INFORME FINAL
Proyecto Final de Carrera

Rush - Sistema de reserva de asientos en colectivos

Alumnos:

Fleitas, Tomás Andrés
Hauque, Federico Gabriel
Re, Luis Santiago

Director: Mg. Ing. Lisandro Vrancken

Fecha de entrega: 23/10/2020

Índice

1. Introducción

El propósito central de este proyecto se enfoca en reducir considerablemente las colas de pasajeros que se forman para abordar los colectivos regulares que transitan entre las ciudades de Santa Fe y Paraná. La propuesta consistió en el desarrollo de una aplicación que permite reservar previamente asientos en los colectivos, de manera que los usuarios puedan asegurarse un lugar en su viaje sin esperar largas horas en una cola.

Se busca de esta manera reducir los tiempos de espera de las colas para abordar estos viajes, mejorando la calidad de un servicio que presenta numerosas dificultades tanto para los pasajeros como las empresas. Esperar en una cola no solo conlleva una considerable pérdida de tiempo, sino también priva a los usuarios de la posibilidad de organizar sus viajes con certeza. Además, la aglomeración de personas originada por las colas hace impensada la realización de viajes de estas características en el contexto de una pandemia como la que, al momento de redactar estas palabras, sigue generando dificultades en todo el mundo.

Específicamente, el desafío central del sistema a desarrollar, es la imposibilidad de interactuar con el mecanismo de pago del boleto. En efecto, la mayoría de los sistemas utilizados para la reserva de pasajes incluyen dentro de sus funcionalidades la posibilidad de abonar el pasaje. En el caso que nos ocupa, la existencia de sistema SUBE nos obliga a pensar una solución diferente para poder formular este sistema de reservas.

a) Planteo del problema

Las dificultades de transporte físico de personas y de bienes entre nuestra ciudad y el resto del país son de público conocimiento, pero pocos casos de rutas específicas han adquirido tanta repercusión en los medios de comunicación [1][2] y redes sociales [3] cómo son los trastornos ocasionados por las demoras que sufren los pasajeros que viajan entre Santa Fe y Paraná. Las largas colas que se forman en ciertas ocasiones llegan a ocupar cientos de metros de largo en las estaciones y obligan a aquellos que tienen compromisos horarios en su destino a anticipar el inicio de su viaje, debiendo arriba a la Terminal de Ómnibus mucho tiempo antes de su horario de partida ideal.

Intuitivamente, los usuarios reclaman mayor cantidad de colectivos disponibles en los horarios pico. Sin embargo, la estructura de subsidios para los precios del transporte en la República Argentina imposibilita incorporar más servicios si el Estado no aumenta significativamente los montos de subsidios. Obvio es señalar que los presupuestos públicos en el país están sujetos a tantas tensiones que no permiten pensar en que esa posibilidad sea viable.

Si bien son numerosas las oportunidades en que notamos largas colas de personas para abordar los colectivos entre Santa Fe y Paraná, en ambas Terminales, rara vez se observa el mismo fenómeno para los colectivos que parten hacia otras ciudades. En esos casos, cada pasajero tiene su asiento reservado con antelación y sabe que, en el tiempo pactado, podrá comenzar su viaje con poca o ninguna demora. Si existiese mucha demanda en un horario

determinado, los viajeros saben de forma anticipada cuáles colectivos están llenos, y pueden evaluar qué otros horarios podrían convenirles.

Tal como vemos en otros servicios similares, no hay motivos para que se produzcan colas cuando se elimina la incertidumbre sobre la demanda que habrá en cada horario. Muchos pasajeros que tienen compromisos o viajes en conexión en la terminal de destino necesitan tener una certeza de cuándo lograrán finalizar su viaje. Si no tienen esa seguridad, se ven obligados a arribar a la terminal de origen con varias horas de anticipación a la hora de viaje deseado. Como resultado, las colas terminan estando constituidas no sólo por aquellos pasajeros que planean viajar inmediatamente, sino también los que tienen algún compromiso varias horas después pero no quieren arriesgarse a que la demora comprometa su horario de llegada previsto. El temor a encontrar largas colas, produce un círculo "vicioso" en el comportamiento de los usuarios, que alarga aún más esas colas, multiplicando los costos de espera de los pasajeros.

Aun asumiendo el obstáculo que la limitante del presupuesto público nos impone, no existen impedimentos para eliminar la incertidumbre de los usuarios, permitiéndoles reