2019].

[3] Invision, “Invision” [En línea]. Disponible: https://www.invisionapp.com/ [Último acceso: Febrero 2019].

[4] Scrum, “Why do we use Fibonacci for estimation” [En línea]. Disponible: https://www.scrum.org/forum/scrum-forum/7897/why-do-we-use-fibonacci-series-estimation/ [Último acceso: Febrero 2019].

[5] IEEEXplor, “Adaptive sampling frequency for sampled-data control systems” [En línea]. Disponible: https://ieeexplore.ieee.org/document/1105415 [Último acceso: Febrero 2019].

[6] Logentries, “Logentries” [En línea]. Disponible: https://logentries.com/. [Último acceso: Febrero 2019].

[7] Slack, «Slack,» [En línea]. Disponible: https://slack.com/intl/es/. [Último acceso: Febrero 2019].

[8] SitePoint, “The (Silver) Bullet for the N+1 Problem” [En línea]. Disponible: https://www.sitepoint.com/silver-bullet-n1-problem/ [Último acceso: Febrero 2019].

[9] Heroku, “Customer Promises” [En línea]. Disponible: https://www.heroku.com/policy/promise . [Último acceso: Febrero 2019].

[10] Heroku, “Heroku Status” [En línea]. Disponible: https://status.heroku.com/ [Último acceso: Febrero 2019].

[11] Heroku, “Production Check” [En línea]. Disponible: https://devcenter.heroku.com/articles/production-check [Último acceso: Febrero 2019].

[12] Amazon, “Amazon SLA” [En línea]. Disponible: https://aws.amazon.com/s3/sla/ [Último acceso: Febrero 2019].

[13] 12 Factor App, “The Twelve factor app” [En línea]. Disponible: https://12factor.net/es/ [Último acceso: Febrero 2019].

[14] Rspec, “Rspec” [En línea]. Disponible: http://rspec.info/ [Último acceso: Febrero 2019].

[15] Webmock, “Webmock” [En línea]. Disponible: https://github.com/bblimke/webmock [Último acceso: Febrero 2019].

[16] Google, “HTTPS” [En línea]. Disponible: https://support.google.com/webmasters/answer/6073543?hl=en [Último acceso: Febrero 2019].

[17] Wikipedia, “Man in the middle” [En línea]. Disponible: https://en.wikipedia.org/wiki/Man-in-the-middle_attack [Último acceso: Febrero 2019].

[18] Wikipedia, “Bcrypt” [En línea]. Disponible: https://en.wikipedia.org/wiki/Bcrypt [Último acceso: Febrero 2019].

[19] Amazon, “Amazon S3” [En línea]. Disponible: https://docs.aws.amazon.com/es_es/AmazonS3/latest/dev/Welcome.html [Último acceso: Febrero 2019].

[20] Amazon, “AWS RDS” [En línea]. Disponible: https://docs.aws.amazon.com/es_es/AmazonRDS/latest/UserGuide/Welcome.html [Último acceso: Febrero 2019].

[21] Heroku, “Active Storage Heroku” [En línea]. Disponible: https://devcenter.heroku.com/articles/active-storage-on-heroku [Último acceso: Febrero 2019].

[22] ActiveAdmin, “ActiveAdmin” [En línea]. Disponible: https://activeadmin.info/ [Último acceso: Febrero 2019].

[23] Wikipedia, “Google Developers” [En línea]. Disponible: https://en.wikipedia.org/wiki/Google_Developers [Último acceso: Febrero 2019].

[24] Wikipedia, "Google Play” [En línea]. Disponible: https://es.wikipedia.org/wiki/Google_Play [Último acceso: Febrero 2019].

[25] SQLite, “About SQLite” [En línea]. Disponible: https://www.sqlite.org/about.html [Último acceso: Febrero 2019].

[26] UWaterloo, “The Traveling Salesman Problem” [En línea]. Disponible: http://www.math.uwaterloo.ca/tsp/ [Último acceso: Febrero 2019].

[27] Google, “Directions API” [En línea]. Disponible: https://developers.google.com/maps/documentation/directions/intro [Último acceso: Febrero 2019].

[28] Realtime Api Hub, “Polling” [En línea]. Disponible: https://realtimeapi.io/hub/polling/ .[Último acceso: Febrero 2019].

[29] Pablo Giugni, “La calidad como filosofía de gestion” [En línea]. Disponible: https://www.pablogiugni.com.ar/joseph-m-juran/ [Último acceso: Febrero 2019].

[30] CodeBeat, “CodeBeat” [En línea]. Disponible: https://codebeat.co/ [Último acceso: Febrero 2019].

[31] Wikipedia, “Peter F. Drucker” [En línea]. Disponible: https://es.wikipedia.org/wiki/Peter_F._Drucker [Último acceso: Febrero 2019].

[32] Simplypsycology, “LikertScale” [En línea]. Disponible: https://www.simplypsychology.org/likert-scale.html [Último acceso: Febrero 2019].

[33] Atlassian, “Gitflow” [En línea]. Disponible: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow [Último acceso: Febrero 2019].

[34] Project Management Institute. 2004. *PMBOK Guide, A Guide to the Project Management Body of Knowledge*. 3ra ed. Pennsylvania: Project Management Institute Inc.

[35] Postman, “Postman” [En línea]. Disponible: https://www.getpostman.com/ [Último acceso: Febrero 2019].

[36] Fabric, “Fabric” [En línea]. Disponible: https://get.fabric.io/ [Último acceso: Febrero 2019].

[37] Rollbar, “Rollbar” [En línea]. Disponible: https://rollbar.com/ [Último acceso: Febrero 2019].

[38] Lifewire, “What is Google Sheets” [En línea]. Disponible: https://www.lifewire.com/google-sheets-4157491 [Último acceso: Febrero 2019].

ANEXO 1 - “Análisis de metodologías ágiles”

El equipo realizó un análisis de las principales metodologías ágiles para evaluar cuál utilizar en el proyecto. El criterio que se utilizó para determinar la viabilidad en el proyecto es el siguiente: no se adapta (1), se adapta parcialmente (2) o se adapta (3). A continuación se presentan las metodologías estudiadas con su nivel de adaptación al proyecto.

Metodología	Nivel de adaptación
Scrum	3
Feature Driven Development	2
Lean	2
Kanban	1

Tabla 13 - Comparación de metodologías ágiles

Scrum

Es un proceso que se orienta a proyectos en entornos complejos, donde es necesario obtener resultados de forma temprana y requerimientos cambiantes o poco definidos. Busca potenciar la innovación, complejidad, productividad y flexibilidad. Posee entregas parciales del producto final.

Ventajas	Desventajas
Altamente adaptable a los de requerimientos durante el ciclo de vida	Como los requisitos del producto se pueden ajustar con frecuencia, rara vez se documentan. Esto significa que el desarrollo puede tomar un giro lateral si el propietario del producto no está absolutamente seguro de los requisitos del producto.
El *Product Owner* está siempre en contacto con el equipo de desarrollo para mantener transparencia	Solo los desarrolladores senior pueden controlar un modelo de gestión de proyectos de este tipo, lo que significa que debería haber al menos un maestro Scrum dedicado en cada equipo y deberían mantener una mirada de cerca para garantizar que el trabajo de cada desarrollador vaya en la dirección correcta.

A medida que las pruebas se realizan después de cada iteración, los errores y los “features” innecesarios se corrigen rápidamente. Volver a una versión anterior del producto es bastante fácil.	Los marcos de tiempo iniciales se ajustan con demasiada frecuencia y entregar el producto esperado precisamente a tiempo no es algo que ocurra a menudo.

Tabla 14 - Ventajas y desventajas de Scrum

El modelo Scrum de metodología Ágil es ideal para un equipo más pequeño que trabaja en el desarrollo continuo y la implementación de nuevas características para un producto con un largo desarrollo. Las pruebas son exhaustivas, las iteraciones son rápidas, las nuevas funciones que se realizan rápidamente y el producto está operativo en todo momento.

Feature-Driven Development (FDD)

En el desarrollo basado en features o el modelo FDD, los features son el enfoque del desarrollo. A medida que se agregan las features, se introducen los nuevos conjuntos de requisitos. Esto funciona mejor para equipos internos más grandes de desarrolladores, trabajando en la mejora incremental de un producto a gran escala. Este enfoque también se utiliza a menudo como un flujo de trabajo interino entre las metodologías de desarrollo de software de Waterfall y Agile.

Ventajas	Desventajas
Funciona muy bien con productos grandes que requieren actualizaciones constantes y entrega de valor.	No funciona con equipos chicos.
Basado en una documentación de buenas prácticas ya establecidas. Asegurando a que desarrolladores con cualquier tipo de experiencia encuentren su rol y trabajar en el proyecto.	No funciona con proyectos chicos con entregas pautadas.
Cada iteración da un entregable más completo que el anterior.	Requiere un equipo altamente capaz para monitorear el proyecto durante todo el proceso y sus etapas de desarrollo.

Tabla 15 - Ventajas y desventajas de FDD

El modelo FDD es mejor utilizado por los grandes equipos de desarrolladores, que trabajan para mejorar constantemente el rendimiento de un producto integrado, como

un software bancario. Hoy en día es bastante popular, sin embargo, aún pierde popularidad con otros modelos de desarrollo de software similares a Agile.

Lean

El modelo de desarrollo de software Lean tiene sus raíces en el enfoque de Toyota para hacer las cosas: cuando necesita cambiar algo, haga solo los cambios que aporten más VALOR, requiera la menor ESFUERZO (presupuesto) y tome solo el 30% del TIEMPO planeado . Tal enfoque ayudó a Toyota a construir un flujo de trabajo capaz de cambiar sus transportadores de construcción de automóviles para que produjeran otro modelo de vehículos Toyota en apenas unas horas, mientras que los otros fabricantes necesitaban semanas para hacerlo.

Ventajas	Desventajas
El MVP se entrega rápidamente	La documentación debe ser absolutamente precisa y se necesita un analista experto para garantizar que el 100% de los requisitos se comprenda.
Los gastos son bastante bajos.	Este enfoque es adecuado solo para desarrolladores altamente calificados con inmenso conocimiento en el campo; aprender sobre la marcha es imposible y no es aceptable, ya que pone en riesgo el proyecto.
El equipo está motivado para hacer que cada Feature del producto sea perfecta, no simplemente para realizar las tareas.	Si bien están muy motivados para entregar un producto pulido, los desarrolladores pueden perder el enfoque en los objetivos iniciales, por lo que el líder del equipo o el analista deben respaldar la toma de decisiones flexible de los desarrolladores con un monitoreo diligente de los procesos.

Tabla 16 - Ventajas y desventajas de Lean

La metodología de desarrollo de software Lean funciona de maravilla para renovar un producto antiguo, como reescribir el motor utilizando las tecnologías más recientes, reemplazar la interfaz obsoleta con una funcionalidad de primera categoría, etc.

Kanban

Kanban Board es un enfoque que se puede utilizar realmente sobre cualquiera de las metodologías de desarrollo de software anteriores, aunque originalmente se derivó de trabajar bajo el modelo Lean. Se concentra en señalar la Feature más importante en el desarrollo en este momento, muestra el esfuerzo que ya se ha realizado y ayuda a

resaltar el espacio para la mejora continua, ayudando a buscar y encontrar la perfección en cada característica del producto.

Ventajas	Desventajas
Limitar el trabajo en progreso (WIP) a las áreas más importantes para mantener la cantidad de cambio mínima en un momento dado y acelerar el proceso de desarrollo.	Las notas adhesivas no pueden predecir los marcos de tiempo, por lo que los proyectos prolongados que involucran meses de desarrollo benefician poco de este enfoque.
Casi cero inversiones en herramientas o capacitación adicional para su equipo de desarrollo.	Al simplificar el WIP, Kanban no es bueno para la planificación y puede ser completamente revisado por un modelo más adecuado para la planificación, como Scrum. Luego, de nuevo, Kanban funciona mejor que Lean o Scrum, simplemente muestra el flujo de desarrollo y ayuda a evitar los cuellos de botella.
Tan simple como las notas adhesivas y una pizarra, esta técnica de visualización ayuda a mantener la mano en el pulso del desarrollo del producto.	Como cada tarea del Tablero Kanban es igualmente importante, esta práctica no funciona en tiempos de problemas de mantenimiento, ya que no hay marcadores de urgencia y las tareas no se pueden priorizar.

Tabla 17 - Ventajas y desventajas de Kanban

Kanban es ideal para un equipo más pequeño que trabaja en el pulido de un producto, como ejecutar un sistema de inteligencia empresarial, donde se espera una mejora constante del sistema y genera valor.

ANEXO 2 – “Reglamento interno”

El equipo definió el siguiente reglamento interno antes de comenzar el cual especifica prácticas estrechamente alineadas con los objetivos del equipo y del proyecto, a tener en cuenta a lo largo de todas las etapas.

A continuación se listan las pautas de convivencia en general.

1. Los integrantes del equipo llevarán registro de las horas trabajadas que se utilizará de forma constructiva para ajustar desbalances en el equipo pero no se utilizará para la recriminación entre los integrantes.

2. Cada integrante se compromete a escuchar las críticas constructivas y a hacerle llegar al otro posibles mejoras que se le ocurran, desde tanto la forma de redactar a la comunicación, preferiblemente durante las instancias de retrospección.

3. Los integrantes se comprometen a recibir positivamente peticiones de cambio en los *Pull Requests* en sus repositorios remotos por parte del otro integrante. Ambos integrantes deben tener presente en todo momento que cualquier petición o sugerencia de cambio no tiene otro fin más que la excelencia de la calidad del producto y no debería ser instancia de discusión defensiva.

4. Los integrantes intentarán distribuir las horas de trabajo semanales de tal forma que no ocurran situaciones por ejemplo en la que el desarrollador que tenga asignada la parte *back-end* de una funcionalidad no la codifique hasta tarde en el *Sprint*, ocasionando que el desarrollador *front-end* no pueda probar su código con tiempo o no sepa cómo interactuar con la API porque no se ha decidido aún.

5. Los integrantes se comprometen a discutir las soluciones de desarrollo como primera tarea de un *Sprint*. Esta regla tiene como fin evitar el retrabajo por aspectos no considerados apropiadamente y mejorar la productividad.

6. Ambos integrantes deben estar de acuerdo con una solución antes de que cualquiera de los dos comience a escribir la primera línea de código.

7. El equipo debe actuar en todo momento recordando que “dos cerebros piensan mejor que uno”. Ningún integrante debe sentir o tener la presión de tener que tomar una decisión importante solo. Se debe utilizar el proyecto como una oportunidad para mejorar las habilidades de trabajo en equipo y confianza en el conocimiento y capacidad del otro.

8. Los integrantes se comprometen a escuchar soluciones sin juzgar y a proponer soluciones aunque no hayan sido totalmente procesadas. La idea es que el equipo trabaje en conjunto para descartar, validar y perfeccionar soluciones juntos.

9. Cualquier cambio o evento importante que ocurra relacionado con, desde el cliente o el tablero de historias de usuario hasta el código o la documentación, deberá ser expresado claramente al otro integrante y de ser apropiado, validado, discutido y documentado.

10. Cariño. El equipo se compromete a trabajar en el proyecto final con cariño por el trabajo propio, respeto por el del otro, e intentando que no se transforme solamente en una obligación o carga. El equipo debe recordar que este proyecto es una de las primeras oportunidades que tiene de probar las habilidades de ingeniería que adquirió durante los cuatro años de carrera que ha elegido cursar.

11. Los integrantes, que saben que el proyecto de fin de carrera es normalmente una instancia estresante, se comprometen a apoyarse mutuamente y a salir adelante como equipo. Deben confiar en que, si siguen el proceso que definieron en la primera etapa, el proyecto podrá ser una instancia de disfrute y no de caos.

ANEXO 3 – “Gestión de la comunicación”

Con el fin de mantener una organización al momento de comunicarse con todos los involucrados del proyecto se definió un plan para la comunicación. El plan especificó los medios a utilizar para comunicarse entre el equipo, con el tutor y con los clientes. El mantener un criterio al decidir el medio de comunicación evitó situaciones de pérdida de información por ejemplo por no saber en qué medio se registró o a quién se incluyó en la conversación.

Comunicación interna

1. Los integrantes acordaron intentar que la mayoría de la comunicación relacionada al proyecto sea presencialmente. Esto no fue un problema ya que trabajaron juntos la mayor parte del tiempo.

2. Como los dos integrantes del equipo ya contaban con un equipo privado creado en Slack [7] utilizado para otras materias que había probado ser muy útil, decidieron definir canales de comunicación específicos para el proyecto en esta herramienta por donde sincronizarse en casos que no pudieron estar reunidos presencialmente. Se utilizaron principalmente para la coordinación de reuniones, recordatorios o notificaciones de prioridad baja, envío de imágenes, conversaciones cortas, etc.

3. Para la especificación de endpoints durante el desarrollo, el equipo definió decidirlos en conjunto antes de comenzar a desarrollar. Se compartió un equipo sincronizado en la herramienta Postman [35] donde se especificaron las rutas.

4. Se definió documentar los resultados de reuniones internas importantes. Por ejemplo, las tareas que surgieron de la discusión de la solución de una historia de usuario quedaron documentadas en la tarjeta en la herramienta de gestión que se utilizó. Las conclusiones de una retrospectiva se registraron junto con las otras actividades de la reunión de coordinación. Se pueden ver ejemplos de las mismas en el ANEXO 22 – “Reuniones de coordinación”.

Comunicación interna con el tutor

1. Los integrantes y el tutor se comprometieron a intentar que la mayoría de la comunicación relacionada al proyecto se diera presencialmente. Dudas que surgieron fueron enviadas por Whatsapp si constituían casos sencillos y concretos, o por email para dudas extensas y/o complejas, o envío de documentos.

2. El tutor fue invitado al repositorio remoto donde el equipo guardó los avances de documentación, a través del cual pudo visualizar avances y agregar comentarios. El equipo se comprometió a mantener el repositorio organizado de una forma a definir de antemano, en la cual sea fácilmente visualizable para el tutor los avances que se espera que revise.

3. El equipo inicialmente definió enviar actas de reunión luego de reuniones con el tutor, pero descontinuó esta práctica luego de discutirlo en una reunión de coordinación ya que se consideró que tomaba mucho tiempo y no aportaba más valor que un email de recapitulación con menos formato.

4. Se definió siempre incluir a ambos integrantes en los correos electrónicos enviados al tutor.

5. Para la comunicación por Whatsapp el equipo creó un grupo integrado por los estudiantes y el tutor. Se comprometieron a solamente hablar por el grupo y no de forma privada.

Comunicación con el cliente

1. Las reuniones con el cliente que el equipo consideró necesarias se realizaron presencialmente, o, de ser absolutamente necesario, por videoconferencia, herramienta a decidir.

2. Al finalizar una reunión se definió enviar una recapitulación o acta de reunión resumiendo de forma estructurada lo mostrado y hablado en la reunión. Se pueden ver ejemplos de las mismas en el ANEXO 23 – “Actas de reunión”.
3. Las actas de reunión, dudas del producto u otras conversaciones relacionadas al producto se llevaron por email. En su defecto, conversaciones cortas como coordinación de horarios pudieron ser habladas por un medio informal como WhatsApp.

4. El cliente técnico tuvo acceso al repositorio remoto donde se alojó el código fuente dado su conocimiento en las tecnologías e intención de aportar sugerencias en cuanto al código, pero no tuvo acceso a la documentación destinada a la entrega final. De ser necesaria la creación o exposición de cierta documentación al cliente, se crearía una carpeta en el repositorio remoto compartida de forma solo lectura donde se compartan solamente los archivos que el equipo considere.

5. Se definió siempre incluir a ambos integrantes en los correos electrónicos enviados al cliente. En casos de correos de gestión o recapitulación se procuró también enviar con copia al tutor para que pueda observar y aconsejar al equipo sobre la comunicación con el cliente.

6. Para la comunicación por Whatsapp el equipo creó un grupo integrado por los estudiantes y el cliente. Se comprometieron a solamente hablar por el grupo y no de forma privada.

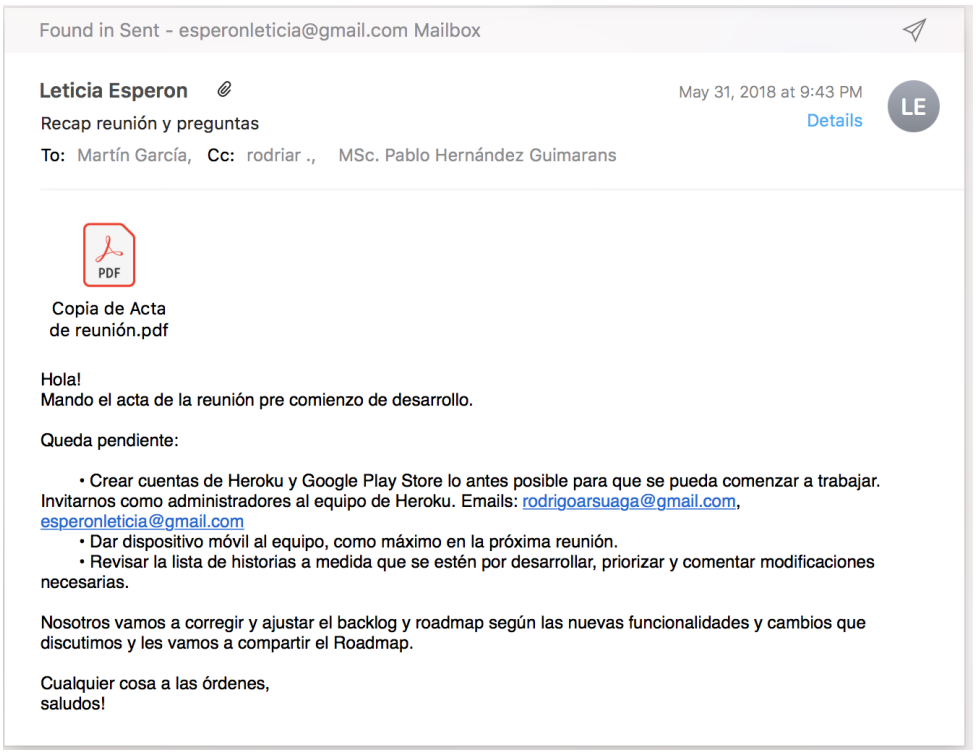


Ilustración 87 - Comunicación a través de correo electrónico por envío de acta de reunión

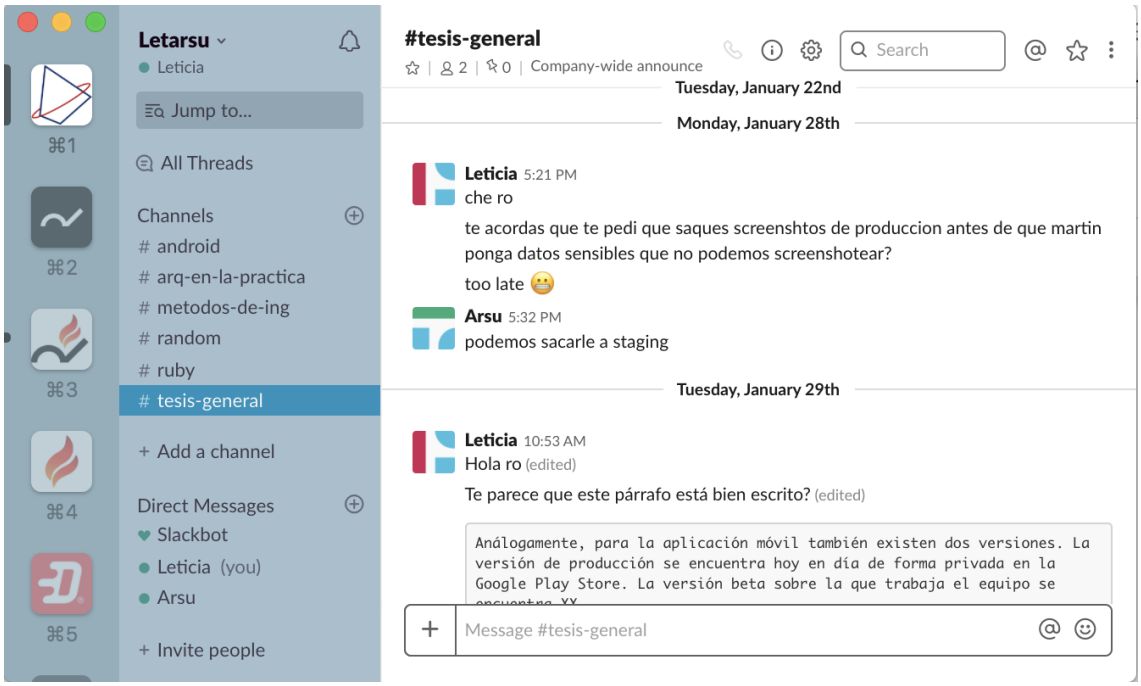


Ilustración 88 - Canales de comunicación interna a través de Slack

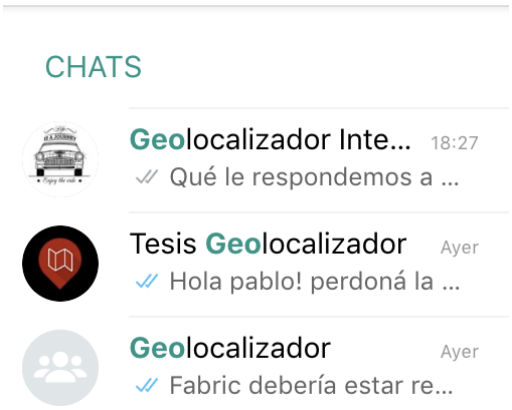


Ilustración 89 - Canales de comunicación en Whatsapp

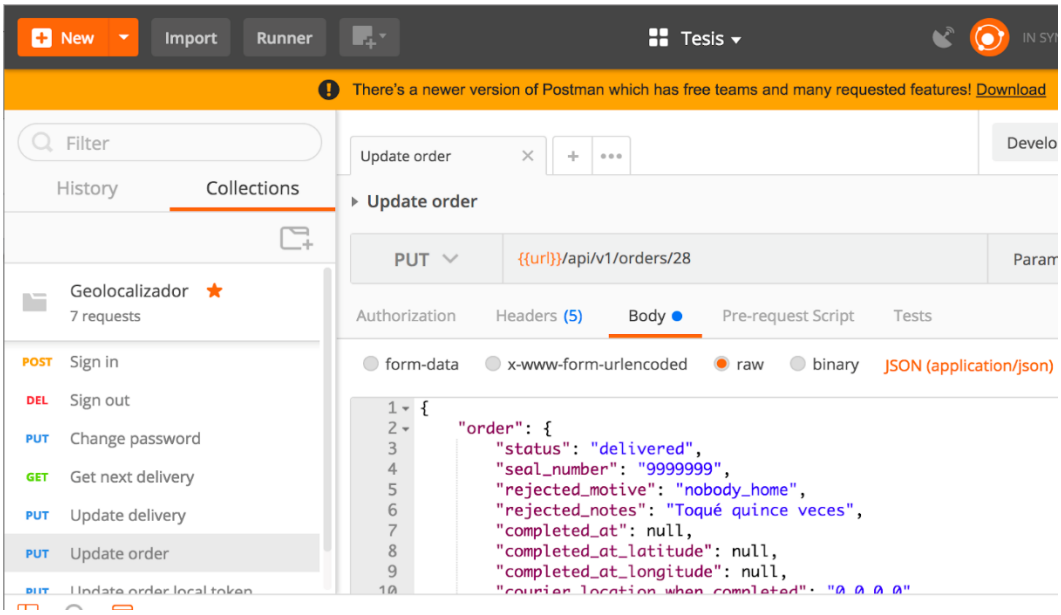


Ilustración 90 - Sincronización interna de la especificación de la API utilizando Postman

ANEXO 4 – “Resumen de herramientas utilizadas”

Herramienta	Descripción	Utilización
	Herramienta de diseño utilizada para la prototipación	Prototipación
	Herramienta que permite la creación y manejo de *Boards* que contienen distintas columnas con U*ser Stories*	Gestión y seguimiento de las tareas relacionadas a requerimientos y defectos del sistema
	Herramienta que permite el trackeo de horas permitiendo etiquetar y generar reportes semanales, mensuales, etc.	Seguimiento de las horas
	Herramienta de mensajería dentro de varios canales separados por temática	Comunicación interna del equipo
	Herramienta de administración de tareas que permite agrupar, etiquetar, priorizar y completar TO-DOs.	Sincronización de tareas internas al equipo
	Herramienta utilizada para la creación, edición, versionado y gestión de varios tipos de documentos entre varios usuarios en la nube	Almacenamiento de la documentación. Redacción de documentos iniciales, hojas de cálculo, presentaciones.
	Editor de texto con altas capacidades para el manejo de índices, referencias, formato, etc.	Recopilación final de la documentación
	*Platform as a Service* para desplegar la aplicación	Alojamiento del servidor
	Permite la creación de repositorios de código y el manejo de su versionado	Almacenamiento del código fuente

circleci	Es una plataforma de integración continua que se integra a los repositorios de Github, permitiendo visualizar el estado de las pruebas unitarias y herramientas de análisis de código.	Integración continua del back-end.
CODEBEAT	Permite evaluar la calidad del código según estándares y buenas prácticas.	Medir objetivamente la calidad del código
logentries™	Permite al equipo el monitoreo, filtrado, almacenamiento y reporte de los logs del servidor	Manejo de logs del servidor
Rollbar	Centraliza, reporta y notifica al desarrollador de ocurrencias de errores en el servidor.	Identificación de errores en el servidor
fabric	Herramienta que reporta incidentes en la aplicación móvil otorgando los datos necesarios para poder identificar su causa	Monitoreo de incidentes de la aplicación móvil
draw.io	Permite al equipo generar diagramas de todo tipo para la documentación	Creación de diagramas para la documentación
POSTMAN	Cliente HTTP que permite crear un listado con todas las consultas, clasificarlas, titularlas y compartir la colección entre usuarios	Pruebas manuales de integración de la API. Especificación de la API

Tabla 18 - Resumen de herramientas utilizadas

ANEXO 5 - “Resumen de justificaciones de herramientas utilizadas”

Github

El equipo decidió utilizar GitHub*,* cliente de *git*, para el versionado del código permitiendo mantener el repositorio remoto, hacer revisiones de código*,* seguir el historial de cambios y muchas otras funcionalidades útiles al proyecto. Se utilizó *GitHub* y no alternativas como Bitbucket, principalmente por motivos de comodidad y gustos personales de los integrantes.

CircleCI

CircleCI es una plataforma de integración disponible para varios lenguajes y que se integra a los repositorios de Github. El equipo decidió configurar CircleCI en el repositorio del *back-end* para que corra automáticamente las pruebas y análisis de calidad de código en cada nueva *Pull Request* y cada vez que se integran dos ramas, permitiendo visualizar rápidamente si hay alguna prueba o regla fallando, con el fin de no integrar código con defectos o que no cumpla la calidad esperada.

Trello

El equipo decidió utilizar Trello que es una herramienta de colaboración que organiza los proyectos en tableros. Trello fue seleccionado para mantener el tablero de Scrum debido a que es fácil de utilizar, aspecto fundamental considerando que el cliente también lo manipularía. El equipo consideró que las funcionalidades que brinda la versión gratis del producto serían más que suficientes para el fin con el cual se lo utilizaría, siendo así otras alternativas como Jira o Asana innecesariamente complejas.

Heroku

El equipo decidió utilizar Heroku como servidor remoto dada sus capacidades como PaaS (*Platform as a Service*). Heroku es una plataforma en la nube que permite a los desarrolladores crear, distribuir, monitorear y escalar aplicaciones sin preocuparse por muchos aspectos de infraestructura como equilibradores de carga, estado de los servidores particulares, ni siquiera del sistema de archivos, generalmente alcanzando un simple *git push* para desplegar toda una aplicación.

Google Calendar

El equipo seleccionó *Google Calendar* para coordinar reuniones con los clientes y con el tutor, permitiendo a todos encontrar y recordar más fácilmente los horarios de las reuniones. Los encuentros exclusivos entre los desarrolladores no se continuaron agendando mediante ninguna herramienta ya que con la coordinación hablada fue más que suficiente, considerando además que ambos integrantes estaban naturalmente juntos la mayor parte del tiempo.

Google Hangouts

El equipo eligió Google Hangouts como herramienta para las reuniones no presenciales dado que se integra con Google Calendar, es fácil de utilizar, permite compartir pantalla y acceder a las llamadas fácilmente desde un *link* o del calendario.

WhatsApp

Para casos de conversaciones donde se requieren respuestas rápidas, el equipo decidió comunicarse entre sí, con el cliente y con el tutor a través de WhatsApp. Se procuró utilizar esta herramienta solamente para preguntas concretas y avisos rápidos como se detalla mejor en el Reglamento Interno.

Email

Las recapitulaciones y dudas más extensas tanto con el tutor como con el cliente se enviaron a través de correo electrónico para mantener la formalidad.

Google Drive

El equipo decidió utilizar Google Drive para el manejo de documentos debido a su capacidad para compartir archivos compatibles con cualquier ordenador con acceso a Internet. También permite la edición por múltiples personas en simultáneo en tiempo real y otorga una alta capacidad de almacenamiento e historial de la cual se hizo mucho uso a lo largo del proyecto. También permite acceder a los documentos desde cualquier dispositivo móvil iOS o Android y permite el manejo de no sólo archivos de texto, sino también tablas de cálculo, diagramas y presentaciones.

Otra funcionalidad atractiva de Google Drive que resultó ser muy utilizada por los integrantes fue la posibilidad de realizar comentarios en ciertas líneas y el chat.

Se consideró que Google Drive resultaría especialmente útil para las primeras fases del proyecto, donde los documentos se realizan por separado y se corrigen y revisan muy frecuentemente. El equipo quedó abierto a, una vez en la última etapa de documentación, recopilar y unir los documentos en un archivo final posiblemente en otro formato como Word o Látex con funcionalidades de edición de texto más avanzadas.

Un problema de Google Drive que terminó siendo un poco molesto, es la necesidad de conexión a Internet, con la cual no siempre se contó al trabajar en la ORT.

WunderList

Wunderlist permitió al equipo gestionar y compartir tareas que no están relacionadas a las historias de usuario y que deberían ser transparentes al cliente. Agrega más funcionalidades al listado de tareas de Apple, como sincronización entre usuarios, priorización, seguimiento, etc., y además es gratis.

Google Sheets

Se utilizó Google Sheets para realizar los diagramas y gráficas ya que es fácil de utilizar y permite sincronización en la nube.

Toggl

Toggl fue la herramienta utilizada para el seguimiento de horas trabajadas por cada integrante. Es una herramienta web de uso gratis, sencilla y concreta que además tiene una extensión de Chrome para hacer más fácil aún el comenzar y finalizar una tarea. Toggl permite registrar las horas trabajadas por cada integrante del equipo en diferentes tareas y visualizar estadísticas básicas sobre la distribución del trabajo a lo largo del tiempo y entre los integrantes.

Adaptarse a Toggl normalmente requiere un tiempo ya que los usuarios suelen olvidarse de “encenderlo” o “apagarlo”, pero dado a que ambos integrantes ya estaban familiarizados con esta herramienta por su experiencia laboral, el hábito se implantó muy rápidamente.

Fabric

Fabric permite analizar con facilidad el estado de la aplicación móvil dándole al equipo estadísticas importantes como la cantidad de dispositivos en uso y la cantidad de *crashes* lo cual permite al usuario encontrar defectos de la aplicación que ocurrieron mientras se completaban pedidos. Se eligió porque es la mejor alternativa gratuita.

Logentries

Logentries le dio al equipo la capacidad de monitorear el *back-end* con facilidad ya que esta herramienta provee un mejor acceso a los *logs*, facilitando la retención, centralización, filtrado y consulta de los mismos. Se eligió porque es la mejor alternativa gratuita.

Rollbar

Rollbar ayuda a realizar un seguimiento eficaz de errores y problemas que ocurran en el servidor, permitiendo agrupar, resumir, reportar y notificar de errores nuevos o reincidencias. Se está utilizando en el plan de prueba.

Draw.io

Draw.io es una herramienta para crear aplicaciones de creación de diagramas y la aplicación web de creación de diagramas más utilizada en el mundo. El equipo la utilizo para crear todo tipo de diagramas para la documentación. Se eligió porque es la mejor alternativa gratuita.

Postman

Postman ayuda a construir un flujo de trabajo para el desarrollo de API permitiendo tener una documentación de la misma y probar todos los *endpoints* con facilidad. Fue

clave durante todo el proceso de desarrollo para poder probar continuamente el programa y sincronizar entre los desarrolladores.

ANEXO 6 – “Prototipación”

En esta sección se describen los prototipos realizados de tanto la aplicación móvil como la web y los principales resultados obtenidos de ellos.

Los primeros prototipos realizados fueron de la aplicación móvil y se dibujaron en papel junto con el cliente.

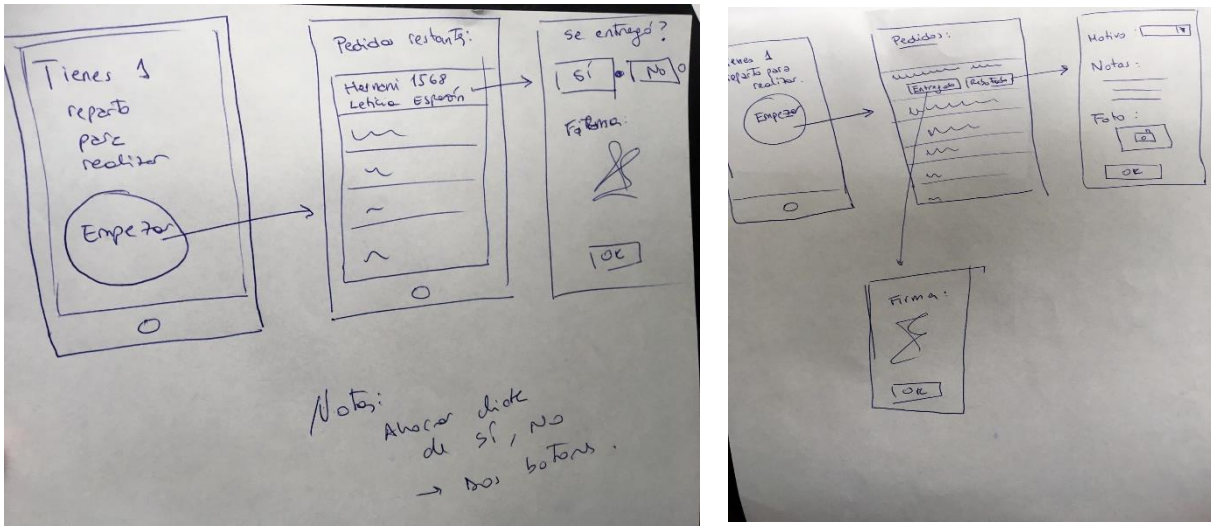


Ilustración 91 - Primeros prototipos en papel realizados en conjunto

Con ellos se afirmó la idea de que el repartidor debe visualizar la lista de los pedidos a entregar, e irlos marcando como completado o rebotado.

Con esta idea en mente, el equipo procedió a realizar prototipos de la aplicación móvil en Invision, herramienta que permite relacionar pantallas haciendo que clicks en ciertos botones invoquen la siguiente pantalla, simulando funcionar.

Primero se prototipó el flujo principal, y luego otros requerimientos menos importantes como gestión de la sesión.

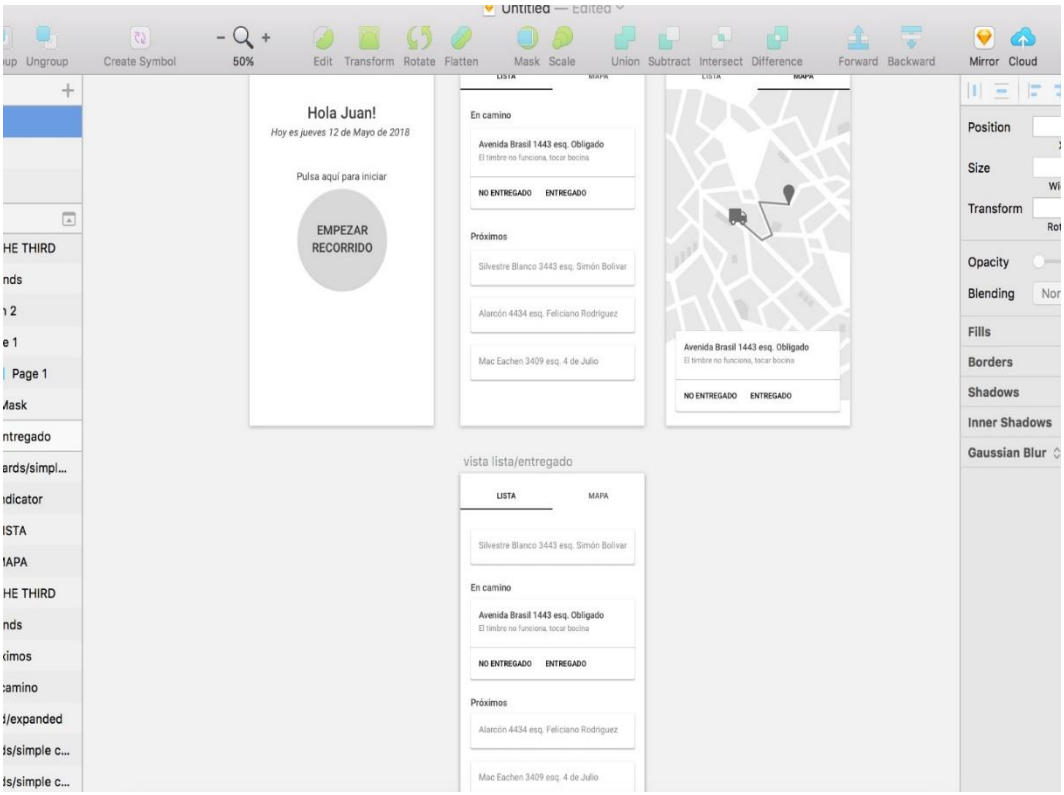


Ilustración 92 - Sincronización interna de la especificación de la API utilizando Postman

A continuación se muestran algunas de las pantallas prototipadas.

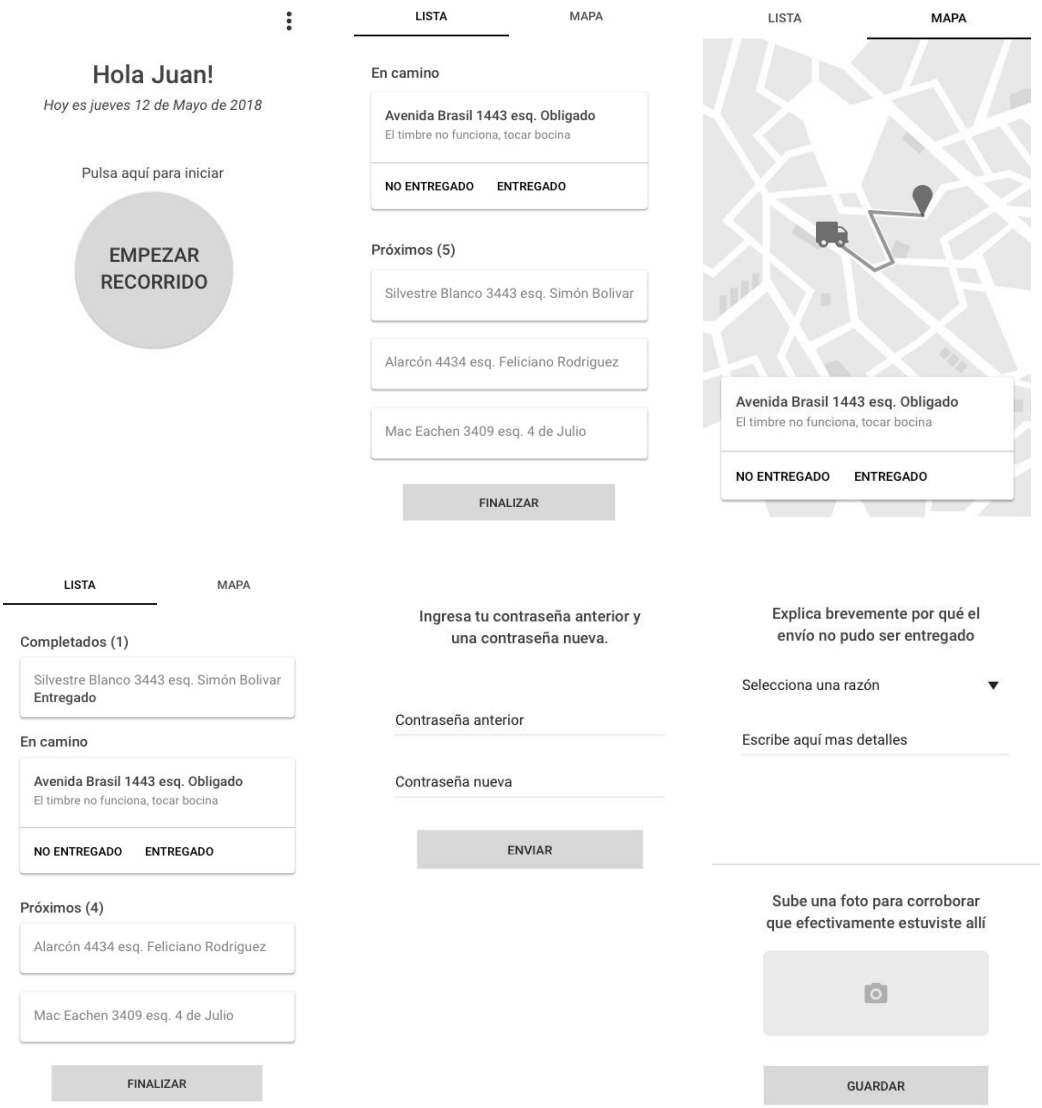


Ilustración 93 - Prototipos iniciales de la aplicación móvil

Por otro lado, también se realizaron prototipos de algunas funcionalidades de la aplicación web, para los cuales se utilizó la misma herramienta utilizada para el maquetado final de la interfaz del administrador, ActiveAdmin. Los prototipos realizados con ActiveAdmin no funcionaban como un simulador pero permitieron generar los diseños para que el usuario realizara comentarios y sugerencias.

A continuación se muestran alguna de las pantallas de la aplicación web prototipadas.

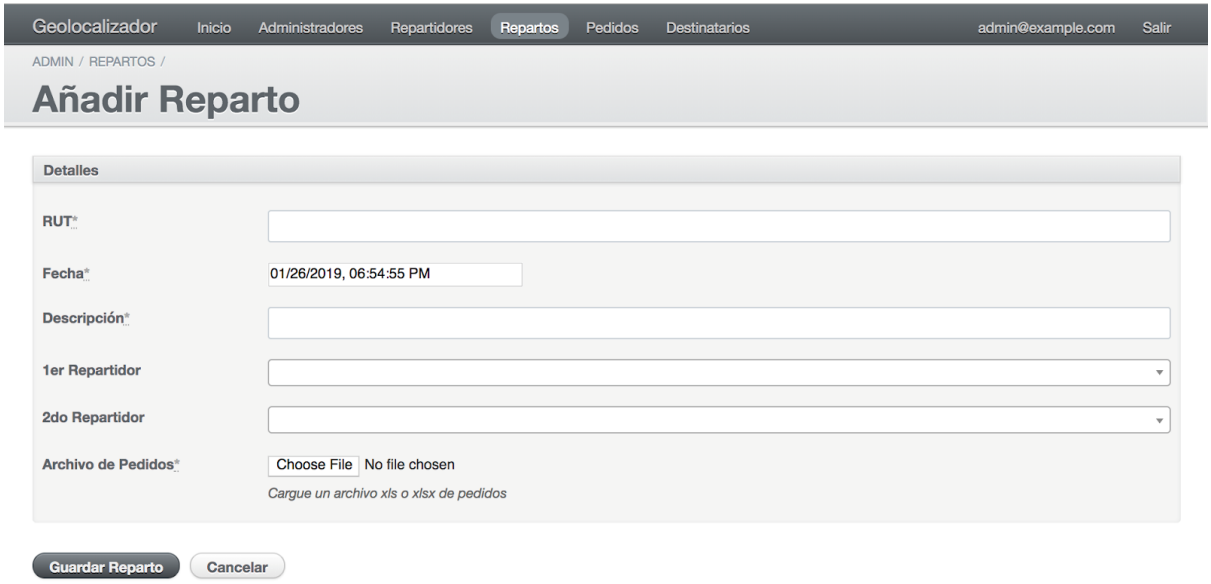


Ilustración 94 Prototipo de la funcionalidad de crear reparto

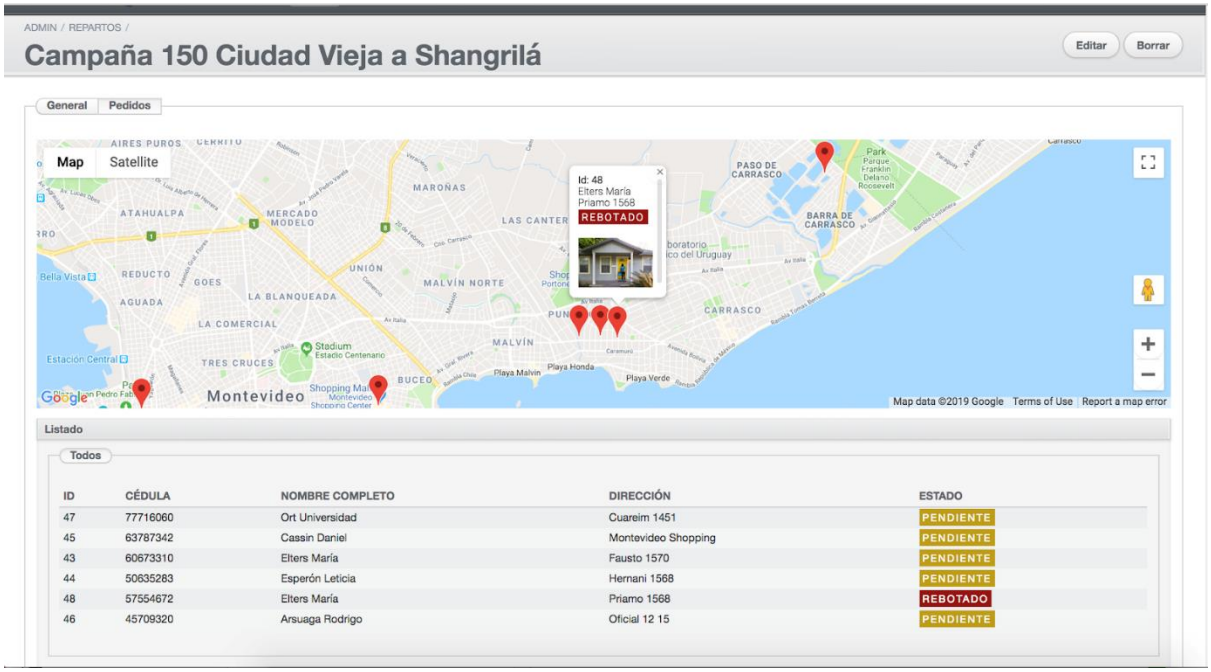


Ilustración 95 - Prototipo de la funcionalidad de ver pedidos de un reparto

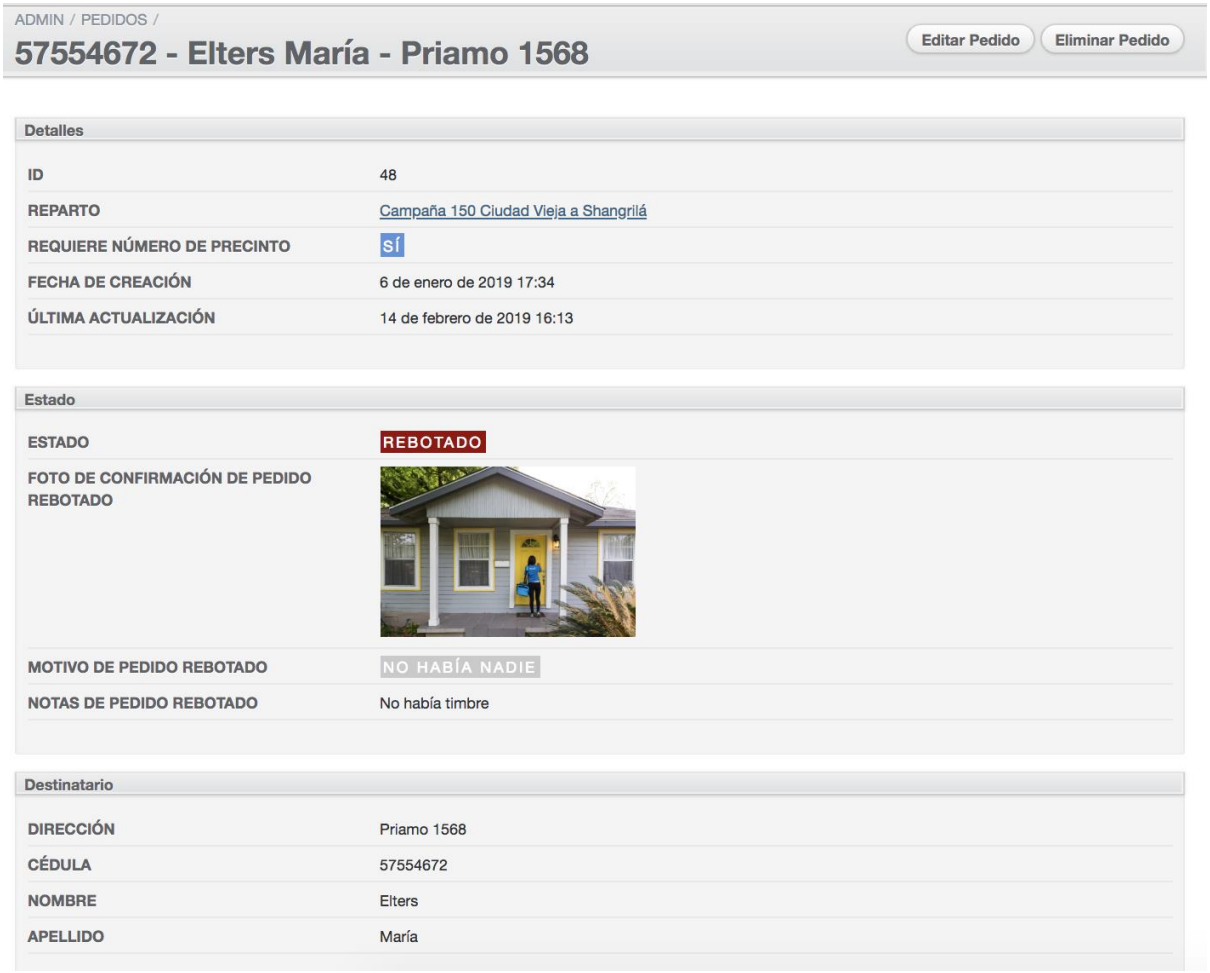


Ilustración 96 - Prototipo de la funcionalidad de ver detalles de un pedido completado

Luego de la validación de los prototipos anteriores, se procedió a realizar los diseños de la aplicación móvil con ayuda de una diseñadora gráfica amiga del equipo que tuvo la solidaridad de ayudarlos. La diseñadora procuró trabajar sobre los prototipos, teniendo en cuenta las correcciones que el equipo había hecho sobre los prototipos originales. Los diseños que realizó la diseñadora son iguales a los que se utilizan hoy en la aplicación, con alguna funcionalidad menos que se agregó sobre el camino.

A continuación se muestra cómo se visualiza el panel de Invision con los diseños finales y la aplicación en modo simulador.

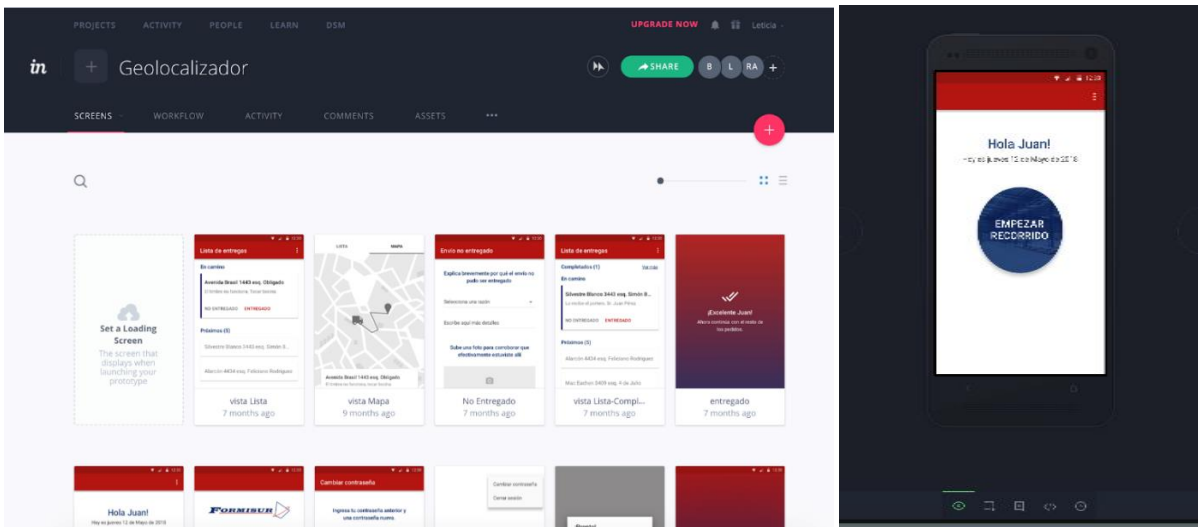


Ilustración 97 Utilización de Invision para prototipar y simular

ANEXO 7 – “Resultado de pruebas de los prototipos con usuarios”

Los prototipos construidos se presentaron a los usuarios con quienes se validaron.

En el caso de la aplicación móvil se pidió utilizar la aplicación como si se estuviera en un reparto, intentando completar los pedidos. Los integrantes del equipo observaban al usuario realizar las acciones sobre el prototipo, identificando errores y dudas. Al finalizar se le pedía su opinión y sugerencias.

A continuación podemos visualizar a Rodrigo observando a Nicolás Benítez (Administrador de Formisur y ocasionalmente repartidor) interactuar con los prototipos.

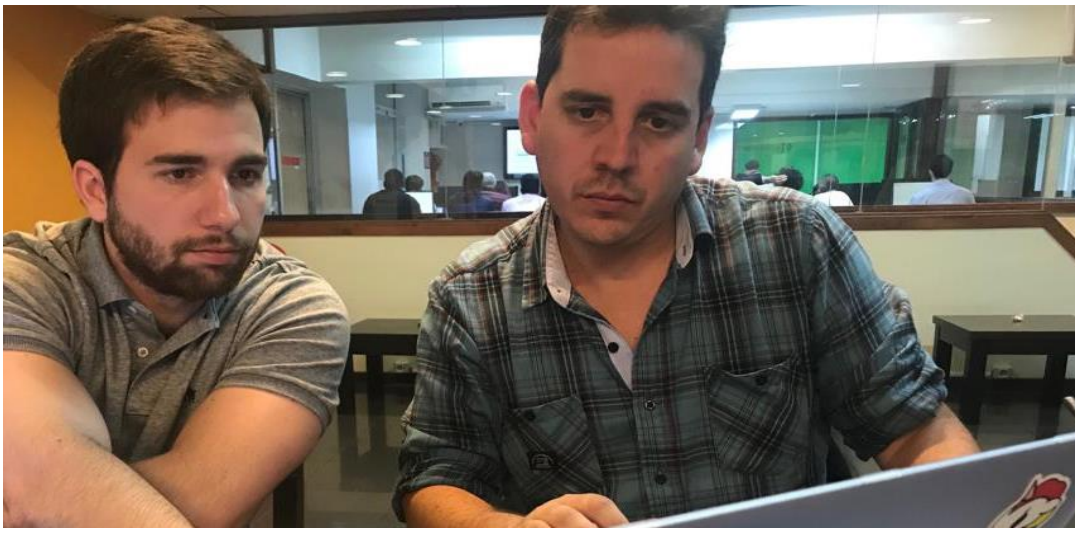


Ilustración 98 - Pruebas de usabilidad utilizando los prototipos

A continuación se describen las principales conclusiones obtenidas de los prototipos de papel, móvil y web.

Pantalla	Conclusiones y sugerencias
Prototipo en papel	
Lista de pedidos	• Recordar siempre minimizar la cantidad de clicks y pantallas • Simplificar la interfaz lo más posible • Letras y botones grandes para utilizar en la calle • Utilizar lenguaje claro y conciso que usen los repartidores.

Prototipo móvil v1	
Iniciar reparto	• Todo ok. • Si no hay repartos el botón no debería estar y debería decir que no hay ningún reparto para realizar hoy.
Lista de pedidos	• No se entiende que entregado y rebotado son botones • El botón de finalizar no debería estar • No se precisa ver la lista de pedidos completados • Sí es útil visualizar la cantidad de pedidos completados y pendientes
Mapa	• No debería tener tanto foco. Es una funcionalidad que no van a usar mucho. Se acordó hacer que el mapa se abra con un botón al lado de los botones de entregado y rebotado, en lugar de tomar una pestaña entera que puede desorientar a los repartidores. • Utilizar el término "pedido" en lugar de “envío"
Vista de rebotado	Todo ok
Prototipo móvil v2	
Iniciar reparto	Todo ok
Lista de pedidos	• Cambiar los colores de los botones entregado y rebotado • Incluir el nombre del destinatario
Vista de rebotado	• Al terminar de marcar como rebotado, mostrar una notificación que se vaya sola en vez de una pantalla entera o de una notificación que deba aceptarse para esconderse, para ahorrar tiempo al repartidor.
Prototipo aplicación web	
Crear reparto	• Se debería poder no seleccionar ningún repartidor al principio • No se entendió que se tenía que seleccionar un archivo Excel, poner un comentario o nota
Ver listado de repartos	• No fue fácil entender cómo ir a editar un reparto, agregar links de las acciones en el listado mismo • Agregar filtro por rut de cliente en la página de listado de repartos • Agregar filtro por fechas • Incluir cantidad de pedidos del reparto en los datos del listado

	• No incluir los repartos eliminados dentro de la lista de todos los pedidos, separarlo en otra tab
Ver datos de un reparto	• En la lista de pedidos agregar un resumen del total, cantidad de entregados, rebotados, etc. • Falta pensar cómo se van a mostrar las alertas
Ver datos de un pedido	• El administrador no entendió la diferencia de fecha de creación y fecha de la última actualización.

Tabla 19 - Resultados de pruebas con usuarios utilizando los prototipos

ANEXO 8 – “Pruebas con los usuarios en repartos reales”

Durante el proyecto hubo dos temporadas de pruebas en repartos reales, una al finalizar cada *Release* planificado. En la primera el software estaba en una versión muy inmadura pero tenía todas las funcionalidades principales. Como comentario, la funcionalidad de enrutamiento no entró en la primera prueba porque no se había terminado a tiempo ya que se subestimó. Sí entró en la segunda prueba. Para visualizar qué funcionalidades entraron en cada Release se puede referir al ANEXO 30 – “Release Plan” considerando que las pruebas fueron al final de los *Sprint*s 8 y 14.

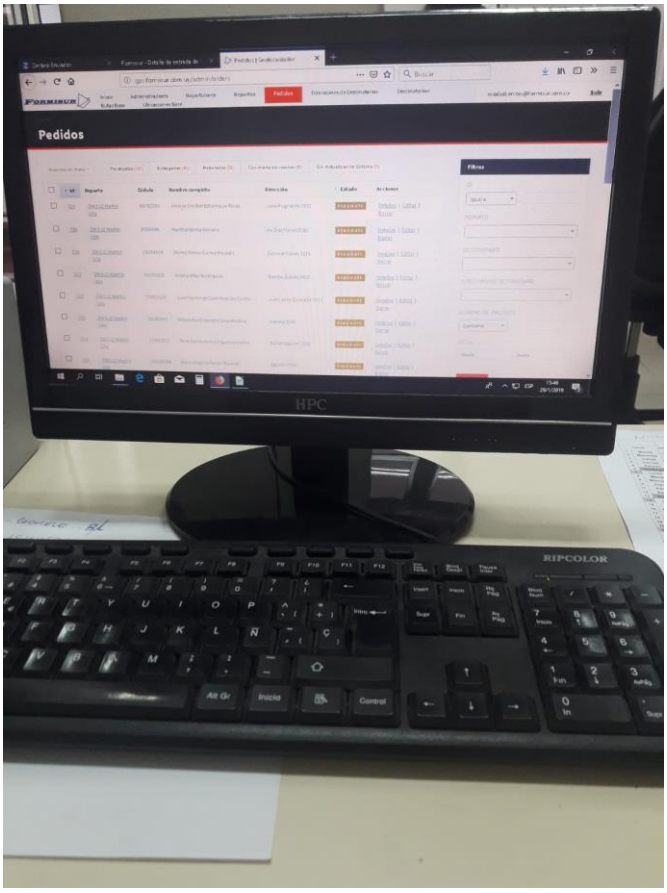
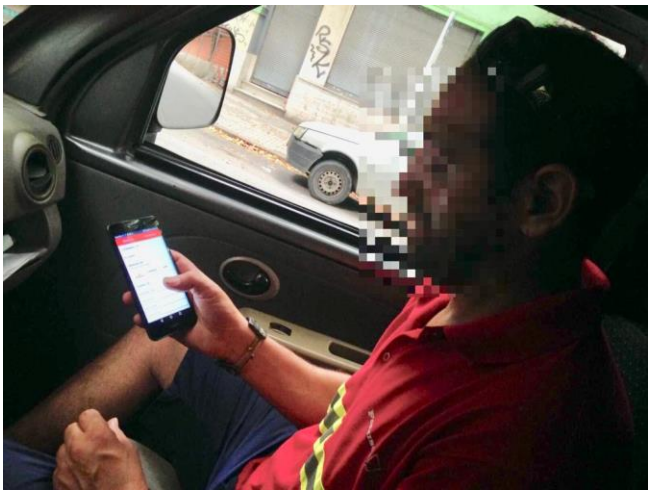


Ilustración 99 - Aplicación del administrador utilizándose en Formisur



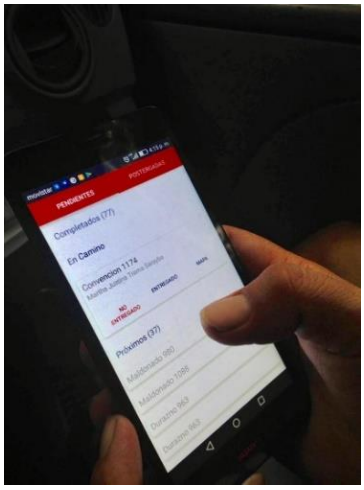


Ilustración 100 - Repartidores utilizando Geolocalizador en un reparto

Los repartos de prueba fueron planificados con anticipación. El cliente solicitó a algunos repartidores utilizar la aplicación en algunos repartos de menor importancia.

El administrador comenzó planificando los repartos el día anterior para mayor seguridad. En la primera prueba lo realizó con la presencia del equipo para asegurarse de entender todo correctamente. El equipo utilizó esto como otra instancia de prueba de usabilidad igual que la que se hizo con los prototipos. El administrador no tuvo problema en programar el reparto, quizás porque ya conocía la interfaz, por lo que el equipo se sintió satisfecho. Sí se hicieron algunas sugerencias de cambio como se describirá a continuación.

Por otro lado no se pudo observar a los repartidores utilizando la aplicación final porque no se consideró posible que el equipo acompañara a los mismos ya que estos tenían lugar y tiempo limitado. Es por eso que el equipo se encargó de instalar herramientas que permitan monitorear el uso de la aplicación para conocer cómo utilizaron la aplicación.

Durante el reparto se detectaron defectos, mejoras pequeñas, y sugerencias de cambio o de nuevas funcionalidades.

Aplicación	**Resultados**
PRUEBAS MVP	
Aplicación web	Defectos
	No se encontraron
	feedback

	• Sería útil poder filtrar la lista de pedidos de un reparto para ver los pendientes de geolocalizar más fácilmente • Mostrar link a 'editar dirección de destinatario' en listado de pedidos del reparto • Sería útil que el admin pueda ver el tiempo total de un reparto • Sería útil poder visualizar cuántos pedidos tiene un reparto, agrupados por estado • Hay repartos que precisan registrar un número de precinto. Los repartidores los anotaban en un papel. El admin debería seleccionar si el pedido requiere número de precinto al entregarse. • Restringir resultados de geolocalizaciones a Uruguay • Sería útil poder visualizar cuántos pedidos tiene un reparto, agrupados por estado
Aplicación móvil	Defectos
	• Se rompe la app en los repartos de más de 110 pedidos • A veces se pedía rebotar el mismo reparto muchas veces • Repartidor queda trancado si se borra una orden durante el reparto • Si se elimina un pedido en un reparto en marcha, no se sincronizan los pedidos a partir de él
	feedback
	• Ocurrieron casos donde el destinatario avisó que no iba a estar en su casa de mañana. El repartidor debería poder postergar un pedido para dentro de un rato. • Ocurrió que el repartidor se tuvo que desviar y pasó justo por la casa de otro destinatario que estaba más adelante en la lista de pedidos. Se debería poder buscar un pedido y completarlo fuera de orden • No permitir borrar pedidos de un reparto en marcha, porque es complejo asegurarse que el repartidor se encuentra con conexión como para enterarse del cambio
PRUEBAS V1	
Aplicación web	Defectos
	• No se encontraron
	feedback
	• El cuadrito con información del pedido en el mapa debería ser más grande para que entre la foto sin hacer *scroll*
Aplicación móvil	Defectos
	• Si posterga hacia la derecha la aplicación deja de funcionar

	feedback
	• Solo pedir foto de confirmación si el motivo es 'nadie en casa' • Mejorar la calidad de las fotos de confirmación • Poner un globito rojo con la cantidad en la pestaña de postergados

Tabla 20 - Resultados de pruebas con usuarios durante repartos reales

Los defectos y *feedback* reportados en la segunda prueba fueron muchos menos dado al esfuerzo previo del equipo a testear ampliamente la aplicación y arreglar lo máximo posible antes de la última prueba. Los defectos y mejoras encontradas durante la prueba se arreglaron en el *Sprint* 15 junto con otras sugerencias de mejora de otros *Sprints*.

ANEXO 9 – “Product Backlog”

A continuación se encuentra la lista de requerimientos funcionales y no funcionales en formato de historias de usuario implementadas. No se incluyen los defectos ni las sugerencias sobre las que se trabajó.

User Stories
Ícono de la app y pantalla de inicio
Como admin debo poder loguearme a la web con usuario y contraseña
Como admin debo poder crear usuarios de la app para los repartidores
Como admin debo poder ver repartidores
Como admin debo poder filtrar los repartidores
Como repartidor debo poder loguearme
Como repartido debo poder cerrar sesión
Como admin debo poder crear un reparto
Como admin debo poder ver la lista de repartos
Como admin debo poder ver los datos de un reparto
Como admin debo poder ver los datos de un pedido
Como admin debo poder ver la lista de pedidos
Como admin debo poder filtrar repartos
Como admin debo poder filtrar pedidos
Como admin debo poder administrar direcciones de destinatarios
Como admin debo poder filtrar las correcciones de direcciones
Como admin debo poder borrar y recuperar un reparto
El reparto debe crearse con las direcciones de destinatarios arregladas

Como admin debo poder recalcular un reparto cuando corregí las direcciones de los destinatarios
Como repartidor debo poder iniciar el próximo reparto
Como repartidor en un reparto debo poder ver la próxima dirección en mi pedido
Como repartidor debo poder navegar hacia la próxima dirección
Como repartidor debo poder marcar un pedido como rebotado
Como admin debo poder ver los pedidos rebotados
Como repartidor debo poder marcar un pedido como entregado
Como admin debo poder ver los pedidos entregados
Como repartidor debo poder finalizar mi reparto automáticamente si terminé el último pedido
Como repartidor debo poder finalizar mi reparto automáticamente si terminé el último pedido
Como repartidor mi app debe funcionar aunque no tenga conexión, y debe actualizarse apenas la consiga
El sistema debe ordenar los paquetes de un reparto de forma de tener el camino más rápido
El enrutamiento automático de órdenes debe poder desactivarse para todos los futuros repartos desde una variable de entorno
Como admin debo poder visualizar los pedidos rebotados con alertas
Como admin puedo ver el tiempo total de un reparto
Como admin debo poder hacer que un pedido requiera número de precinto al entregarse
Como admin debo poder visualizar la ubicación del repartidor en el momento en que marcó un pedido como completado
Como admin debo poder resolver una alerta
El sistema debe notificar a otro sistema externo cuando un pedido es completado
Como admin debo poder visualizar rápidamente la última ubicación conocida del repartidor
Como repartidor debo poder postergar un pedido en un reparto
Como repartidor debo poder buscar un pedido y completarlo fuera de orden

Como admin debo poder visualizar cuántos pedidos tiene un reparto, agrupados por estado
Como admin debo poder filtrar la lista de pedidos de un reparto para ver los pendientes de geolocalizar más fácilmente
Como admin debo poder eliminar y recuperar repartidores
Como repartidor debo poder cambiar mi contraseña
Como repartidor debo poder deslizar derecha o izquierda para postergar un pedido
Como nuevo repartidor estoy obligado a cambiar mi contraseña la primera vez que entro
Como admin debo poder cambiarle la contraseña a un repartidor desde la web
Como admin debo recibir instrucciones para resetear mi contraseña si me la olvidé
Como admin debo poder visualizar un resumen de los datos (Dashboard)
Integración con Routeme para resolver el problema de enrutamiento (Segunda implementación)
El admin debe poder seleccionar la ubicación de origen y destino de un reparto
El sistema debe recordar el enrutamiento de un reparto para no tener que volver a calcularlo en un reparto igual en el futuro
Como admin debo poder visualizar el recorrido planificado para mi reparto automáticamente ruteado
El sistema debe reintentar automáticamente enviar para enrutamiento automático si pasaron más de 6 horas sin respuesta
Como admin debo poder elegir si un reparto se debe enrutar automáticamente o no al crearlo
Como admin debo poder monitorear el recorrido de un reparto

Tabla 21 - Product Backlog

ANEXO 10 – “Análisis de tecnología back-end”

Al igual que ocurrió con *Java*, el cliente predefinió la utilización de Ruby on Rails para el desarrollo del *back-end* del sistema.

Los integrantes apoyaron la decisión del cliente porque la experiencia previa con la que uno de los integrantes contaba en esta tecnología le permitió comprobar la ventaja de la utilización del lenguaje en contextos como en el que se encontraban.

Uno de los principios fundamentales de Rails es *Convention Over Configuration*. El *Framework* parte de la base de que cada vez que un desarrollador tiene que preocuparse de la configuración del proyecto está dejando de invertir tiempo en el desarrollo en lo que realmente importa: la lógica de negocio. En general existe un patrón que se repite en la mayoría de los proyectos, entonces Rails asegura al programador que si mantiene la convención, el *Framework* interpretará y se encargará de la configuración por detrás. Ejemplo de esto es el mapeo automático entre el modelo, la tabla, el controlador y su vista. Rails estandariza y facilita muchas configuraciones permitiendo al equipo comenzar a entregar lo antes posible logrando un código altamente mantenible, siendo esto lo que el equipo buscaba.

Al programar el *back-end* en Ruby on Rails, se necesitó menos líneas de código que si se hubiera utilizado alguno de muchos otros lenguajes, también gracias a la cantidad de gemas (especie de librerías *Open Source*) que al integrarlas simplifican el desarrollo de muchas funcionalidades, permitiendo maximizar el alcance del proyecto en el mismo tiempo sin perder rendimiento en la escala en la que se planeó utilizar. Las gemas resultan de alta utilidad para todo tipo de aspectos, como autenticación, interfaces, trabajos asíncronos, borrado lógico, geolocalización, encapsulamiento de la integración con servicios externos, etc.

Finalmente, es un lenguaje para el cual existe mucho soporte en la web y se utiliza por muchas *start ups* del estilo del proyecto.

Por todas estas razones, sumadas al gran tamaño del proyecto y el pequeño tamaño del equipo, se acordó que el lenguaje sería apropiado y se apoyó la decisión del cliente.

ANEXO 11 – “Análisis de tecnologías de hosting”

Se realizó un análisis para seleccionar la tecnología utilizada para alojar la API y la aplicación web para el administrador, que serían desarrolladas en Ruby on Rails.

Se consideraron solamente alternativas en la nube, ya que, en primer lugar, la empresa no cuenta ni quería contar con servidores propios por todas las responsabilidades que eso implica, y además el equipo quería poder tener la flexibilidad de trabajo en la nube, sobre todo en un proyecto que recién comienza.

Se evaluaron alternativas IaaS y PaaS, específicamente los servicios de Amazon EC2 (IaaS) y los servicios de Heroku (PaaS).

Recordando las diferencias entre PaaS y IaaS, en PaaS se provee una plataforma que permite desarrollar, correr y administrar aplicaciones sin la complejidad de tener que diseñar y mantener la infraestructura, que es enteramente administrada y gestionada por el proveedor del servicio. Esto permite al desarrollador desentenderse de la configuración específica de la infraestructura, pudiendo modificar solamente algunos aspectos de la misma.

Por otro lado, en IaaS se permite configurar y utilizar bajo demanda recursos de almacenamiento, procesamiento y red, entre otros. Tiene la capacidad de poder configurar a más bajo nivel la infraestructura, permitiendo mejor adaptación a las necesidades de cada cliente.

A continuación se realizó un cuadro que compara información pertinente al proyecto del equipo:

	AWS	**Heroku**
Tipo de servicio	IaaS	PaaS
Soporta aplicaciones Ruby on Rails	Sí	Sí
Soporte en la web	Alto	Alto
Experiencia del equipo	Medio	Alto
Librerías útiles	No, solo algunos otros servicios de Amazon que son integrables pero en general difíciles de configurar. Ejemplo Cloudwatch, RDS, Redis, etc.	Muchas (Add-ons, por ejemplo de monitoreo, APM, bases de datos, caché, etc.)
Hosteado en	Sus propios servidores	Servidores de AWS
Costo	$0.013 por hora	Opción de Tier gratis mientras se desarrolla. USD 7 por dyno por mes Se precisan dos dynos por ambiente. Total: USD 21 luego de salir a producción
Opción de tier gratis	Sí pero la aplicación Ruby usa un poco más de memoria que la que se ofrece en el tier gratis, por lo que no es una opción.	Sí. Pero el servidor se “duerme" y puede demorar más en contestar luego de un tiempo. Apropiado mientras se desarrolla, no para producción.
Se precisa experiencia en DevOps	Sí	No
Auto scaling y escalamiento horizontal	Sí	Sí
Despliegue rápido	No	Sí
Soporte personalizado	Sí	No

Rollback	Sí	Sí
Desplegable con Github	No	Sí
Región geográfica más cercana	Brasil	USA

Tabla 22 - Comparación de tecnologías de Hosting

Ponderando los resultados, se eligió Heroku ya que es más apropiado para un proyecto nuevo que estará en construcción durante un año. El equipo quiere desplegar lo más rápido posible y estar tranquilos de que la infraestructura no es un problema. El precio en una aplicación que recién comienza es bajo en ambos servicios y el cliente accedió a ambos por lo que no se consideró un problema. Si el proyecto tiene éxito y la empresa se expande, el moverse a Amazon es posible.

ANEXO 12 – “Análisis de tecnologías de almacenamiento de datos”

Para almacenar los datos del servidor de Geolocalizador se tomaron en cuenta las opciones más frecuentes utilizadas con Ruby On Rails: MySQL y PostgreSQL.

A continuación se compara información pertinente al proyecto:

	PostgreSQL	**MySQL**
Experiencia del equipo	Alta	Media
Integración con Heroku (tecnología de hosting elegida)	Alta	Baja
Soporte en la web	Alta	Alta
Tipo	Relacional orientado a objetos	Relacional
Índices parciales	Sí	No
Tipo de datos Array	SÍ	No
Tipo de datos JSON	Sí	No
Claves primarias UUID	Sí	No

Tabla 23 - Comparación de tecnologías de almacenamiento

Las discusiones en la web de la performance o errores conocidos de cada una no llegan a nada concreto de si una es mejor que la otra. Ponderando los resultados, considerando que en general la comunidad de Ruby On Rails tiende a elegir cada vez más PostgreSQL sobre MySQL dado el soporte a las funcionalidades útiles que trae consigo PostgreSQL como son los tipos de datos Array y JSON, y considerando que PostgreSQL además se integra perfecto con Heroku con solo tres clicks, el equipo se decidió por esta opción.

ANEXO 13 – “API REST Geolocalizador”

A continuación se especifican los endpoints de la API provistas por el back-end para ser consumidos por la aplicación móvil de Geolocalizador.

Descripción	Verbo	Endpoint
Sign in de repartidor	POST	/api/v1/couriers/sign_in
Sign out de repartidor	DELETE	/api/v1/couriers/sign_out
Consultar el próximo reparto para el repartidor logueado.	GET	/api/v1/courier/delivery
Consultar los datos de un reparto en particular	GET	/api/v1/deliveries/:id
Actualizar el estado de un reparto	PUT	/api/v1/deliveries/:id
Actualizar el estado de una orden	PUT	/api/v1/orders/:id
Consultar los datos del repartidor logueado	GET	/api/v1/courier
Actualiza los datos del repartidor logueado	PUT	/api/v1/courier
Registrar la posición actual del repartidor logueado	POST	/api/v1/deliveries/:delivery_id/locations

Tabla 24 - Especificación de endpoints para el consumo de la aplicación móvil

A continuación se lista el *endpoint* provisto para los webhooks del servicio externo Routeme, el cual contacta a Geolocalizador para avisar cuando una ruta fue calculada.

Descripción	Verbo	Endpoint
Registrar ruta calculada	POST	/api/v1/routes

Tabla 25 - Especificación de endpoints para el consumo de Routeme

Dado a que los endpoints para la comunicación entre la aplicación web y el *back-end* son más de 70, se resumen en la siguiente tabla.

Descripción	Endpoint	Verbos disponibles
Root	/admin	GET
Dashboard	/admin/dashboard	GET
Manejo de la sesión del administrador	/admin/login	GET, POST
	/admin/logout	GET, DELETE
Manejo de la contraseña	/admin/password/new	GET
	/admin/password/edit	GET
	/admin/password	PUT, POST
Manejo de direcciones de destinatarios	/admin/recipient_addresses/batch_action	POST
	/admin/recipient_addresses	GET, POST
	/admin/recipient_addresses/:id	GET, PUT, DELETE
	/admin/recipient_addresses//new	GET
	/admin/recipient_addresses/:id/edit	GET
Manejo de ubicaciones base	/admin/base_locations/batch_action	POST
	/admin/base_locations	GET, POST
	/admin/base_locations/:id	GET, PUT, DELETE
	/admin/base_locations/new	GET
	/admin/base_locations/:id/edit	GET
Manejo de destinatarios	/admin/recipients/batch_action	POST
	/admin/recipients	GET, POST
	/admin/recipients/:id	GET, PUT, DELETE
	/admin/recipients/new	GET

	/admin/recipients/:id/edit	GET
Manejo de pedidos	/admin/orders/batch_action	POST
	/admin/orders	GET, POST
	/admin/orders/:id	GET, PUT, DELETE
	/admin/orders/new	GET
	/admin/orders/:id/edit	GET
	/admin/orders/:id/resolve_completed_location_alert	PUT
Manejo de repartos base	/admin/base_deliveries/batch_action	POST
	/admin/base_deliveries	GET, POST
	/admin/base_deliveries/:id	GET, PUT, DELETE
	/admin/base_deliveries/new	GET
	/admin/base_deliveries/:id/edit	GET
Manejo de repartos	/admin/deliveries/batch_action	POST
	/admin/deliveries	GET, POST
	/admin/deliveries/:id	GET, PUT, DELETE
	/admin/deliveries/new	GET
	/admin/deliveries/:id/edit	GET
	/admin/deliveries/:id/restore	PUT
	/admin/deliveries/:id/recalculate	PUT
Manejo de repartidores	/admin/couriers/batch_action	POST
	/admin/couriers	GET, POST

	/admin/couriers/:id	GET, PUT, DELETE
	/admin/couriers/new	GET
	/admin/couriers/:id/edit	GET
	/admin/couriers/:id/restore	PUT
Manejo de administradores	/admin/admin_users/batch_action	POST
	/admin/admin_users	GET, POST
	/admin/admin_users/:id	GET, PUT, DELETE
	/admin/admin_users/new	GET
	/admin/admin_users/:id/edit	GET

Tabla 26 - Especificación de endpoints para el consumo de la aplicación web

ANEXO 14 – “Análisis de tecnología front-end para aplicación móvil”

Como requerimiento no funcional el cliente especificó desde el principio que se trabajaría con Java nativo de Android para el desarrollo front-end.

Debido a que la empresa solo otorga celulares Android a sus cadetes, es esencial priorizar el soporte para esta plataforma, más específicamente al modelo de dispositivo que utilizan sus repartidores.

Para explicar mejor por qué el equipo estuvo de acuerdo con la decisión del cliente de desarrollar en Java nativo primero cabe destacar las posibles alternativas.

- Aplicaciones nativas
  - Funcionan únicamente en un sistema operativo.
  - Acceso total a todas las funcionalidades del hardware y software del dispositivo.
  - Mejor experiencia de usuario que facilita el entendimiento de la aplicación.
  - Mejor performance que el resto de las tecnologías.

- Aplicaciones híbridas
  - Funcionan en todos los sistemas operativos.
  - Desarrollo único para el que existen muchos lenguajes.
  - Acceso a la mayoría de las funcionalidades de hardware y software del dispositivo.
  - Experiencia de usuario e interfaz de usuario mejor que las aplicaciones web pero peor que las nativas.

De decidir desarrollar el sistema en Android nativo se podría tener un sistema más performante y con mejor usabilidad dado que se dispone de todos los recursos y funcionalidades del dispositivo y el sistema operativo. Por otro lado, las ventajas de desarrollar en híbrido, principalmente la portabilidad, no resultan apropiados en el contexto del proyecto donde la empresa está segura de que no va a utilizar otro sistema operativo que ese.

Por esas razones el equipo apoyó la decisión del cliente de desarrollar la aplicación en Java nativo de Android.

ANEXO 15 – “Algoritmo fuerza bruta para problema de enrutamiento”

El equipo implementó rápidamente un algoritmo de orden O(N!) para el algoritmo de enrutamiento que considera todos los caminos posibles e intentó correrlo en un *dyno One-off* de Heroku para tener un estimativo del tiempo.

El algoritmo se veía algo así:

router_service_brute_force.rb

```ruby
class RouterService
  def initialize(delivery)
    return false unless delivery.pending_routing?
    @recipient_addresses = delivery.recipient_addresses
  end

  def sort_orders
    remaining_points_ids = @recipient_addresses.pluck(:id).to_a
    route = []
    route << remaining_points_ids.slice!(0)
    until remaining_points_ids.empty? do
      next_point_id = cheapest_next_point(route.last, remaining_points_ids).id
      index_of_next_point = remaining_points_ids.index(next_point_id)
      route << remaining_points_ids.slice!(index_of_next_point)
    end

    print_route(route)
    route
  end

  def cheapest_next_point(from_id, possible_recipient_addresses_ids)
    possible_recipient_addresses = RecipientAddress.where(id: possible_recipient_addresses_ids)
    min_cost = possible_recipient_addresses.joins(:cost_matrix_cells_as_destination)
                  .where("cost_matrix_cells.recipient_address1_id = #{from_id}")
                  .minimum("cost_matrix_cells.cost")
    possible_recipient_addresses.joins(:cost_matrix_cells_as_destination)
                  .find_by("cost_matrix_cells.cost = #{min_cost}")
  end

  def print_route(route)
    p '----------------'
    route.each do |point_id|
      p RecipientAddress.find(point_id).corrected_address
    end
    p '----------------'
  end
end
```

Ilustración 101 - Algoritmo de enrutamiento a fuerza bruta

Desafortunadamente el equipo descartó este algoritmo enseguida ya que, para comenzar, su ejecución tomó más que 3 horas y se aprendió que Heroku no permite correr tareas por períodos prolongados de tiempo. De utilizar esta solución, el equipo debería desplegar su propia infraestructura y generar la comunicación necesaria con la API que guarda los datos.

Además, para los casos donde los repartos se planifican en la misma mañana, ya más de dos horas de espera por una ruta es demasiado tiempo.

Por otro lado, en el algoritmo anterior los costos ya estaban pre calculados en un modelo llamado *Cost Matrix Cell*, que había sido simulado para retornar información aleatoria para la prueba. En un escenario real, esta información debería conseguirse de servicios que consideraran factores como calles, avenidas, dirección de las mismas, reglas de tráfico, etc. Esto requeriría tener almacenadas 16,900 celdas para un reparto de 130 pedidos, que implicaría una base de datos de un plan pago dada la cantidad de filas, gastos importantes en los servicios como Google Traffic API [27], además de la complejidad de asegurar que el código mantenga esa tabla completa y actualizada todo el tiempo.

Por todas estas razones el equipo comenzó a buscar alternativas de servicios en la nube que resuelvan este tipo de problemas.

ANEXO 16 – “Análisis de servicios de enrutamiento”

Las herramientas que se analizaron son todas las que el equipo encontró en búsquedas en Internet, buscando en Google, en foros, en Github, etc.

Servicio	Link	Se adapta al problema	Precio	Documentación y soporte	Notas
Google Maps Direction API	www.developers.google.com/maps/documentation/directions/intro	No. Hasta 25 waypoints	$10.00 cada mil consultas	Buena documentación y ayuda online	
Google maps tsp solver	www.developers.google.com/optimization/routing/tsp	No. Hasta 25 waypoints	$15.00 cada mil consultas	Buena documentación y ayuda online	
BADGERMAPPING	www.badgermapping.com	Sí	$66/usuario	Buena documentación y ayuda online	
roadwarriorllc	www.roadwarriorllc.com	Sí	$5/usuario + $10 mensuales base	Buena documentación y ayuda online	
simpliroute	www.simpliroute.com	Sí	$40/usuario cada mes	Buena documentación y ayuda online	
tarotrouting	www.tarotrouting.com	Sí	$60/usuario cada mes	Buena documentación y ayuda online	
Optimap	Shut down	Parecía que sí	Está fuera de servicio		
YOURS	www.wiki.openstreetmap.org/wiki/YOURS	Sí	Gratis	Documentación escasa e interfaz de prueba anticuada	Se probó enrutar un reparto y el camino no era más rápido

					que el base que utilizan los repartidores
Mapquest	www.business.mapquest.com/products/routing-directions-api	No. Hasta 25 waypoints	Gratis	Documentación adecuada	
MapBox Direction Map API	www.docs.mapbox.com/help/tutorials/getting-started-directions-api	No. Hasta 25 waypoints	Gratis	Buena documentación y ayuda online	
OSRM	www.project-osrm.org	No. No ordena los puntos	Gratis	Buena documentación	
Graphhopper	www.graphhopper.com	Sí. Ordena hasta máximo de 150 puntos	304€ por mes	Buena documentación y ayuda online	
Route4me	www.route4me.com	Sí. Ordena hasta máximo de 200 puntos	Comienza en 999 USD	Buena documentación y ayuda online	
Routific	www.api.routific.com	Sí. Sin límite de puntos	150 USD por mes	Buena documentación	
Locus	www.locusmap.eu	No. Enfocado en el área de EU	9,99 EUR por mes	Buena documentación	
LKH	http://akira.ruc.dk/~keld/research/LKH	No. Enfocado solo para viajar entre ciudades	Gratis	Documentación básica	
Bizom	www.bizom.in	Sí	60 USD por mes por usuario	Buena documentación y ayuda online	
Triposo	www.triposo.com	No. Enfocado en tours	500 USD por mes	Buena documentación y ayuda online	

Routeme	routeme.io	Sí	A negociar. Menos de USD 30 por mes. Posibilidad de negociar reducir el costo por propaganda o alojamiento de necesidad de cómputo en Amazon	La API es sencilla de utilizar y se encuentra bien documentada en la web.	El equipo descubrió esta opción luego de haber construido su propio algoritmo. Soporte personalizado. Contacto directo con propietarios y desarrolladores. La ruta fue probada y es precisa. Otras funcionalidades incluyen visualizar recorridos creados desde la web, con link para abrirlos en Google Maps, exportarlos, y tiempo estimado de viaje.

Tabla 27 - Comparación de servicios de enrutamiento

ANEXO 17 – “Plan de calidad”

Fase	Actividad	Entrada	Salida	Herramientas y tecnología	Involucrados
Análisis del negocio	Análisis del contexto y del problema	Problema presentado por los clientes	Conocimiento del problema	Reuniones con interesados	Equipo
	Indagación de herramientas tecnológicas	Sistema a crear	Documentos sobre las herramientas elegidas para crear el sistema	Reuniones con clientes e investigación	Equipo y clientes
Ingeniería de requerimientos	Identificación de requerimientos	Solución	Especificación de requerimientos	Story Mapping, prototipación	Equipo y clientes
	Priorización	Requerimientos	Requerimientos priorizados	Story Mapping, reuniones con clientes, prototipación	Equipo e interesados
	Estimación	Requerimientos priorizados	Requerimientos estimados	Punto función, adaptación simplificada de Planning Poker	Equipo
	Definición de alcance inicial	Requerimientos estimados y priorizados	Product Backlog	Release Map con fecha final y margen	Equipo
Arquitectura	Análisis de requerimientos no funcionales	Solución	Especificación de requerimientos no funcionales e identificación de los atributos de calidad	Reuniones con clientes	Equipo
	Diseño de la arquitectura	Rands	Diagramas iniciales de la arquitectura	Aplicación de patrones y tácticas para los problemas a enfrentar, UML	Equipo
		RNFs, diagramas inciales	Selección de herramientas	Análisis comparativos	Equipo

	Validación de la arquitectura inicial	Diagramas iniciales	Arquitectura verificada y validada	Revisión con expertos	Equipo
	Definición de estándares	Listado de tecnologías a utilizar	Documento de estándares a utilizar	Guías de los lenguajes	Equipo
Desarrollo	Configuraciones previas	Listado de tecnologías a utilizar, necesidades de la empresa	Definición de ambientes de desarrollo	Ambientes de desarrollo para Ruby y Android	Equipo
	Desarrollo	Product Backlog	Funcionalidades implementadas	*Backend* en Ruby, Frontend Android nativo	Equipo
	Revisión de código	Código fuente, definición de estándares a utilizar	Código que cumple con los estándares	Herramientas de análisis de código, revisiones entre los integrantes	Equipo
	Pruebas unitarias	Casos de prueba, criterios de aceptación y sistema construido	Lista de resultados y porcentaje de cobertura	Frameworks de testing, integración continua	Equipo
Pruebas	Pruebas de usabilidad	Prototipos y sistema construido	feedback, sugerencias de cambios	Poner producto en manos de usuarios, observación, encuestas, entrevistas, herramientas de registro de las acciones del usuario	Equipo y usuarios (repartidores y administradores)
	Pruebas de validación	Prototipos, sistema construido	feedback, sugerencias de cambios	Demostración guiada, pruebas en repartos reales	Equipo

Actividad independiente de fase	Plan de proyecto	Características del proyecto y equipo	Plan de proyecto, selección de herramientas de gestión		Equipo
	Gestión de riesgos	Conocimientos de riesgos del proyecto	Plan SQA, Versión de análisis de riesgos		Equipo
	Aseguramiento de la calidad		Plan SQA		Equipo
	Gestión de la configuración	Listado de tecnologías a utilizar	Plan SCM		Equipo
	Pruebas	Características del proyecto y producto a construir	Plan de pruebas		Equipo

Tabla 28 - Plan de calidad

ANEXO 18 – “Estándares de codificación”

A continuación se listan y referencian estándares de programación de los lenguajes de programación utilizados en el proyecto.

Lenguaje	Estándar	Referencia
API	REST	https://restfulapi.net/resource-naming/
	Guías generales para diseño de APIs por Microsoft	https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design
	Prácticas para HTTP y JSON API extraídas originalmente del trabajo de Heroku Platform API	https://geemus.gitbooks.io/http-api-design/content/en/
Ruby	Guía oficial de Ruby	https://github.com/styleguide/ruby
	Guías de la organización Rootstrap (donde trabaja Leticia)	https://github.com/rootstrap/tech-guides/tree/master/ruby
Ruby on Rails	Guías de la organización Rootstrap (donde trabaja Leticia)	https://github.com/rootstrap/tech-guides/blob/master/ruby/rails.md
Java	Guías oficial de Google	https://google.github.io/styleguide/javaguide.html
HTML	Guías W3S internacionalmente conocidas	http://www.w3schools.com/html/html5_syntax.asp
CSS	Guías W3S internacionalmente conocidas	http://www.w3schools.com/css/default.asp
	Guías de la organización Rootstrap (donde trabaja Leticia)	https://github.com/rootstrap/tech-guides/blob/master/css.md

Tabla 29 - Estándares de los lenguajes

ANEXO 19 – “Aplicación de Estándares”

Para automatizar lo máximo posible el cumplimiento con las buenas prácticas de cada lenguaje de programación, se hizo uso de herramientas que analizan la calidad de código.

La siguiente tabla muestra las herramientas de análisis de calidad de código utilizadas.

Aplicación	Herramienta	Referencia
Back-end	Rubocop	https://github.com/rubocop-hq/rubocop
	Reek	https://github.com/troessner/reek
	Brakeman	https://github.com/presidentbeef/brakeman
	Rails Best Practices	https://github.com/flyerhzm/rails_best_practices
Front-end	Google Java Style Guide	https://google.github.io/styleguide/javaguide.html

Tabla 30 - Herramientas de estándares

Las herramientas están configuradas por defecto para respetar las prácticas recomendadas más conocidas de cada lenguaje, identificando también síntomas de un mal diseño y distribución de responsabilidades. Sin embargo, la mayoría de ellas permiten configurarse personalizadamente e incluso agregar chequeos propios.

En la siguiente imagen se visualiza un ejemplo del chequeo en Java.

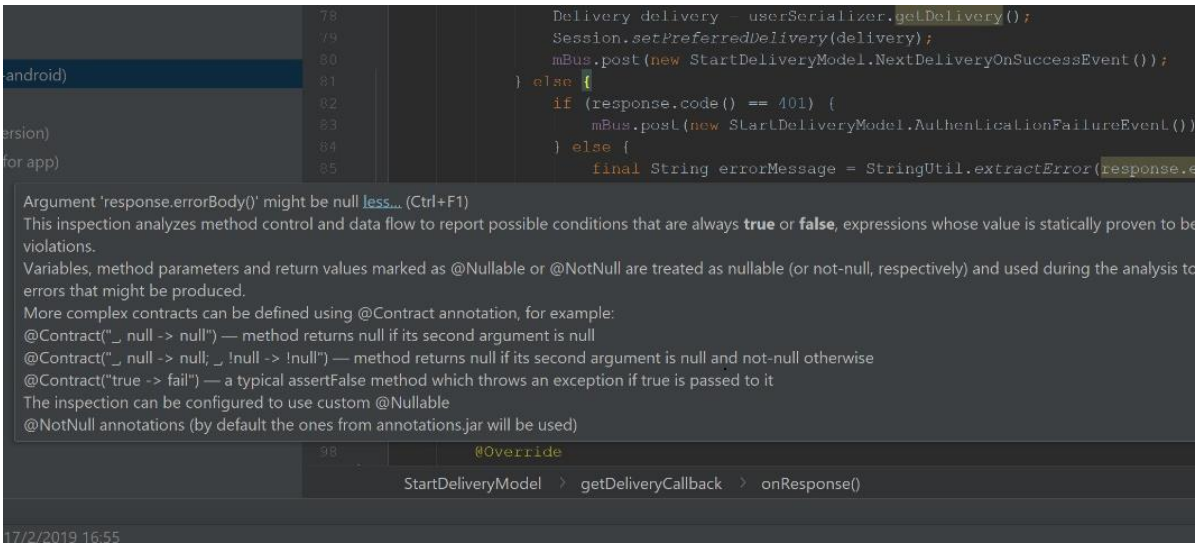


Ilustración 102 - Ejemplo de lint Java

A continuación se muestra parte de la configuración de Reek, una de las herramientas utilizadas para analizar la calidad de código en *Ruby*. Reek se focaliza principalmente en reducir la complejidad del código, como se puede visualizar en los nombres de las reglas.

Ilustración 103 - Ejemplo de configuración de herramienta de análisis de código en Ruby

A continuación se muestra cómo se visualizan los “*code smells*" en al correr Rubocop, otra de las herramientas utilizadas en Ruby, en un método de prueba de muy baja calidad.

```
bundle exec rubocop app config lib spec
Inspecting 147 files
.....................W.....................................................................................
.....................................

Offenses:

app/models/delivery.rb:130:5: W: Useless assignment to variable - my_unused_var_to_test_code_analysis.
    my_unused_var_to_test_code_analysis = 3
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
app/models/delivery.rb:131:101: C: Line is too long. [145/100]
    # line too long line too long line too long line too long line too long line too long line too long lin
e too long line too long line too long
                                                                                                    ^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
app/models/delivery.rb:132:5: W: Literal 3 used in void context.
    3
    ^
app/models/delivery.rb:134:7: C: Incorrect indentation detected (column 6 instead of 4).
      # unindented_code
      ^^^^^^^^^^^^^^^^^
app/models/delivery.rb:135:5: C: Favor modifier if usage when having a single-line body. Another good alter
native is the usage of control flow &&/||.
    if true
    ^^
app/models/delivery.rb:135:8: W: Literal true appeared in a condition.
    if true
       ^^^^
app/models/delivery.rb:139:5: W: Unreachable code detected.
    false
    ^^^^^
app/models/delivery.rb:140:5: W: end at 140, 4 is not aligned with def at 129, 2.
    end
    ^^^
app/models/delivery.rb:142:1: C: Extra blank line detected.
app/models/delivery.rb:143:1: C: Extra blank line detected.

147 files inspected, 10 offenses detected
rake aborted!
```

Ilustración 104 - Ejemplo de ejecución de herramienta de análisis de código en Ruby

En Ruby estas herramientas se corren con el comando:

rake code_analysis

que es una tarea del proyecto programada para correr las cuatro herramientas mencionadas anteriormente.

```
1  task :code_analysis do
2    sh 'bundle exec brakeman . -z -q'
3    sh 'bundle exec rubocop app config lib spec'
4    sh 'bundle exec reek app config lib'
5    sh 'bundle exec rails_best_practices .'
6  end
```

Ilustración 105 - Tarea de Ruby para correr las herramientas de análisis de código

Cuando todos los chequeos de todas las herramientas en Ruby aprueban, se visualizan de la siguiente manera:

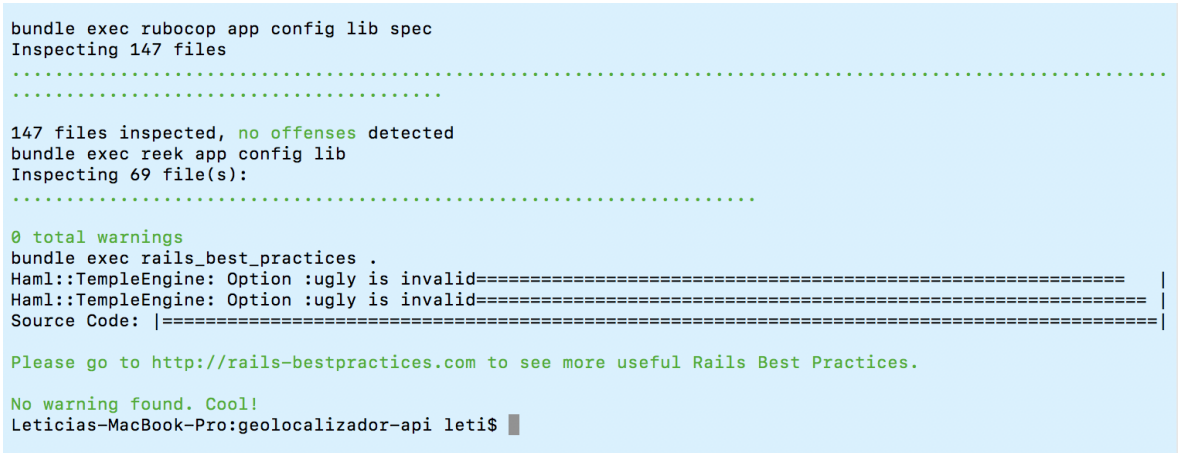


Ilustración 106 - Herramientas de análisis de calidad de código

ANEXO 20 – “Plan de gestión de incidentes”

El equipo tuvo en cuenta que, como todo proyecto, surgirían bugs y que se debía definir en qué momento se corregirían. El equipo definió un plan de gestión de incidentes que tuvo como intención explicar cómo proceder ante la aparición de un defecto.

Se definió que un *bug* podría ser identificado por cualquiera persona que utilice la aplicación, ya sea un integrante del equipo, uno de los clientes o un repartidor. Los bugs se reportaron al equipo quien se encargó de reproducirlos e introducirlos al proceso para que puedan ser corregidos cuando llegue el momento.

8.7. Categorización de defectos

El equipo entiende que no todos los bugs son igual de importantes por lo que se comenzó definiendo la categorización de bugs según su prioridad dada por la severidad:

1. Alta: Aquellos que generan que el sistema no esté disponible o que casos de prueba básicos estén fallando.

2. Media: Aquellos que si bien permiten que el sistema esté disponible y sin casos de prueba importantes que fallen, ocasionan que alternativos fallen.

3. Baja: Aquellos que si bien permiten que el sistema esté disponible y que los casos de prueba no fallen pero hay detalles que arreglar.

8.8. Registro de defectos

Se utilizó *Trello* para el seguimiento de *bugs*, donde se creó una columna *Bugs* para colocar los nuevos defectos reportados. Idealmente el cliente también agregaría bugs a esta columna pero en la práctica el cliente no utilizó el Trello por su cuenta sino que reportó directamente al equipo quien luego lo registró.

Al encontrarse frente a un nuevo reporte de un *bug*, el equipo primero corroboró que efectivamente se trate de un defecto y no de un uso incorrecto (el cual podría ser motivo de replanteo del flujo). Luego se creó la *card* en *Trello* en la columna *Bugs* con un título, descripción, capturas y/o pasos para reproducirlo y una prioridad según la categorización de defectos definida. Si el *bug* no se pudo reproducir en la misma reunión se agregó un comentario acerca de esto para que el equipo intente reproducirlo más tarde. A los reportados bugs se les iba asignando una estimación a medida que el equipo tenía tiempo de estimarlos.

8.9. Tolerancia a defectos

Con la intención de priorizar y no acumular *bugs* indefinidamente, el equipo decidió definir una cantidad máxima de defectos de cada categoría permitida por ambiente. Los integrantes no deberían permitirse superar esta cantidad debiendo introducir un *Sprint* extra para la corrección de los mismos si fuese necesario con el objetivo de no seguir acumulando defectos.

Ambiente de detección	**Prioridad**	**Máximo de defectos permitidos**
Desarrollo y Producción (ambientes disponibles a los clientes)	1	0
	2	2
	3	5
Local (ambiente local de desarrollo)	Cualquiera	Sin límite

Tabla 31 - Tolerancia a defectos

La primera definición de este plan separaba los ambientes producción y desarrollo, siendo menos tolerable con los defectos en producción. La realidad es que la separación de cantidad de defectos permitida entre desarrollo y producción no fue del todo útil ya que el ambiente de producción se utilizó puntualmente para las pruebas en repartos reales, y recién a partir de la segunda prueba se utilizó regularmente, pero para entonces solo quedaban unas semanas de desarrollo. Es por eso que se definió unificar la cantidad de defectos permitidos y facilitar la aplicación de este plan.

8.10. Arreglo de defectos

Se planificó que los defectos reportados se fueran solucionando apenas se pudiese, incluyendo las *cards* de *bugs* dentro de las funcionalidades planeadas en cada *Sprint*, intentando en la medida que sea posible no retrasar el cronograma de *Sprints*.

También se trabajó en arreglos de bugs en los *Sprint*s agregados bajo necesidad durante el proyecto para cumplir con la tolerancia a defectos explicada anteriormente.

Cuando un integrante comenzó a trabajar en la corrección de un *bug*, tomó la *card* correspondiente de la columna *Bugs* y la movió a *Sprint Backlog,* como se hizo con cualquier otra tarea. Su flujo a partir de aquí fue el mismo que con las *cards* de historias

de usuarios en el sentido que se fueron moviendo por las columnas a medida que su estado cambiaba.

A continuación se puede visualizar una captura de pantalla del tablero de Trello, con la columna *Bugs* y varias cards con etiqueta roja de *bug* en diferentes estados.

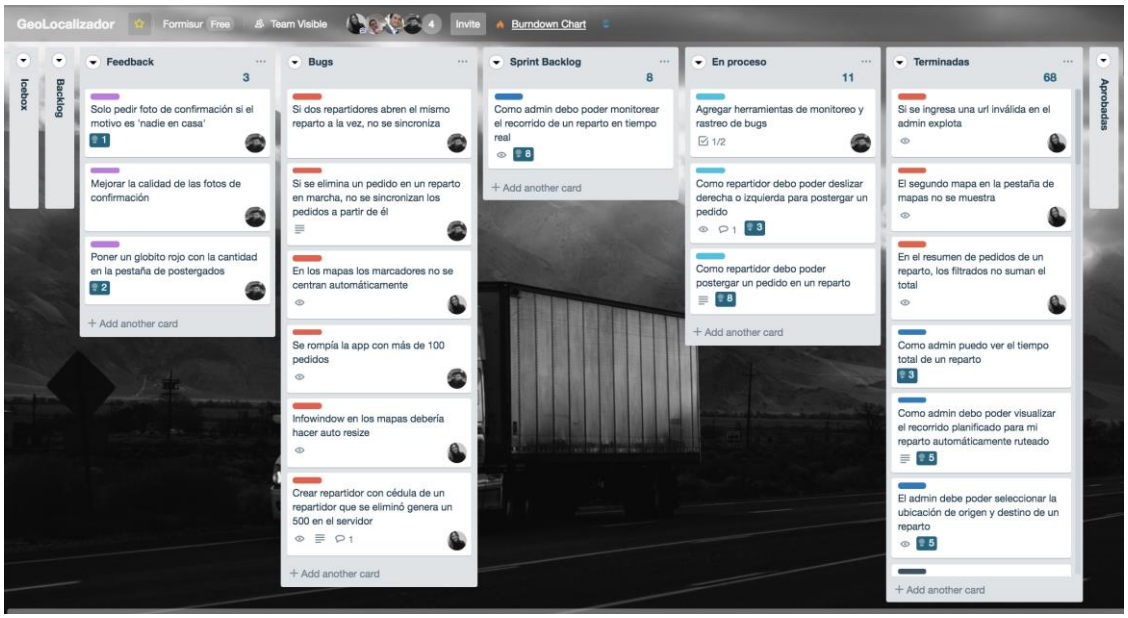


Ilustración 107 - Ejemplo de columna de bugs de Trello

A continuación se muestra un ejemplo de una card de bug identificado, y resuelto. Se puede apreciar en su actividad cómo se comenzó a trabajar en él días después de que fue reportado, ya que fue un defecto de prioridad baja.

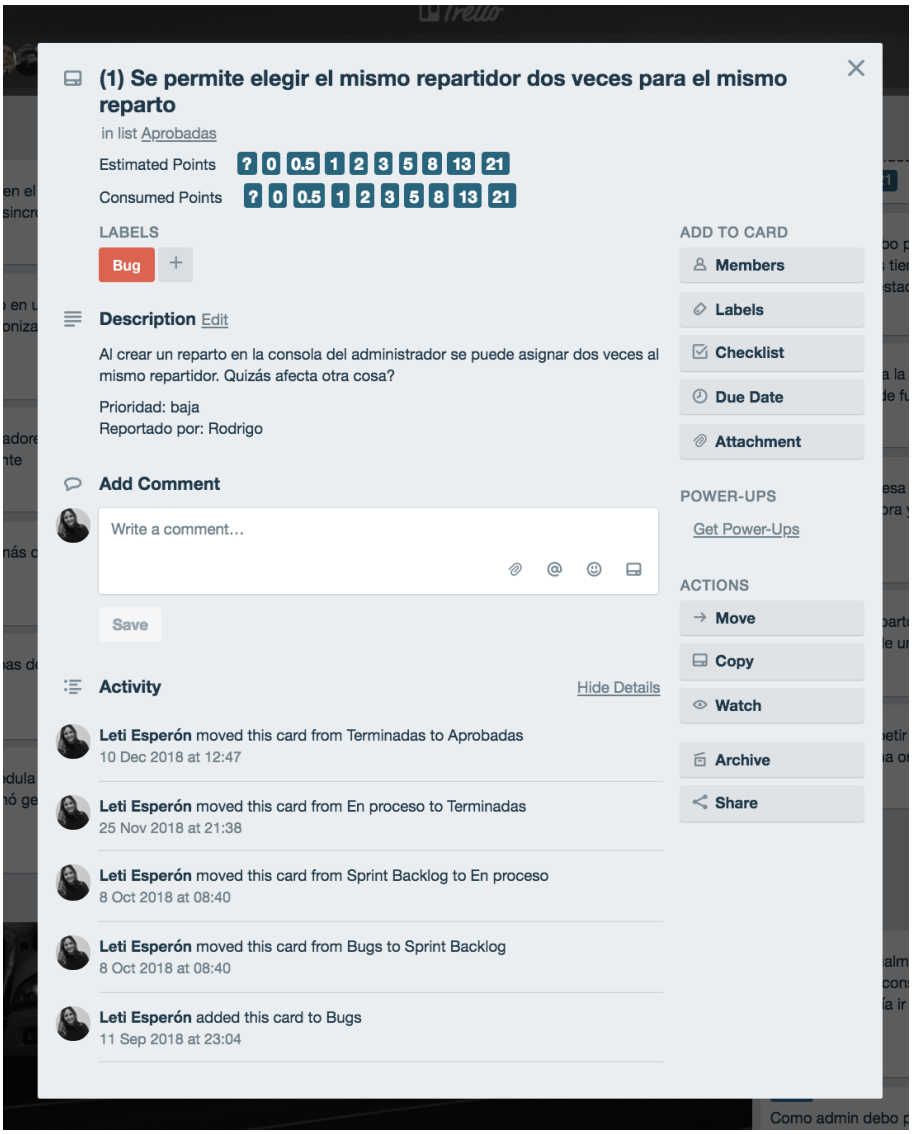


Ilustración 108 - Ejemplo de bug en Trello

Además de Trello para realizar el seguimiento, se utilizaron herramientas para identificar bugs. Estas fueron Fabric [36] del lado de la aplicación móvil y Rollbar [37] del lado del servidor y de la aplicación web. Para visualizar más detalles de cómo fueron utilizadas las herramientas para la gestión de incidentes referirse a la sección 5.3.3 - “Monitoreabilidad”.

ANEXO 21 – “Encuestas de satisfacción”

Se realizaron dos instancias de encuestas, una al finalizar cada *Release*. Se separó la encuesta para el administrador y repartidores. Al finalizar el proyecto también se envió una encuesta a los clientes que evalúa otros aspectos del trabajo del equipo.

Como las preguntas eran pocas y fueron pocas las personas encuestadas dada que las pruebas significaban una gran inversión de tiempo para los clientes, el equipo consideró más fácil para los repartidores no utilizar ninguna herramienta para encuestas y realizarlas oralmente o a través de los clientes. Esto permitió además recolectar todas las sugerencias y conclusiones reportadas en el ANEXO 7 – “Resultado de pruebas de los prototipos con usuarios”, que permitieron mejorar el nivel de satisfacción en la siguiente encuesta.

Las encuestas fueron realizadas utilizando la escala de Likert del 1 al 5, midiendo el nivel de conformidad con una afirmación positiva siendo 5 muy de acuerdo y 1 muy en desacuerdo.

A continuación se resumen las preguntas y resultados de las encuestas.

Pregunta	Resultados E1	Resultados E2
Encuesta a los repartidores		
La nueva aplicación me ahorró tiempo comparado con el viejo proceso.	5 - 4 - 4	5 - 5 - 5
La nueva aplicación me eliminó la necesidad de manejar boletas en papel durante el reparto.	3 - 3 - 2	5 - 5 - 5
Encontré la nueva ruta más rápida que la que hacíamos antes.	N/A	4 - 3 - 5
En general, la nueva aplicación me pareció rápida.	5 - 4 - 4	5 - 4 - 4
En general, la nueva aplicación me pareció fácil de usar.	5 - 5 - 5	5 - 5 - 5
Me pareció sencillo marcar un pedido como entregado.	5 - 5 - 5	5 - 5 - 5
Me pareció sencillo marcar un pedido como rebotado.	5 - 4 - 4	5 - 5 - 4
Encuesta al administrador		
La nueva aplicación me ahorró tiempo comparado con el viejo proceso.	5	5
La nueva aplicación me eliminó la necesidad de manejar boletas en papel.	3	5
En general, la nueva aplicación me pareció fácil de usar.	5	5

El proceso de crear un reparto me pareció sencillo.	5	5
Me resultó fácil conocer el estado de un reparto.	4	5
Me resultó fácil conocer el estado de un pedido en particular para responder consultas de destinatarios.	4	5
Las alertas me resultaron útiles.	N/A	5
Las alertas me resultaron fáciles de visualizar.	N/A	5
Encuesta general a los clientes		
Identificación del problema y solución		
El equipo un buen trabajo identificando los problemas iniciales a resolver.		5 - 5
El equipo un buen trabajo diseñando la solución a los problemas a resolver.		5 - 5
Los prototipos propuestos por el equipo (mock ups) eran de mi agrado y resolvían el problema.		5 - 5
Alcance		
El alcance final del producto final cumplió o superó mis expectativas iniciales.		5 - 4
El equipo tuvo en cuenta mis opiniones y prioridades para definir el alcance del backlog.		5 - 5
Gestión		
El equipo cumplió con las ceremonias de gestión acordadas en un principio (tener *Sprint* Planning y *Sprint* Review cada dos semanas).		5 - 5
Las reuniones estaban preparadas por parte del equipo.		4 - 5
El equipo cumplía con lo prometido *Sprint* a *Sprint*. (Terminaban las funcionalidades planeadas para el final del *Sprint*, corregían el feedback brindado, etc.)		5 - 5
Flexibilidad y Proactividad		
El equipo fue flexible ante peticiones de cambios en las funcionalidades ya terminadas, en las priorizaciones, o en funcionalidades planeadas.		5 - 5
El equipo estaba disponible para consultas/reuniones improvisadas.		5 - 5

El equipo era proactivo para sugerir soluciones o funcionalidades que podrían dar valor al producto	4 - 4
Consideré las ideas del equipo como valiosas, aportaban al producto.	4 - 4
El equipo fue demandante en la medida justa y necesaria en cuanto a consultas, reuniones, cuentas en servicios, encuestas, pruebas de validación, etc..	5 - 5
Defectos	
El equipo hizo un buen trabajo en cuanto a identificación, registro, y arreglo de defectos.	5 - 5
Técnico	
Quedé conforme con la solución final para el problema de enrutamiento de pedidos. (Integración con Routeme).	5 - 5
El equipo cumplió con la mantenibilidad de código que esperaba. (Aplica para Martín)	5 - 5
Diseño	
Quedé conforme con el diseño gráfico de la aplicación móvil que brindó la diseñadora.	5 - 5
El equipo respetó los diseños gráficos de la aplicación móvil que brindó la diseñadora.	5 - 5
General	
En general quedé conforme con el producto final realizado por el equipo.	5 - 5
En general quedé conforme con el trabajo del equipo.	5 - 5
Personalmente sentí que estuve comprometido con el proyecto.	5 - 5
Impacto	
El proyecto era de gran importancia o impacto para la empresa al comenzar el proyecto.	5 - 5
El proyecto es de gran importancia o impacto para la empresa actualmente.	5 - 5
Es altamente probable que utilice el producto realizado por el equipo.	5 - 5

Tabla 32 - Resultados de encuestas

ANEXO 22 – “Reuniones de coordinación”

A continuación se adjuntan reuniones de coordinación de ejemplos. Se realizaron cinco en total durante el proyecto. En cada una de ellas se realizaron las siguientes actividades:

1. Recapitulación de funcionalidades realizadas y pendientes del *Milestone*
2. Recapitulación de *bugs*
3. Retrospectiva
4. Backlog Grooming
5. Pruebas
6. Nueva versión del análisis de riesgos

Reunión 1

Fecha: Fin Milestone 1 - 25/06/18

Funcionalidades terminadas

Al día el equipo lleva terminadas todas las funcionalidades previstas para el fin del Milestone:

Setup de Trello e historias de usuario
Roadmap
Mock ups
Primeros diseños
Capacitación de Ruby y Android
Selección de herramientas
Setup Heroku y Google Playstore
Ícono de la app y landing
Como admin debo poder loguearme a la web con usuario y contraseña
Como admin debo poder crear usuarios de la app para los repartidores
Como admin debo poder ver repartidores
Como admin debo poder eliminar y recuperar repartidores
Como repartidor debo poder loguearme
Como repartido debo poder cerrar sesión
Como repartidor debo poder cambiar mi contraseña

Bugs sin corregir

No se reportaron bugs.

Retrospectiva

Start	Stop	Continue
Traquear las horas utilizando etiquetas.	Hacer actas de reuniones con el tutor. Es muy tedioso. Mandar mail informal o anotarlo en el wunderlist compartido.	La división de las tareas y los roles está dando resultado.
Marcar al final de cada *Sprint* las horas que hizo cada uno así es menos tedioso al final. Llevar más al día el documento de métricas por *Sprint*.	Dejar documentos colgados en la carpeta 'En Proceso'. Corregirlos de una vez así quedan prontos.	Se está llegando a las funcionalidades del *Sprint* exitosamente. Está bien decidido el largo de los *Sprint*s. Están bien estimadas las historias de usuario hasta ahora.
Hacer pre-demos para no improvisar en las reuniones.		Las Reviews y Plannings son del tiempo correcto. Sin más ni menos.
Mejorar la forma de escribir los criterios de aceptación.		Está siendo útil el traqueo de las horas. Permitió calcular la velocidad del equipo.
Hacer las demos más despacio para que el cliente tenga tiempo de dar su opinión y encontrar posibilidades de mejora.		Se está utilizando correctamente GitFlow (sin ser por la integración a master que se habló en 'Stop'), y se está respetando el plan de PR e integración continua. Como resultado los códigos están prolijos y robustos.
Turnar el armado de actas entre los integrantes. Es una tarea muy tediosa como para que siempre las haga la misma persona y sirve que ambos estén en sincronía de lo que se habló en la reunión.		El Release Plan el Trello se están llevando al día.
		El Wunderlist está siendo útil para sincronizar tareas y organizar dudas entre los integrantes.

Se reescriben y re priorizan las funcionalidades del Milestone 2:

El plan original contiene:

Milestone 2: Administración de pedidos y repartos
Como admin debo poder administrar destinatarios
Como admin debo poder ver clientes
El sistema debe obtener los datos de clientes de la base de datos de otra aplicación
Como admin debo poder crear un reparto
Como admin debo poder ver los datos de un reparto
Como admin debo poder ver la lista de repartos
Como admin debo poder cancelar un reparto

Se sustituyen por:

Milestone 2: Administración de pedidos y repartos
Como admin debo poder crear un reparto
Como admin debo poder ver la lista de repartos
Como admin debo poder ver los datos de un reparto
Como admin debo poder ver los datos de un pedido
Como admin debo poder ver la lista de pedidos
Como admin debo poder filtrar repartos
Como admin debo poder filtrar pedidos
Como admin debo poder administrar direcciones de destinatarios
Como admin debo poder filtrar las correcciones de direcciones
Como admin debo poder cancelar un reparto
El reparto debe crearse con las direcciones de destinatarios arregladas

Se elimina la funcionalidad “El sistema debe obtener los datos de clientes de la base de datos de otra aplicación” ya que se considera que los datos del cliente no son de importancia al sistema actual, al menos no es una prioridad.

Se agregan funcionalidades de filtro que no son difíciles de hacer pero facilitarán la prueba de las funcionalidades en el futuro.

Pruebas

Qué se quiere probar	Entradas	Resultado Esperado	Funciona
Android			
Ingreso de la aplicación			

Se abre aplicación sin estar logueado.	-	Se muestra pantalla de landing	OK
Se abre aplicación logueado.	-	Se muestra la pantalla principal	OK
Como repartidor debo poder loguearme			
Ingreso con C.I. no válida	Se ingresa con C.I. “123AA” y contraseña “123456”	Se muestra mensaje de error “Introduce su cédula de 8 dígitos” y no permite el ingreso	OK
Ingreso con C.I. válida y contraseña no válida	Se ingresa con C.I. “45709320” y contraseña “123456”	Se muestra mensaje de error “Cédula o Contraseña no válida” y no permite el ingreso	OK
Ingreso con C.I. no registrada y contraseña no válida	Se ingresa con C.I. “12345678” y contraseña “123456”	Se muestra mensaje de error “Cédula o Contraseña no válida” y no permite el ingreso	OK
Ingreso con C.I. válida y contraseña válida	Se ingresa con C.I. “45709320” y contraseña “asdasdasd”	Permite el ingreso a la pantalla principal	OK
Como repartido debo poder cerrar sesión			
Click en botón de cerrar sesión	-	Vuelve a pantalla de landing y elimina datos de sesión	OK
Como repartidor debo poder cambiar mi contraseña			
Ingreso para cambio de contraseña con antigua contraseña no válida	Se ingresa con antigua “123123123”	Muestra mensaje de error “Contraseña incorrecta”	OK
Ingresa con contraseñas nuevas con menos de 6 caracteres	Se ingresa nueva contraseña “12345”	Muestra mensaje de error “Debe tener 6 caracteres o más”	OK
Ingresa con contraseñas nuevas pero que no coinciden	Se ingresa nueva contraseña “123456” y “1234567”	Muestra mensaje de error “Las contraseñas no coinciden”	OK
Ingresa con contraseñas nuevas validas	Se ingresa nueva contraseña “123456” y “123456”	Muestra mensaje de aceptación y cambia contraseña	OK
Admin app			

Como admin debo poder loguearme a la web con usuario y contraseña			
Ingreso con email y contraseña invalidas	Se ingresa usuario “admin” y contraseña “123456”	Muestra mensaje de error “Correo o contraseña inválidos”	OK
Ingreso con email valido y contraseña invalido	Se ingresa usuario “admin” y contraseña “123456”	Muestra mensaje de error “Correo o contraseña inválidos”	OK
Ingreso con email y contraseña validos	Se ingresa usuario “admin@example.com” y contraseña “password”	Ingresa a la página principal	OK
Como admin debo poder crear usuarios de la app para los repartidores			
Ingreso repartidor con cédula incorrecta	Se ingresa repartidor con cédula “asdasd”	Muestra mensaje de error “no puede estar en blanco y deben ser 8 dígitos, sin puntos ni guiones”	OK
Ingreso repartidor con cédula repetida	Se ingresa repartidor con cédula “45709320”	Muestra mensaje de error “ya está en uso”	OK
Ingreso repartidor con cédula válida pero nombre vacío	Se ingresa repartidor con cédula “45709321” y nombre “” y apellido “”	Muestra mensaje de error “no puede estar en blanco”	OK
Ingreso repartidor con dato válidos	Se ingresa repartidor con cédula “45709321” y nombre “Martín” y apellido “García”	Registra al repartidor y pasa a la pantalla de detalles del mismo	OK
Como admin debo poder ver repartidores			
Al ingresar a la sección de repartidores se visualizará una lista de repartidores	-	Se muestra listado de todos los repartidores del sistema	OK
Como admin debo poder eliminar y recuperar repartidores			
Cuando hago click en Borrar el usuario pasará su estado de Activado a “NO”	-	Se pasa a la lista de eliminados	OK
Cuando hago click en restaurar de un repartidor eliminado	-	Se pasa a la lista de actuales	

Análisis de riesgos

Identificador	Riesgo	Fecha de identificación
R1	Escasez de tiempo para dedicarle la atención necesaria al proyecto	15/5/2018
R2	Mala estimación de la velocidad del equipo	15/5/2018
R3	Demorar más de lo estimado por falta de dominio en desarrollo Android	15/5/2018
R4	Mala estimación de una o varias historias	15/5/2018
R8	Cambios grandes en los requerimientos ya implementados	15/5/2018
R6	Inhabilitación de un integrante del equipo por enfermedad	15/5/2018
R5	Dificultad de implementación del problema del enrutamiento	15/5/2018
R7	Choque de roles	15/5/2018
R9	Conflictos entre los integrantes del equipo	15/5/2018
R10	Dificultad de integración de Google Maps para el manejo de caminos	15/5/2018
R11	Pérdida de los archivos de elaboración del proyecto	15/5/2018
R12	Cancelación del proyecto por parte del cliente	15/5/2018
R13	Repartidores no se acostumbran a utilizar el software	15/5/2018

Identificador	Impacto	Probabilidad	Magnitud
R1	3	0,8	2,4
R2	3	0,8	2,4
R3	3	0,8	2,4
R4	3	0,8	2,4
R8	4	0,4	1,6
R6	4	0,4	1,6
R5	3	0,5	1,5
R7	3	0,5	1,5
R9	4	0,3	1,2
R10	2	0,4	0,8

R11	4	0,1	0,4
R12	5	0,3	1,5
R13	4	0,1	0,4

Reunión 2

Fecha: Fin Milestone 2 - 23/07/18

Funcionalidades terminadas

El equipo culminó todas las funcionalidades planificadas en la reunión de coordinación anterior para el fin del Milestone 2:

Como admin debo poder crear un reparto
Como admin debo poder ver la lista de repartos
Como admin debo poder ver los datos de un reparto
Como admin debo poder ver los datos de un pedido
Como admin debo poder ver la lista de pedidos
Como admin debo poder filtrar repartos
Como admin debo poder filtrar pedidos
Como admin debo poder administrar direcciones de destinatarios
Como admin debo poder filtrar las correcciones de direcciones
Como admin debo poder cancelar un reparto
El reparto debe crearse con las direcciones de destinatarios arregladas

Bugs sin corregir

No se reportaron bugs.

Retrospectiva

En gris: puntos que se mantienen de la retrospectiva anterior. Las que están en 'start' es porque no se comenzaron a implementar.

Start	Stop	Continue
Hacer que ambos integrantes estén familiarizados de todo el proyecto. El front-end debería conocer profundamente el funcionamiento del panel del administrador aunque lo desarrolle el back-end.		En este Milestone el front-end no tuvo casi trabajo y se encargó de tareas de gestión que normalmente se encarga la desarrolladora back-end. Se demostró que el equipo puede ser flexible en cuanto a las tareas para equilibrar el trabajo pero se debe balancear más la distribución back y front.
Documentar las sugerencias de		Se está llevando al día el

cambio que surgieron en el Milestone en la reunión de coordinación, al igual que lo que se hace con los bugs.		tablero de Trello.
Hacer pre-demos para no improvisar en las reuniones.		Se está llevando al día el documento de métricas.
Mejorar la forma de escribir los criterios de aceptación.		Se traquean las horas utilizando etiquetas como se definió en la retrospectiva anterior.
Hacer las demos más despacio para que el cliente tenga tiempo de dar su opinión y encontrar posibilidades de mejora.		Se está llegando a las funcionalidades del *Sprint* exitosamente. Está bien decidido el largo de los *Sprint*s. Están bien estimadas las historias de usuario hasta ahora.
		Las Reviews y Plannings son del tiempo correcto. Sin más ni menos.
		Se está utilizando correctamente GitFlow, y se está respetando el plan de PR e integración continua. Como resultado los códigos están prolijos y robustos.
		El Release Plan se están llevando al día y el cliente pidió acceso.
		El Wunderlist está siendo útil para sincronizar tareas y organizar dudas entre los integrantes.

Backlog Grooming

En el plan original se habían escrito las funcionalidades:

Milestone 3: Viaje del repartidor
Como repartidor debo poder iniciar el próximo reparto
Como repartidor en un reparto debo poder ver la próxima dirección en mi pedido
Como repartidor debo poder marcar un pedido como rebotado
Como admin debo poder ver los pedidos rebotados
Como repartidor debo poder marcar un pedido como entregado
Como admin debo poder ver los pedidos entregados

Como repartidor debo poder finalizar mi reparto automáticamente si terminé el último pedido
Como repartidor mi app debe funcionar aunque no tenga conexión, y debe actualizarse apenas la consiga

En la reunión de coordinación se planificaron las funcionalidades:

Milestone 3: Viaje del repartidor
Como repartidor debo poder iniciar el próximo reparto
Como repartidor en un reparto debo poder ver la próxima dirección en mi pedido
Como repartidor debo poder finalizar mi reparto automáticamente si terminé el último pedido
Como repartidor debo poder marcar un pedido como rebotado
Como admin debo poder ver los pedidos rebotados
Como admin debo poder hacer que un pedido requiera número de precinto al entregarse
Como repartidor debo poder marcar un pedido como entregado
Como admin debo poder ver los pedidos entregados
Como repartidor mi app debe funcionar aunque no tenga conexión, y debe actualizarse apenas la consiga
Como repartidor debo poder visualizar en un mapa el camino hacia el próximo destinatario
El sistema debe ordenar los paquetes de un reparto de forma de tener el camino más rápido

Se dividieron funcionalidades complejas y se agregaron funcionalidades como la del número de precinto que surgieron de sugerencias de cambio durante el Milestone anterior.

Pruebas

Qué se quiere probar	Entradas	Resultado Esperado	Funciona
Admin app			
Como admin debo poder crear un reparto			
Ingreso con datos válidos excepto por rut	Se ingresa reparto válido con RUT “asdasd” inválido	Muestra mensaje de error “Deben ser 12 dígitos”	OK
Ingreso con datos válidos excepto por fecha la cual se ingresa una anterior a la del momento	Se ingresa reparto válido con fecha “12/12/1900” invalida	Muestra mensaje de error “Debe ser en el futuro”	OK

Ingreso con datos válidos excepto por la descripción	Se ingresa reparto válido con descripción“” invalida	Muestra mensaje de error “Debe ser en el futuro”	OK
Ingreso con datos válidos excepto por el archivo	Se ingresa reparto con archivo con error en línea 4	Muestra mensaje “Error en línea 4”	OK
Ingreso con datos válidos	Se ingresa reparto válido	Muestra mensaje de aprobado	OK
Como admin debo poder ver la lista de repartos			
Ingreso a la lista de repartos	Se ingresa a la lista de repartos sin repartos agregados	Se muestra una tabla vacía	OK
Ingreso a la lista de repartos	Se ingresa al listado de repartos con repartos agregados	Se muestra una tabla con repartos	OK
Como admin debo poder ver los datos de un reparto			
Al ingresar un pedido debería ver sus detalles	Aprieto en el detalle del pedido	Se muestra los detalles del pedido	OK
Como admin debo poder ver la lista de pedidos			
Ingreso a la lista de pedidos	Se ingresa a la lista de pedidos sin pedidos agregados	Se muestra una tabla vacía	OK
Ingreso a la lista de pedidos	Se ingresa al listado de pedidos con pedidos agregados	Se muestra una tabla con pedidos	OK
Como admin debo poder filtrar repartos			
Ingreso a la lista de repartos y filtro por id	Se ingresa a la lista de repartos y se filtra por id válido “12”	Se muestra reparto	OK
Ingreso a la lista de repartos y filtro por id	Se ingresa a la lista de repartos y se filtra por id inválido “123123”	No se muestran repartos	OK
Ingreso a la lista de repartos y filtro por RUT	Se ingresa a la lista de repartos y se filtra por RUT válido “123456789101”	Se muestran repartos	OK
Ingreso a la lista de repartos y filtro por RUT	Se ingresa a la lista de repartos y se filtra por RUT inválido “12”	No se muestran repartos	OK

Ingreso a la lista de repartos y filtro por descripción	Se ingresa a la lista de repartos y se filtra por descripción válida “Canelones”	Se muestran repartos	OK
Ingreso a la lista de repartos y filtro por descripción	Se ingresa a la lista de repartos y se filtra por descripción inválida “12”	No se muestran repartos	OK
Como admin debo poder filtrar pedidos			
Ingreso a la lista de pedidos y filtro por id	Se ingresa a la lista de pedidos y se filtra por id válido “5”	Se muestra pedido	OK
Ingreso a la lista de pedidos y filtro reparto	Se ingresa a la lista de pedidos y se filtra por id inválido “123123”	No se muestran pedidos	OK
Ingreso a la lista de repartos y filtro por reparto	Se ingresa a la lista de pedidos y se filtra por reparto “Reparto 3”	Se muestran pedidos	OK
Ingreso a la lista de repartos y filtro por destinatario	Se ingresa a la lista de pedidos y se filtra por Reparto “Martin García”	Se muestran pedidos	OK
Ingreso a la lista de repartos y filtro por dirección del destinatario	Se ingresa a la lista de pedidos y se filtra por dirección “Oficial 12 N 15”	Se muestran pedidos	OK
Como admin debo poder administrar direcciones de destinatarios			
Ingreso datos válidos excepto por dirección original	Se ingresa datos válidos excepto por dirección original “”	Se muestra mensaje de error “no puede estar en blanco”	OK
Ingreso datos válidos excepto por dirección a mostrar	Se ingresa datos válidos excepto por dirección original “”	Se muestra mensaje de error “no puede estar en blanco”	OK
Ingreso datos válidos	Se ingresa datos válidos	Se crea nueva dirección	OK
Como admin debo poder filtrar las correcciones de direcciones			
Ingreso a listado de dirección es y selecciono por destinatario	Se selecciona el destinatario “Martin García”	Se muestra la dirección correspondiente	OK
Como admin debo poder borrar y recuperar un reparto			

Selecciona la opción de borrar en un reparto	Se selecciona borrar en “Reparto 3”	Se elimina el reparto	OK
Selecciona la opción de recuperar un reparto	Se selecciona recuperar “Reparto 3”	Se recupera el reparto	OK
El reparto debe crearse con las direcciones de destinatarios arregladas			
Se crea reparto con direcciones sin arreglar	Se sube archivo con direcciones para arreglar	No permite comenzar el reparto hasta completar arreglar las direcciones	OK
Se crea reparto con direcciones arregladas	Se sube archivo con direcciones arregladas	Se permite comenzar el reparto	OK
Como admin debo poder recalcular un reparto cuando corregí las direcciones de los destinatarios			
Se recalcula el reparto	Se arregla las direcciones y se aprieta en la opción de recalcular	Se recalcula el reparto y permite comenzarlo	OK

Análisis de riesgos

Identificador	Riesgo	Fecha de identificación
R1	Escasez de tiempo para dedicarle la atención necesaria al proyecto	15/5/2018
R2	Mala estimación de la velocidad del equipo	15/5/2018
R4	Mala estimación de una o varias historias	15/5/2018
R8	Cambios grandes en los requerimientos ya implementados	15/5/2018
R3	Demorar más de lo estimado por falta de dominio en desarrollo Android	15/5/2018
R5	Dificultad de implementación del problema del enrutamiento	15/5/2018
R6	Inhabilitación de un integrante del equipo por enfermedad	15/5/2018
R9	Conflictos entre los integrantes del equipo	15/5/2018
R10	Dificultad de integración de Google Maps para el manejo de caminos	15/5/2018
R11	Pérdida de los archivos de elaboración del proyecto	15/5/2018

R12	Cancelación del proyecto por parte del cliente	15/5/2018
R13	Repartidores no se acostumbran a utilizar el software	15/5/2018
R7	Choque de roles	15/5/2018

Identificador	Impacto	Probabilidad	Magnitud
R1	4	0,8	3,2
R2	3	0,8	2,4
R4	3	0,6	1,8
R8	3	0,6	1,8
R3	3	0,6	1,8
R5	3	0,6	1,8
R6	4	0,4	1,6
R9	5	0,2	1
R10	2	0,4	0,8
R11	4	0,1	0,4
R12	5	0	0
R13	4	0,1	0,4
R7	3	0,1	0,3

ANEXO 23 – “Actas de reunión”

Como se definió en el plan de comunicación el equipo definió enviar actas de reunión luego de las reuniones con el cliente o el tutor. Comenzado el proyecto el equipo evaluó y decidió descontinuar las recapitulaciones para los casos de reunión con el tutor pero continuó la de las reuniones con los clientes. Estas actas permitieron llevar un registro organizado de lo acordado en cada reunión para permitir entre otras cosas:

1. Aclarar malentendidos en la *Sprint* Review antes de comenzar a desarrollar.
2. Resumir las funcionalidades mostradas en la *Sprint* Plannings para recordarle al cliente qué es lo nuevo que tiene que probar.
3. Resumir el feedback y defectos mencionados que igualmente deben haber sido reportados en Trello pero que el cliente no suele acceder por su cuenta.
4. Evitar situaciones extremas en clientes que suelen mentir sobre lo acordado

Las actas de reunión se enviaron por correo y almacenaron en el repositorio remoto como se muestra a continuación.

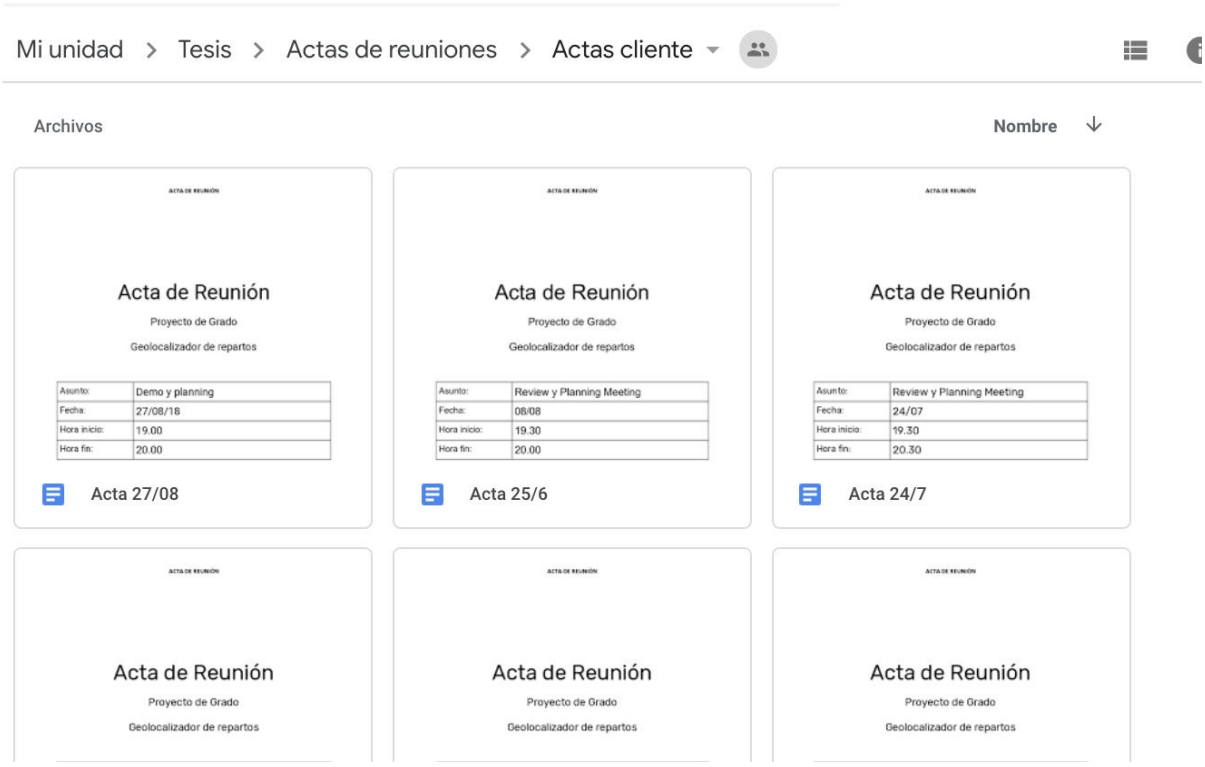


Ilustración 109 - Ejemplo de actas de reunión

A continuación se muestran un ejemplo de acta de reunión con el tutor y otro con el cliente enviadas durante el proyecto.

Acta de Reunión

Proyecto de Grado

Geolocalizador de repartos

Asunto:	Reunión pre-comienzo
Fecha:	31 de mayo

Tabla 33 - Información de la reunión

Participación

Asistieron:	Leticia Esperón, Rodrigo Arsuaga, Martín García, Nicolás Benítez
No asistieron:	-

Tabla 34 - Asistencias de reunión

Resumen desarrollo de la reunión

- Se mostraron y aprobaron los cambios realizados a los prototipos de las pruebas realizadas la reunión pasada.
- Se mostraron las herramientas (Invision, Trello) y se explicó la metodología de trabajo a utilizar.
- Se pidió un poco más de información necesaria para la documentación.
- Se discutieron a grandes rasgos los requerimientos complementarios y surgieron nuevas ideas que fueron registradas en Trello.
- Se tuvo una pequeña charla técnica sobre tecnologías y posibles soluciones.

Se resuelve

Metodología:

- Se trabajará en un marco de trabajo ágil, aplicando la metodología Scrum.
- Se le llamará *Sprint* al tiempo de desarrollo entre una Review y una Demo.
- Planning se le llama a la instancia donde el equipo planea el trabajo a realizar en el *Sprint*.
- Demo es la instancia en la que el equipo expone el trabajo realizado en el *Sprint* y el cliente aprueba y/o da feedback del mismo.
- Se acordó trabajar en *Sprint*s de dos semanas, y agrupar las Plannings y Demos los lunes a las 19.30 hrs.

- Release Map es el plan que dice en qué *Sprint* se desarrollará cada funcionalidad agrupando las funcionalidades en Milestones que son objetivos de la aplicación en cuanto a funcionalidad.

Mock ups:

- Cambiar botón de Enviar por algo más sencillo de entender.
- No mostrar los envíos completados.
- Se aprobó el estilo del diseño y la tipografía.

Requerimientos:

- Va a ser necesario introducir un ABM básico de destinatarios.
- Va a ser necesario identificar los clientes de alguna manera. Se considerará que el back-end consuma la base de datos de clientes que tiene la empresa en otra aplicación. Se agrega como requerimiento al Icebox.
- Se cambió el flujo de creación de un reparto. Antes se pensaba que el pedido se introducía antes que el reparto. Ahora se acordó que al crear un nuevo reparto se ingresa el Excel (uno o más) de pedidos, el cliente, y se asigna manualmente en el momento o luego los dos repartidores a cargo.
- El cliente aprueba que la app móvil soluciona su problema.

Tecnologías:

Equipo investigará y considerará:

- Tiddle para el manejo de sesiones
- JB o FastJson api como sustituto de Jbuilder

Se acordó Active Admin para el panel de administrador, dejando la posibilidad de realizar el mapa aparte, pero se discutirá cuando llegue el momento.

Para el manejo de mapas se planteó utilizar Weiss en lugar de Google Maps porque a veces el mejor camino es más acertado. Sin embargo, se destacó que Google Maps es más “fiable" en cuanto a que no deje de existir.

Testing:

- Se discutió el punto del ambiente “beta" para testing acordando en el uso de un ambiente Staging o Develop previo al mismo.
- Se acordó que el equipo recibiría un celular Android para poder desarrollar.
- El cliente aceptó sacar la aplicación a la vida real, haciendo que se utilice por los cadetes en repartos pequeños a medida que la aplicación tenga funcionalidades validables.

Pendientes

1. Crear cuentas de Heroku y Google Play Store
2. Dar dispositivo móvil al equipo
3. Cliente debe revisar la lista de historias a medida que se estén por desarrollar, priorizar y comentar modificaciones necesarias.

Próxima reunión

Fecha y hora: lunes 11 de junio

Agenda tentativa: *Sprint* demo y Planning

Acta de Reunión

Proyecto de Grado

Geolocalizador de repartos

Asunto:	Reunión con tutor para discutir avances
Fecha:	09/05/2018
Hora inicio:	19:00
Hora fin:	20:15

Tabla 35 - Información general de la reunión

Participación

Asistieron:	Leticia Esperón, Rodrigo Arsuaga, Pablo Hernández
No asistieron:	-

Tabla 36 - Asistencias de reunión

Agenda

1. Comentarios y preguntas sobre el trabajo asignado para esta reunión
2. Planificación de trabajo para la próxima reunión

Desarrollo de la reunión

Correcciones sobre trabajo anterior:

- En la sección Cronograma se tiene que poder visualizar un calendario.
- Matriz de interesados: Al poner una gráfica poner leyendas y unidades y explicar la escala.
- Utilizar los términos de forma consistente. Cuando se habla de una persona se debe usar siempre el mismo nombre. Si se referencia a Martín García como Martín García, se debe mantener en todo el documento. El término “cliente" tiene que ser explicado en el glosario. Para identificar el software utilizar la misma palabra siempre.
- Planes de costos y adquisiciones no van en nuestro proyecto.
- La etapa de desarrollo es casi seguro que finalice mucho después de la fecha original planeada, por lo que es mejor planificarla para fines de diciembre y será evaluada a medida que avanza el cronograma.

Comentarios sobre trabajo a realizar:

- Calendario y reglamento pueden ser parte del plan de gestión de RRHH.
- Para el SCM del Drive se pueden organizar las carpetas como columnas de un Kanban.
- En el SCM también se define la nomenclatura de los archivos. Es válido manejar un documento por tema como Calidad, Arquitectura, etc.
- Definir cada cuánto se hacen pruebas de integraciones.
- Verificar que estemos incluyendo los requerimientos no funcionales en la lista de requerimientos para no olvidarlos al estimar o priorizar.

Otros comentarios:

- Se define usar un acta de reunión estructurada para la recapitulación de reuniones.
- Pablo aprobó la fecha de inicio del *Sprint* 0 propuesta en el calendario (04/06/18).
- Pablo no recomienda darle al cliente una fecha de finalización de la etapa de desarrollo desde ya, y recomienda postergar la estimación del alcance final para luego de calculada la velocidad del equipo.

Se resuelve

Tareas por realizar para la semana que viene:

1. Correcciones trabajo semana anterior
2. Ver documentos 302 y 303
3. Definir ciclo de vida y metodología
4. Comenzar a trabajar en el Plan de SCM
5. Comenzar a trabajar en el Plan de gestión de riesgos
6. Comenzar a trabajar en el Plan de calidad

7. Comenzar a trabajar en el Plan de testing

Pendientes

1. Averiguar reservación de salas de reuniones en bedelía.

Próxima reunión

Fecha y hora: miércoles 16/05/18, 19:30 hrs

Lugar: Universidad Ort Uruguay

Agenda tentativa:

- Comentarios y preguntas sobre el trabajo asignado para esta reunión
- Planificación de trabajo para la próxima reunión

ANEXO 24 – “Conclusiones de revisiones formales de Universidad ORT Uruguay”

8.11. Revisión 1

Fecha: 20/06/2018

Revisor: Rafael Bentancour

Fortalezas del Grupo

El grupo se encuentra avanzado en la especificación de requerimientos, habiendo definido ya los planes y acuerdos necesarios previos a la etapa de desarrollo y encontrándose empezando esta etapa con los requerimientos y diseños definidos.

Se evaluó que el grupo hizo un gran esfuerzo en la especificación de requerimientos, prototipos y diseño en lo cual se debe hacer más hincapié al presentar.

Rafael sugiere mejor la presentación para que refleje mejor todo el trabajo que hubo detrás de las decisiones. Se señala que el equipo está muy avanzado en el proceso y que cuenta con mucha organización y altos estándares y que todo eso debe ser transmitido en la presentación. Se sugiere trabajar con el tutor para lograrlo.

Oportunidades de Mejora

El grupo se presenta ante un par de desafíos técnicos mencionados por el revisor que no habían sido totalmente considerados por el grupo, como por ejemplo la dificultad al transformar en coordenadas una dirección cuyo formato no esté correctamente definido. El equipo consideró agregar un *Spike* para poder investigar cómo enfrentarse a este y otros desafíos técnicos como el armado del camino más rápido, previo al *Sprint* donde se desarrolle.

Se recomendó o surgió en base a intercambios:

- Que cuando el producto ya se pueda probar, el grupo vaya a hacer el recorrido junto con los repartidores. El grupo consultará con la empresa si esto es posible,
- Recordar antes de la prueba instalar alguna herramienta para mejorar recolección de incidentes, posiblemente que permita al equipo visualizar los clicks que hace el repartidor en la aplicación, permitiendo evaluar la usabilidad y detectar errores comunes o posibles defectos.

Acciones a realizar sobre la presentación

- Explicar mejor la relación entre problema y solución. Representar visualmente un mapeo entre cada problema y la solución.
- En la parte de ingeniería de requerimientos, hacer énfasis en qué técnicas se usaron en las tres etapas: reuniones para especificar los requerimientos, que

se especificaron mediante historias de usuario, Story Map y prototipos, y que fueron posteriormente validados por el cliente.
- Mencionar en la presentación del cliente la cantidad de repartidores que tiene la empresa cliente para dar magnitud al problema.
- Agregar un RNF sobre la precisión de la ruta elegida.
- Vincular los objetivos iniciales con las actividades de calidad. Justificar que el aseguramiento de la calidad habla de seguir estándares y herramientas de análisis del código, etc. desde el principio porque al ser un grupo de solo dos personas se busca evitar el retrabajo para maximizar la eficiencia.
- En la presentación oral, mencionar al hablar de las tecnologías utilizadas que se evaluaron otras alternativas. Poner como ejemplo Heroku ya que es una de las decisiones que es más relevante justificar.
- En las siguientes revisiones cuando se tengan posteriores versiones de los riesgos, mostrar una evolución de los mismos.
- Mejorar la visibilidad de la matriz de riesgos. Agregar una diapositiva que se enfoque en dos ejemplos para poder leerlos.
- La columna de plan de respuesta en realidad contiene la estrategia. Escribir concretamente cómo implementar la estrategia o sea para cada riesgo cómo se mitigaría.
- Cambiar en la tabla de gestión de defectos el título “Máximo de defectos permitidos" por “(...)detectados no corregidos”.
- Sacar fondo de la tortuga del Trello.
- Agregar foto de repartidor en la diapositiva de la demo.

8.12. Revisión 2

Fecha: 11/10/2018

Revisor: Rafael Bentancour

Fortalezas del Grupo

El grupo se encuentra avanzado en el desarrollo, actualmente terminando de ajustar algunos detalles del algoritmo propio de enrutamiento. Se felicitó el buen trabajo que se había realizado en la construcción del algoritmo, haciendo uso de condiciones del contexto.

Se destacó el gran énfasis que se puso en la calidad de código y en lo rigurosamente que se fueron siguiendo los planes que se definieron.

Oportunidades de Mejora

Tanto Rafael como Pablo notaron al equipo demasiado ansioso con la resolución de esto, ya que estaban muy preocupados por el atraso que había implicado (el doble de lo que se había planificado, 2 *Sprints* en total). Se sugirió pausar para distenderse, tomar distancia y evaluar el progreso.

Nuevamente se mencionó que el equipo no estaba reflejando correctamente en la presentación todo el trabajo que realizó, el cual se evidencia luego de que se indaga al grupo por ciertas decisiones y se descubre el proceso que hubo detrás. Se sugiere manifestar esto en la presentación.

Se menciona que las debilidades del grupo se encuentran en la especificación de la arquitectura y de la calidad, donde si bien se hizo un gran trabajo, no queda claro las acciones concretas o la arquitectura del sistema en la presentación. Se sugiere cambiar el formato de estas partes, mostrando la arquitectura a través de los RNFs. Se sugiere listar los atributos de calidad priorizados, explicando para un par por qué fueron importantes, cómo se midieron y cómo se mantuvieron.

Acciones a realizar sobre la presentación

- Trabajar en el orden de la presentación
- Arreglar la parte de arquitectura y calidad para reflejar lo descrito anteriormente
- Mencionar el análisis heurístico de Nielsen en la presentación
- Explicar cómo se realizó la estimación
- Explicar en qué consiste el Story Map y cómo y para qué se realizó

8.13. Revisión 3

Fecha: 26/11/2018

Revisor: Álvaro Ortas

Fortalezas del grupo

El equipo trabajó mucho en la presentación comparado con la vez pasada, y esta vez el revisor mencionó que estéticamente era muy linda. Se mencionó que el orden estaba bien, que era fluido.

Como comentarios generales, el revisor motivó al equipo y mencionó su potencial para una nota de excelencia. Sugirió revisar los puntos débiles y asegurarse de transmitir todo el trabajo en la documentación final.

Oportunidades de mejora

Se recomendó resumir los datos en la presentación. Se dijeron demasiados datos. Considerar que durante la defensa final el revisor va a haber leído el documento y no va a ser necesario detallar todo.

Se sugirió que si hay que recortar tiempo en la presentación final se puede comenzar por la gestión de riesgos.

Se sugirió practicar mejor la demo guiada y pensar cómo se realizará en la defensa final, dada la complejidad de mostrar algunas funcionalidades sin salir con el repartidor.

Acciones a realizar sobre la presentación

- Explicar mejor el proceso actual de la empresa en los repartos. Es importante, detallarlo, porque es complejo e impactó en las decisiones del producto. Considerar vídeo animado.
- Explicar mejor por qué se puso tanto énfasis en la mantenibilidad del sistema: el cliente lo solicitó.
- Mostrar la evolución de los riesgos con mayor nitidez.
- Explicar el vínculo de los objetivos iniciales con las actividades de calidad realizadas y las métricas que se utilizaron. Todavía no se puede porque el proyecto no ha terminado pero en la presentación final describir las acciones que el equipo fue tomando a partir de la evolución de las métricas.
- Mejorar la visibilidad de la matriz de riesgos. Mostrar todo y focalizar en dos ejemplos.
- Explicar que la gestión de riesgos no fue muy utilizada en el proyecto. Que se realizó pero que no dirigió el desarrollo, lo cual es totalmente válido porque el equipo se gestionó de forma distinta.
- Agregar a la lista de RNF la precisión de la ruta elegida. que sea preciso y consistente
- Agregar como un atributo de calidad el no repudio, ya que el destinatario final no puede negar haber recibido el paquete.
- Destacar lo importante que fue la gestión de cambios. Explicar que si bien parece un procedimiento muy riguroso fue de mucha ayuda.

ANEXO 25 – “Utilización del Trello para seguimiento de tareas”

A continuación se describen las columnas utilizadas resumiendo su funcionalidad.

1*. Icebox*

En la columna *Icebox* se colocaron las solicitudes de cambio, esto es, ideas para nuevas funcionalidades que surgen durante las reuniones. El proceso para las solicitudes de cambio se detalla en la sección 6.6.5 - Control de cambios. Una vez que las solicitudes de cambio se aceptaban para entrar en el alcance final, se agregaban al *Product Backlog*.

2. *Product Backlog*

En esta columna se colocaron las historias de usuario que el equipo planeó incluir en el alcance final. La columna se intentó mantener en orden de prioridad descendente constantemente para facilitar la planeación de los *Sprints*. Esta columna fue evolucionando a lo largo del proyecto ya que se agregaron, quitaron, reordenaron y modificaron algunas historias de usuario. Todos los cambios en el *Product Backlog* eran propuestos o se validaban con el cliente. Al finalizar la etapa de desarrollo no quedaron historias en esta columna.

3. *Bugs*

Los defectos del producto reportados se colocaban en esta columna y luego se trataban como historias de usuario comunes a medida que se comenzaba a trabajar en ellos. Los detalles de cómo se gestionaron los defectos se encuentran en la sección 113 - Gestión de incidentes.

4. *Feedback*

Se consideró *feedback* a pequeños cambios sobre funcionalidades ya implementadas que surgieron durante las diferentes instancias de validación. Las tarjetas de *feedback* podrían llegar a contener una estimación si se trataba de cambios no inmediatos. Se trató de incluir el *feedback* en el *Sprint* inmediato al *Sprint* en el que surgía para agilizar e incentivar la retroalimentación.

5. *Sprint Backlog*

Las historias de usuario a trabajar durante el *Sprint* se movían del *Product Backlog* al *Sprint Backlog* durante la *Sprint Planning*. El equipo ya tenía una idea del plan antes de la reunión gracias a que el *Product Backlog* era constantemente reorganizado junto al cliente según su prioridad. De todos modos, en la *Sprint Planning* se volvía a validar

el plan con el cliente, se discutían los detalles de las funcionalidades a implementar y se acordaban sus criterios de aceptación en casos de requerimientos medio complejos.

6. En Progreso

Durante el *Sprint*, a medida que un integrante comenzaba a trabajar en una historia de usuario la movía de *Sprint Backlog* a *En Progreso*. Dado a que ambos integrantes trabajaron en persona la mayoría del tiempo por lo que sabían en qué estaba trabajando el otro, esta columna no fue de particular utilidad más que para visualizar con un poco mayor de facilidad las tareas que faltaban completar para el *Sprint*.

7. Terminadas

A medida que un integrante, o ambos en el caso de funcionalidades que requirieron tanto *front-end* como *back-end*, terminaba una funcionalidad, se movía a esta columna. No se consideraba como terminada una funcionalidad hasta que el *Pull Request* que la agregaba contenía pruebas unitarias, había sido integrado y la funcionalidad había sido probada con pruebas de caja negra para verificar su correcto funcionamiento según el criterio de aceptación.

Esto permitió reducir el retrabajo y los *bugs* ya que las funcionalidades no se daban por terminadas hasta que el equipo no estaba totalmente seguro de que cumplía con lo acordado.

8. Aprobadas

Las funcionalidades trabajadas durante el *Sprint* se verificaban y validaban junto al cliente durante la *Sprint Review*. Recién luego de que el cliente las vio y aprobó, se movían a Aprobadas. Idealmente, al terminar una reunión, todas las historias de usuario trabajadas en el *Sprint* se encontraban en Aprobadas, y se habían creado nuevas tarjetas con *feedback* en la columna *feedback*. En los *Sprints* donde alguna funcionalidad no pudo ser terminada, la historia permanecía en *En Progreso* y se contaba como *carry-over* al próximo *Sprint*.

ANEXO 26 – “Métricas”

Esta sección tiene como objetivo describir las métricas que el equipo definió registrar, explicando cuál fue la razón por las que las consideró importante. Los valores medidos a lo largo del proyecto se presentan y analizan en los capítulos correspondientes del documento.

Incidentes

Las métricas de incidentes se subdividieron por categoría ya que se consideró que tenían rangos deseados diferentes.

Se midieron por *Sprint* ya que esto permitiría una evaluación periódica y mejoraría la toma de decisiones para el próximo *Sprint*. Permitirían tomar decisiones como si se requiere bajar la velocidad, incrementar las pruebas previas o posteriores a la entrega del software, etc. También son necesarias para saber si hay que aplicar lo especificado en el plan de gestión de incidentes que define una máxima cantidad de defectos tolerados por *Sprint*.

Las métricas de incidentes reportados tienen en común que valores muy bajos de ellas podrían ser síntomas de falta de prueba del software una vez entregado, mientras que valores muy altos podrían ser síntoma de falta de prueba de las funcionalidades antes de darlas por terminadas.

Título	**Defectos de prioridad 1 (Alta) reportados**
Objetivo	Identificar faltas graves durante el testing previo a la entrega del software para promover el desarrollo sin errores.
Unidad	Cantidad / *Sprint*
Rango deseado	Entre 0 y 2.
Cuándo se mide	Al finalizar cada *Sprint*

Tabla 37 - Información de defectos de prioridad alta reportados

Título	**Defectos de prioridad 2 (Media) reportados**
Objetivo	Identificar faltas de severidad media durante el testing previo a la entrega del software para promover el desarrollo sin errores.
Unidad	Cantidad / *Sprint*
Rango deseado	Entre 0 y 2.

Cuándo se mide	Al finalizar cada *Sprint*

Tabla 38 - Información de defectos de prioridad media reportados

Título	**Defectos de prioridad 3 (Baja) reportados**
Objetivo	Identificar faltas de severidad baja durante el testing previo a la entrega del software para promover el desarrollo sin errores.
Unidad	Cantidad / *Sprint*
Rango deseado	Entre 3 y 10.
Cuándo se mide	Al finalizar cada *Sprint*

Tabla 39 - Información de defectos de prioridad baja reportados

Título	**Defectos de prioridad 1 (Alta) solucionados**
Objetivo	Asegurar que el proceso seguido dé lugar al arreglo de defectos, permitiendo cumplir con la máxima cantidad de defectos de cada categoría permitidos. Asegurar que el proceso priorice el arreglo de defectos de prioridad alta.
Unidad	Cantidad / *Sprint*
Rango deseado	Los que sean necesarios para cumplir con la máxima cantidad de defectos de cada categoría permitidos.
Cuándo se mide	Al finalizar cada *Sprint*

Tabla 40 - Información de defectos de prioridad alta solucionados

Título	**Defectos de prioridad 2 (Media) solucionados**
Objetivo	Asegurar que el proceso seguido dé lugar al arreglo de defectos, permitiendo cumplir con la máxima cantidad de defectos de cada categoría permitidos.
Unidad	Cantidad / *Sprint*
Rango deseado	Los que sean necesarios para cumplir con la máxima cantidad de defectos de cada categoría permitidos.
Cuándo se mide	Al finalizar cada *Sprint*

Tabla 41 - Información de defectos de prioridad media solucionados

Título	Defectos de prioridad 3 (Baja) solucionados
Objetivo	Asegurar que el proceso seguido dé lugar al arreglo de defectos, permitiendo cumplir con la máxima cantidad de defectos de cada categoría permitidos. Asegurar que el equipo no invierta demasiado tiempo en el arreglo de defectos de baja prioridad cuando tiene nuevas funcionalidades por construir.
Unidad	Cantidad / *Sprint*
Rango deseado	Los que sean necesarios para cumplir con la máxima cantidad de defectos de cada categoría permitidos.
Cuándo se mide	Al finalizar cada *Sprint*

Tabla 42 - Información de defectos de prioridad baja solucionados

Scrum

Las métricas relacionadas con Scrum involucran la velocidad del equipo y los puntos planificados realizados en cada iteración. Esto permitiría facilitar la planeación del trabajo a realizar en las siguientes iteraciones, con el objetivo de reducir la probabilidad de tener carry-over. Permitiría además evidenciar posibles errores en la estimación de ciertas historias.

Título	Velocidad del equipo
Objetivo	Llevar un rastro del esfuerzo en puntos que el equipo fue capaz de hacer en cada iteración, para facilitar las próximas planeaciones.
Unidad	Puntos / *Sprint*
Rango deseado	Mínimo de 12 que fue la velocidad segura planificada para el primer *Sprint*. Se espera que incremente con el tiempo.
Cuándo se mide	Al finalizar cada *Sprint*

Tabla 43 - Información de la velocidad del equipo

Título	Puntos planificados
Objetivo	Compararla con la métrica de puntos realizados (velocidad) para evidenciar errores en la estimación de la velocidad del equipo o del esfuerzo de algunas historias de usuario.
Unidad	Puntos / *Sprint*

Rango deseado	Mínimo de 12 que fue la velocidad segura planificada para el primer *Sprint*. Se espera que incremente con el tiempo.
Cuándo se mide	Al finalizar cada *Sprint*

Tabla 44 - Información de los puntos planificados

Proceso

Uno de los objetivos del proceso fue mantener el trabajo constante a lo largo de todo el proyecto. Para eso fue necesario medir las horas trabajadas durante todo el proyecto, no solo en la fase de desarrollo. Es por esto que la siguiente métrica se mide por semana y no por *Sprint*.

Por otro lado uno de los objetivos del equipo fue mantener el retrabajo al mínimo, por lo que medir el porcentaje del mismo es necesario. Para ello una de las áreas a medir fue el retrabajo.

Finalmente, un proceso que presente demasiado porcentaje en reuniones y tareas de gestión podría ser síntoma de ineficiencia, siendo otro motivo para rastrear las horas según áreas.

Título	**Horas trabajadas por integrante por semana por área**
Objetivo	Balancear el trabajo a lo largo del proyecto. Evidenciar desbalances entre el trabajo de los integrantes. Poder medir el retrabajo y la eficiencia del proceso.
Unidad	Horas / Semana
Rango deseado	Mínimo de 12 que fue la velocidad segura planificada para el primer *Sprint*. Se espera que incremente con el tiempo.
Cuándo se mide	Al finalizar cada *Sprint*

Tabla 45 - Información de horas trabajas por integrante

ANEXO 27 – “Análisis heurístico”

Para analizar la usabilidad del sistema se realizó un análisis basado en las heurísticas de Nielsen sobre la aplicación móvil de repartidores y la web administradora. En el análisis se detectaron problemas para los cuales se realizó una recomendación y se le asignó una severidad.

Los problemas se clasificaron según su severidad utilizando una escala del 1 al 4 según el siguiente criterio:

Severidad	Descripción
1	Problema menor de usabilidad: la solución puede esperar al próximo proceso de rediseño de la interfaz.
2	Problema medio de usabilidad: solucionar el problema es relevante, debería ser incorporado a las tareas de mantenimiento.
3	Problema mayor de usabilidad: es importante su solución y debería asignársele alta prioridad.
4	Problema grave de usabilidad: es imperativa su solución antes de liberar el sistema o de inmediato si el sistema está en producción.

Ilustración 110 - Clasificación de problemas revelados en el análisis heurístico

En la siguiente tabla se puede apreciar el análisis realizado.

Análisis aplicación web administradores		
Problema	Recomendación	Severidad
Heurística: Coincidencia entre el sistema y el mundo real		
Al filtrar repartos, en caso de no encontrar ninguno aparece “No results found”	Cambiar a “No se han encontrado repartos”	2
Estados de pedidos aparecen en varios lenguajes	Cambiar los estados a español únicamente	3
Heurística: Consistencia y estándares		
Hay secciones en distintos idiomas	Convertir todo al español	3
Heurística: Prevención de errores		
Al eliminar un reparto no pide	Agregar mensaje de confirmación para	2

confirmación	evitar eliminados no deseados	
Heurística: Reconocer es mejor que recordar		
Cuando se tiene un reparto con direcciones sin geolocalizar no hay acceso directo a cómo corregirlas. Hay que ir a través de varias pantallas para poder hacerlo.	Agregar un link directo para poder arreglar la dirección.	3
Heurística: Flexibilidad y eficiencia de uso		
La acción de recalcular un pedido implica demasiados pasos	Agregar la acción a la lista de acciones en el listado de repartos	3
Heurística: Ayudar a los usuarios a recuperarse de errores		
Al ingresar un xml de órdenes con errores notifica que hubo error pero no cuál	Mostrar las líneas que tienen error y cuál es	3
Al ingresar una cédula incorrecta marca un mensaje de error pero no dice como es el formato esperado para una cédula	Agregar un mensaje más representativo del error	2
Heurística: Visibilidad del estado del sistema		
Luego de recalcular un reparto, se recarga la página pero no se explica qué pasó.	Agregar una notificación no invasiva y que no requiera click para ocultar, que avise que el reparto fue recalculado.	1
Luego de eliminar un reparto, un pedido o cualquier otro modelo se redirige al listado pero no se explica si la acción tuvo efecto o no.	Agregar una notificación no invasiva y que no requiera click para ocultar, que avise que el recurso fue eliminado.	1
Al crear un reparto, el reparto no se crea hasta después de un rato porque se está procesando en segundo plano. Hay que recargar la página para ver el nuevo reparto después de un rato.	Crear el reparto en estado de “subiendo archivo de pedidos" así el usuario entiende lo que está pasando.	3
Análisis aplicación móvil repartidores		
Problema	Recomendación	Severidad
Heurística: Coincidencia entre el sistema y el mundo real		

Textos dicen órdenes. Cuando los repartidores entienden por pedido	Convertir los textos a que digan pedido	2
Hay faltas de ortografía por el sistema	Corregir faltas	2
Heurística: Control y libertad para el usuario		
Al aceptar o rechazar no hay forma de deshacer la acción	Agregar una manera de poder volver para atrás	2
Heurística: Prevención de errores		
Cuando marcan direcciones sin geolocalizar no hay acceso directo a cómo arreglarlo. Hay que ir a través de varias pantallas para poder hacerlo	Agregar un link directo para poder arreglar la dirección.	
Heurística: Flexibilidad y eficiencia de uso		
Hay pasos innecesarios en ciertos pedidos que no requieren foto	Quitar la opción de agregar foto en estos casos	3
Heurística: Ayudar a los usuarios a recuperarse de errores		
Al cambiar contraseña, no dice el formato necesario. Solo que es invalida	Agregar mensaje con el formato necesario para la contraseña	2
Al ingresar una contraseña sin la mínima longitud necesaria dice que hubo un error pero no dice cuál	Explicar que el error está vinculado con la longitud de la contraseña y especificar la longitud mínima	1
Heurística: Visibilidad del estado del sistema		
Luego de entregar o rebotar un pedido, se vuelve automáticamente a la lista de pedidos pero no se explica si la acción tuvo efecto.	Agregar una notificación no invasiva y que no requiera click para ocultar, que avise que el pedido se marcó correctamente.	2
Al entrar a la aplicación la pantalla aparece en blanco hasta que de pronto aparece el botón de comenzar un	Agregar mensaje que diga que se están buscando los datos del reparto	2

reparto cuando este pudo ser traído del servidor.		
Al cambiar contraseña solo vuelve a la pantalla inicial sin decir si tuvo éxito el cambio	Agregar nuevo mensaje que confirme estado del cambio de contraseña	2

Tabla 46 - Análisis heurístico del sistema

ANEXO 28 – “Análisis de herramientas de repositorio”

Para elegir la herramienta de repositorio se comparó un modelo distribuido, Git y un centralizado, SubVersion.

A continuación se presentan las principales características y ventajas de cada uno.

Git

- **Privacidad:** Al ser un modelo distribuido el desarrollador puede dejar que otros vean solo lo que es necesario, pudiendo trabajar local y privadamente hasta que lo quiera compartir.
- ***Commits* rápidos y permite trabajar sin conexión:** Al ser un modelo distribuido, como no se sincroniza automáticamente, se hace *commit* localmente y por lo tanto es más rápido e incluso no se precisa conexión a Internet para comenzar a trabajar localmente.
- ***Branching* y *merging* muy fácilmente:** Crear branches para bifurcar el trabajo es muy sencillo y permite hacer una caja de arena de las funcionalidades en las que se trabaja hasta que estén listas para el flujo principal.
- **El flujo de trabajo es flexible:** En comparación con VCS centralizado, git tiene las cualidades que le permiten elegir su propio flujo de trabajo. Puede ser tan simple como un flujo de trabajo centralizado o más complejo y jerárquico. Esto permite adaptarlo según apropiado para el proyecto.
- **La integridad de los datos está asegurada:** Dado que git utiliza árboles SHA1, la corrupción de los datos debido a razones externas se puede detectar fácilmente.
- **Integración con Heroku:** Permite la integración fácilmente con *Heroku ya* que con un simple push se puede enviar una rama al servidor y este se encarga de compilar el código en esa versión y desplegarla.

SVN

- **Modelo centralizado:** El repositorio se encuentra únicamente en un servidor y cada commit se hace directamente al servidor. Esto puede facilitar la sincronización entre los desarrolladores pero no ayuda cuando se tiene un equipo grande trabajando en el mismo proyecto.
- **Performance en proyectos antiguos:** Al no tener que retener todo el historial del repositorio se obtienen beneficios de performance en proyectos con mucho historial.
- **Control de acceso:** *SVN* permite el control de acceso por archivo a los distintos usuarios, asignando distintos permisos a cada uno, creando una jerarquía de acceso.

Dadas las necesidades del equipo y la experiencia del mismo. El equipo decidió trabajar con *Git* por su fácil uso, la integración con Heroku y la experiencia que el equipo tiene con ella.

ANEXO 29 – “Análisis de herramientas de gestión”

Este análisis se hizo al comienzo del proyecto para identificar las mejores herramientas de gestión y definir la que el equipo utilizaría durante todo el proceso. La elección de la herramienta apropiada para el proyecto busca maximizar la eficiencia durante el desarrollo.

Para el análisis de herramientas de gestión se consideraron las más populares o conocidas en metodologías ágiles. Las herramientas evaluadas fueron Asana, Jira y Trello.

A continuación se presentan las ventajas y desventajas de cada una.

Asana

Ventajas	Desventajas
Es gratis	Como provee muchas funcionalidades, a veces puede ser difícil encontrar las herramientas que necesitas. Curva de aprendizaje alta.
Provee plantillas de *checklist* útiles para reciclar los criterios de aceptación.	Puede ser difícil encontrar la información que necesitas a través de la búsqueda.
Se integra con otros recursos de diseño como los prototipos en InVision	La interfaz de usuario es torpe a veces, especialmente cuando se trabaja con varios elementos. A veces las cosas simplemente desaparecen o no se guardan.
Se integra con Slack y Google Calendar para facilitar la sincronización entre integrantes	Solo está disponible en inglés (uno de los clientes no maneja el idioma)

Tabla 47 - Ventajas y desventajas de Asana

Jira

Ventajas	Desventajas
Tiene las herramientas de reportes y gráficos que se suelen utilizar en metodologías ágiles	Curva de aprendizaje alta, no sería fácil capacitar a los clientes para utilizarla con ellos en las reuniones
Tiene manejo de “issues" útil para traquear los defectos de una historia	Pago, empieza en 10 USD por mes

Interfaz muy rápida	Difícil de configurar al principio, muchas configuraciones necesarias antes de empezar

Tabla 48 - Ventajas y desventajas de Jira

Trello

Ventajas	Desventajas
Interfaz muy fácil de usar y entender, permitiría a los clientes sin experiencia utilizarlo sin marearse.	Puede ser demasiado simple en su versión gratuita, no teniendo acceso a las *Sprint* Burndown charts y otras herramientas que podrían ser de utilidad para el equipo.
Muchos add-ons para integrarle, por ejemplo para facilitar la asignación de esfuerzo a las tarjetas, sumar los totales, colapsar columnas, etc.	No ofrece una función de edición sin conexión que sería muy útil para trabajar en lugares públicos.
Admite integraciones con numerosas plataformas que ya usamos (por ejemplo, Slack, Google Drive)	No presenta funcionalidades de traqueo del tiempo, reportes o administración de defectos
El equipo ya tiene experiencia con la plataforma	Funciona trancado en proyectos con más de 200 tarjetas y 10 columnas
Buen historial de todos los movimientos de las tarjetas, útil para visualizar cuándo se comenzaron o completaron	

Tabla 49 - Ventajas y desventajas de Trello

El equipo se decidió por la utilización de Trello debido a su simpleza y fácil uso y la experiencia previa con la herramienta. Trello permitió asignar tareas a los integrantes y visualizar el estado de las historias con facilidad, sin la sobrecarga de herramientas innecesarias que proveen las alternativas.

ANEXO 30 – “Release Plan”

Esta sección tiene como objetivo resumir el estado final Release Plan, esto es, el desarrollo de los *Sprint*s, especificando en cada uno las funcionalidades, defectos o mejorar en las que se trabajó.

En la realidad el Release Plan se llevó en Google Sheets [38] de la forma que se muestra a continuación, pero se resumió al formato tabla para simplificar la lectura.

Sprint:		SPRINT 1		SPRINT 2		SPRINT 3		SPRINT 4		
FUNCIONALIDAD / SEMANA		S1	S2	S3	S4	S5	S6	S7	S8	
Puntos estimados	Velocidad	28/05 - 04/06	04/06 - 11/06	11/06 - 18/06	18/06 - 25/06	25/06 - 02/07	02/07 - 09/07	09/07 - 16/07	16/06 - 23/07	23/
Milestone 1: Administración de repartidores										
Setup repositorio Ruby		X								
Setup repositorio Andoird		X								
Setup Heroku		X								
Setup Google Play Store o equivalente		X								
Ícono de la app y pantalla de inicio				X						
Como admin debo poder loguearme a la web con usuario y contraseña				X						
Como admin debo poder crear usuarios de la app para los repartidores				X						
Como admin debo poder filtrar los repartidores				X						
Como admin debo poder ver repartidores				X						
Como repartidor debo poder loguearme				X						
Como repartido debo poder cerrar sesión				X						
Milestone 2: Administración de pedidos y repartos										
Como admin debo poder crear un reparto						X				
Como admin debo poder ver la lista de repartos						X				
Como admin debo poder ver los datos de un reparto						X				
Como admin debo poder ver los datos de un pedido						X				
Como admin debo poder ver la lista de pedidos						X				
Como admin debo poder filtrar repartos						X				
Como admin debo poder filtrar pedidos						X				
Como admin debo poder administrar direcciones de destinatarios								X		
Como admin debo poder filtrar las correcciones de direcciones								X		
Como admin debo poder borrar y recuperar un reparto								X		
El reparto debe crearse con las direcciones de destinatarios arregladas								X		
Como admin debo poder recalcular un reparto cuando corregí las direcciones de los destinatarios								X		
Milestone 3: Viaje del repartidor										
Como repartidor debo poder iniciar el próximo reparto										
Como repartidor en un reparto debo poder ver la próxima dirección en mi pedido										
Como repartidor debo poder navegar hacia la próxima dirección										

Ilustración 111 - Release plan en Google Sheets

A continuación se presenta la planificación de las iteraciones en formato tabla.

La columna tipo corresponde a:

- RF - Requerimiento funcional
- RNF - Requerimiento no funcional
- B - Bug

La columna Puntos indica la estimación de esa historia.

La columna *Sprint* indica el *Sprint* en la que la historia fue implementada.

Puntos	Tipo	Título	Sprint
Milestone 1 - Administración de repartidores			
3	-	Setup repositorio Ruby	1
3	-	Setup repositorio Android	1
3	-	Setup Heroku	1
3	-	Setup Google Play Store o equivalente	1
3	RF	Ícono de la app y pantalla de inicio	2
1	RF	Como admin debo poder loguearme a la web con usuario y contraseña	2
2	RF	Como admin debo poder crear usuarios de la app para los repartidores	2
1	RF	Como admin debo poder filtrar los repartidores	2
1	RF	Como admin debo poder ver repartidores	2
3	RF	Como repartidor debo poder loguearme	2
2	RF	Como repartido debo poder cerrar sesión	2
Milestone 2: Administración de pedidos y repartos			
8	RF	Como admin debo poder crear un reparto	3
1	RF	Como admin debo poder ver la lista de repartos	3
1	RF	Como admin debo poder ver los datos de un reparto	3
1	RF	Como admin debo poder ver los datos de un pedido	3
1	RF	Como admin debo poder ver la lista de pedidos	3
1	RF	Como admin debo poder filtrar repartos	3
1	RF	Como admin debo poder filtrar pedidos	4
5	RF	Como admin debo poder administrar direcciones de destinatarios	4

1	RF	Como admin debo poder filtrar las correcciones de direcciones	4
3	RF	Como admin debo poder borrar y recuperar un reparto	4
3	RNF	El reparto debe crearse con las direcciones de destinatarios arregladas	4
2	RF	Como admin debo poder recalcular un reparto cuando corregí las direcciones de los destinatarios	4
Milestone 3: Viaje del repartidor			
8	RF	Como repartidor debo poder iniciar el próximo reparto	5
3	RF	Como repartidor en un reparto debo poder ver la próxima dirección en mi pedido	5
5	RF	Como repartidor debo poder navegar hacia la próxima dirección	5
0.5	B	Si se hace manualmente cierre de sesión desde la consola a un repartidor debería ir a la página de inicio de sesión	5
21	RF	Spike de investigación del problema del enrutamiento de pedidos	6
5	RF	Como repartidor debo poder marcar un pedido como rebotado	7
2	RF	Como admin debo poder ver los pedidos rebotados	7
5	RF	Como repartidor debo poder marcar un pedido como entregado	7
2	RF	Como admin debo poder ver los pedidos entregados	7
1	RF	Como repartidor debo poder finalizar mi reparto automáticamente si terminé el último pedido	7
2	RF	Como repartidor debo poder finalizar mi reparto automáticamente si terminé el último pedido	7
0.5	B	Restaurar reparto no restaura pedidos	7
0.5	B	TRANSLATION MISSING: ES.YES en repartidores en la página de detalles de un reparto	7
0.5	B	Ir a detalles de un pedido borrado se rompe	7

0.5	B	Si se ingresa una url inválida en el admin explota	7
13	RNF	Como repartidor mi app debe funcionar aunque no tenga conexión, y debe actualizarse apenas la consiga	8 y carry-over al 9
13	RNF	El sistema debe ordenar los paquetes de un reparto de forma de tener el camino más rápido	y carry-over al 9
1	RNF	El enrutamiento automático de órdenes debe poder desactivarse para todos los futuros repartos desde una variable de entorno	8
Milestone 4: Monitoreo de repartos			
3	B	Se rompe la app en los repartos de más de 110 pedidos	10
1	B	Pedía rechazar el mismo reparto muchas veces	10
0.5	B	Foto de rechazado no es guardada	10
3	RF	Como admin debo poder visualizar los pedidos rebotados con alertas	10
3	RF	Como admin puedo ver el tiempo total de un reparto	10
2	RF	Como admin debo poder hacer que un pedido requiera número de precinto al entregarse	10
8	RF	Como admin debo poder visualizar la ubicación del repartidor en el momento en que marcó un pedido como rebotado	10
2	RF	Como admin debo poder resolver una alerta	11
2	RF	El sistema debe notificar a otro sistema externo cuando un pedido es rebotado o entregado	11
5	RF	Como admin debo poder visualizar rápidamente la última ubicación conocida del repartidor	11
8	RF	Como repartidor debo poder postergar un pedido en un reparto	11
3	RF	Como repartidor debo poder buscar un pedido y completarlo fuera de orden	11
1	RNF	Restringir resultados de geolocalizaciones a Uruguay	11

1	RF	Como admin debo poder visualizar cuántos pedidos tiene un reparto, agrupados por estado	11
1	RF	Como admin debo poder filtrar la lista de pedidos de un reparto para ver los pendientes de geolocalizar más fácilmente	11
1	RF	Mostrar link a “editar dirección de destinatario” en listado de pedidos del reparto	11
Milestone 5: Funcionalidades complementarias			
0.5	B	Marcar un pedido en el futuro después de las 9pm lo cuenta como otro día.	12
2	B	Si el último pedido se marca en modo avión nunca se sincroniza	12
0.5	B	En la web, en la página de un pedido, el reparto está en blanco	12
0.5	B	En el resumen de pedidos de un reparto, los filtrados no suman el total	12
3	B	Al finalizar un reparto la app deja de funcionar	12
1	B	Pedía rechazar el mismo reparto muchas veces (reapareció)	12
3	RF	Como admin debo poder eliminar y recuperar repartidores	12
5	RF	Como repartidor debo poder cambiar mi contraseña	12
3	RF	Como repartidor debo poder deslizar derecha o izquierda para postergar un pedido	12
1	RF	Como nuevo repartidor estoy obligado a cambiar mi contraseña la primera vez que entro	12
1	RF	Como admin debo poder cambiarle la contraseña a un repartidor desde la web	12
2	RF	Como admin debo recibir instrucciones para resetear mi contraseña si me la olvidé	12
2	RF	Como admin debo poder visualizar un resumen de los datos (Dashboard)	12
8	RNF	Integración con Routeme	13
0.5	B	No se puede editar las direcciones de destinatario de un reparto base	13

2	B	No aparecen los *markers* de la lista de repartos	13
5	RF	El admin debe poder seleccionar la ubicación de origen y destino de un reparto	13
3	RNF	El sistema debe recordar el enrutamiento de un reparto para no tener que volver a calcularlo en un reparto igual en el futuro	13
5	RF	Como admin debo poder visualizar el recorrido planificado para mi reparto automáticamente ruteado	14
3	RNF	El sistema debe reintentar automáticamente enviar para enrutamiento automático si pasaron más de 6 horas sin respuesta	14
2	RF	Como admin debo poder elegir si un reparto se enruta automáticamente o no	14
3	B	El segundo mapa en la pestaña de mapas no se muestra	14
0.5	B	No funciona la duración hasta ahora de un reparto si no ha terminado	14
0.5	B	Repartidor queda trancado si se borra una orden durante el reparto	14
1	B	A veces no procesa pedidos al menos que se abra y se cierre la aplicación	14
8	RF	Como admin debo poder monitorear el recorrido de un reparto en tiempo real	14
1	RF	Agregar link de acciones sobre el pedido en la lista de pedidos de un reparto	15
1	RF	Como admin debo poder ver la cantidad de pedidos en cada estado dentro de la página de detalles de un reparto	15
1	RF	Solo pedir foto de confirmación si el motivo es 'nadie en casa'	15
0.5	RF	Mejorar la calidad de las fotos de confirmación	15
3	RF	Poner un globito rojo con la cantidad en la pestaña de postergados	15
1	RF	Agregar link de editar/borrar/recalcular/restaurar reparto desde la página de detalles	15
0.5	B	Si posterga hacia la derecha la aplicación deja de	15

		funcionar	
0.5	B	Si dos repartidores abren el mismo reparto a la vez, no se sincroniza	15
0.5	B	Si se elimina un pedido en un reparto en marcha, no se sincronizan los pedidos a partir de él	15
0.5	B	En los mapas los marcadores no se centran automáticamente	15
0.5	B	Crear repartidor con cédula de un repartidor que se eliminó genera un 500 en el servidor	15
0.5	B	*Infowindow* en los mapas debería hacer auto *resize*	15

Tabla 50 - Plan de iteraciones

ANEXO 31 – “Escalas para el análisis de riesgos”

El equipo categorizó los riesgos en función a la franja a donde pertenezca su magnitud, calculada como el producto entre su impacto y su probabilidad de ocurrencia.

A continuación se visualizan las escalas utilizadas para la definición de impacto y probabilidad de ocurrencia, así como la delimitación de franjas de los valores de magnitud.

Escala de probabilidad de ocurrencia	
0	No probable
0,2	Poco probable
0,4	Probable
0,6	Muy probable
0,8	Altamente Probable
1,0	Se convierte en problema

Tabla 51 - Escala de probabilidad

Escala de impacto	
1	Bajo impacto con poco efecto sobre el proyecto y no debería de afectar la línea temporal del proyecto.
2	Impacto bajo medio, debería retrasar el proyecto por no más de un par de horas.
3	Impacto medio, debería afectar el proyecto por uno o dos días.
4	Impacto medio alto, debería afectar el proyecto por una semana o más.
5	Impacto alto, debería llevar a un atraso de semana o cancelación.

Tabla 52 - Escala de impacto

Escala de magnitud

De 0.1 a 0.6	Magnitud baja
De 0.7 a 1.5	Magnitud media
De 1.6 a 4.5	Magnitud alta

Tabla 53 - Escala de magnitud

ANEXO 32 – “Listado de riesgos y planes”

Código	Nombre	Estrategia de respuesta	Alerta	Plan de respuesta	Plan de contingencia
R1	Escasez de tiempo para dedicarle la atención necesaria al proyecto	Mitigar	No se está llegando a las mínimas horas semanales propuestas (15 hrs)	Organización del calendario del año y división de tareas en el proyecto	Aumentar la cantidad de horas trabajadas el fin de semana. Abandonar materias u otras actividades.
R2	Mala estimación de la velocidad del equipo	Mitigar	Se tiene carry - over dos *Sprint*s seguidos	Comenzar planeando menos puntos de los que se cree que se pueden hacer. Ir incrementando la velocidad a medida que es posible.	Aumentar horas trabajadas por semana. Reconstrucción del Release Plan para utilizar los *Sprint*s finales planificados vacíos como margen. Simplificación de los requerimientos. Recorte del alcance final.
R3	Demorar más de lo estimado por falta de dominio en desarrollo Android	Evitar	Las tareas de Android toman más del esfuerzo estimado	Realización de guías e investigación en las tecnologías a utilizar. Estimar con un margen. Dejar los primeros *Sprint*s con menos puntos	Hacer un Spike para mejorar en el dominio y dedicar más tiempo
R4	Mala estimación de una o varias historias	Mitigar	Se tiene caary-over en un *Sprint* sin que hayan disminuido las horas por semana	Dejar un período de dos *Sprint*s para recuperar al final. Estimar con margen. Realizar investigaciones antes de comenzar. Utilizar tiempo libre de un *Sprint* para investigar sobre el próximo.	No inclusión de la historia en el Release para no atrasar las pruebas de validación. Utilizar los periodos de arreglo de bugs para terminar las funcionalidades mal estimadas o disminuir el tamaño del scope.
R5	Dificultad de implementación del problema del enrutamiento	Mitigar	No se encuentra una solución adecuada o toma más tiempo de lo planificado	Hacer un Spike donde se haga un análisis previo del problema y se estimen y valoren distintas opciones.	Reestimar la historia y volver a planificar las iteraciones. Considerar solución alternativa. No realizar la historia si implica dejar otras funcionalidades críticas fuera del alcance final.
R6	Inhabilitación por más de una semana de un integrante del equipo por enfermedad	Mitigar	(No es posible definir alertas para accidentes)	(No es posible prevenir accidentes)	Recuperar las horas perdidas en *Sprint*s siguientes. Reconstrucción del Release Plan para utilizar los *Sprint*s finales planificados vacíos como

					margen. Simplificación de los requerimientos. Recorte del alcance final.
R7	Choque de roles	Evitar	Las tareas no parecen completarse por no identificar a quien les corresponde	No separar los roles pero definir a un líder que se encargue de coordinar las actividades del rol y asegurarse que se realicen. Avisarle al otro integrante antes de comenzar a trabajar en una tarea. Planificar división de tareas semana a semana en lugar de fijarla desde el principio del proyecto.	Aprender del error y asegurarse de que la tarea se realice.
R8	Cambios grandes en los requerimientos ya implementados	Evitar	El cliente no parece convencido del requerimiento a implementar durante la *Sprint* Planning	Validación previa con usuarios usando mockups para minimizar la mala comunicación o diseño incorrecto.	No aceptar cambios mayores en los requerimientos ya construidos a menos que esté justificado. Registrar el retrabajo y aprender del error.
R9	Conflictos entre los integrantes del equipo	Evitar	Discusiones continuas sobre el trabajo en el proyecto	Establecer y seguir un reglamento interno de comunicación y trabajo en grupo	Realizar retrospectivas para evaluar un modo de mejorar la forma de comunicarse y trabajar en equipo.
R10	Dificultad de integración de tecnología para navegación	Evitar	Demora de dos días en la *card*	Contar con la documentación e investigar previamente. Estimar estas historias con un margen para cubrirnos en estos casos.	Reestimar la historia y volver a planificar las iteraciones. Considerar solución alternativa. No realizar la historia si implica dejar otras funcionalidades críticas fuera del alcance final.
R11	Pérdida de los archivos de elaboración del proyecto	Evitar	No se respaldaron archivos	Utilización de repositorios para respaldo de archivos y versionado	Restaurar una versión anterior
R12	Cancelación del proyecto por parte del cliente	Evitar	Continuo disgusto del proyecto parte del cliente. Cliente no aparece en las reuniones. Cliente no contesta.	Evaluar continuamente la satisfacción del cliente. Hacerle firmar carta de compromiso con el equipo. Explicarle el impacto que tendría su abandono en el proyecto.	Plantear situación al tutor y en la Software Factory para definir cómo proceder. Si es al principio del proyecto existe la posibilidad de cambiar de cliente. Si es al final, el proyecto quizás puede continuar.

R13	Repartidores no se acostumbran a utilizar el software	Evitar	Los repartidores cometen muchos errores al realizar el reparto con la aplicación. Notas menores a 3 en encuestas de satisfacción a repartidores.	Delegar diseños a una diseñadora. Utilizar componentes nativos. Realizar análisis heurístico. Realizar *tests* de usabilidad con repartidores utilizando los prototipos. Realizar encuestas y actuar sobre la retroalimentación.	Realizar capacitaciones. Replantear el diseño y la arquitectura.
R14	Cliente insatisfecho con las rutas planificadas por Routeme	Evitar	Las rutas generadas no son más rápidas que las rutas base de la empresa	Probar rutas de repartos de producción antes de decidir integrar con el servicio	Pedir soporte a Routeme. Volver al código de la versión inicial del algoritmo propio de enrutamiento.
R15	Routeme decide interrumpir sus servicios	Mitigar	Servicios dejan de funcionar	Definir contrato con la empresa	Volver al código de la versión inicial del algoritmo propio de enrutamiento. Pedirle a Routeme su algoritmo e instalarlo localmente.
R17	Se reportan defectos durante repartos de prueba que el equipo no puede reproducir localmente	Evitar	Errores durante las pruebas para los cuales no se identifica la causa	Utilizar el mismo modelo de dispositivo que los repartidores durante el desarrollo. Instalar herramientas de Crashlytics en la aplicación y herramientas de visualización de logs en el servidor	Salir a probar el software a la calle internamente en reparto con los mismos pedidos que el reparto que falló.

Tabla 54 - Listado de riesgos y sus planes

ANEXO 33 – “Icebox”

A continuación se encuentra la lista de requerimientos funcionales y no funcionales que surgieron durante las reuniones pero que se decidió no implementar.

User Stories
Como admin debo poder cargar más de un archivo de pedidos por reparto
Mostrar un mensaje motivante cuando se finaliza el reparto
Emitir un pequeño sonido agradable cuando se entrega una orden
Como repartidor debo poder sugerir ediciones a la ruta planificada
Como admin debo poder cambiar el orden de los pedidos de un reparto explicando por qué
Como admin debo poder comparar tiempo de reparto de dos repartos con diferente ruta
Como repartidor debo poder ver que hay un nuevo reparto para iniciar sin tener que reabrir la app
Como admin debo poder cambiar el repartidor asignado de un reparto en progreso
Como admin debo poder agregar notas al resolver una alerta
Como administrador debo poder visualizar desvíos importantes de la ruta planificada con alertas
El sistema debe obtener los pedidos de los repartos a realizar automáticamente de otro sistema
Como admin debo poder quitar un pedido de un reparto en marcha
Como repartidor debo recibir una notificación cuando el admin quita un pedido de un reparto que ya he comenzado
Como admin debo poder elegir si el reparto precisa firma de confirmación o no
Como admin debo poder agregar un pedido a un reparto, especificando dónde se debe recoger
Como repartidor debo recibir una notificación cuando el admin agrega un pedido a un reparto que ya he comenzado
El sistema debe recomendar un repartidor calculando a quién le es más fácil volver para buscarlo y la zona a la que reparte cuando estoy agregando un pedido
Como admin debo poder visualizar que el destinatario de un pedido del reparto que estoy creando ha sido reordenado en un reparto previo

Como admin debo poder marcar un pedido como 'retiro en depósito' para que no sea necesario que el repartidor vaya a la dirección
Botón para volver al tope de la lista de pedidos rápido

Tabla 55 - Icebox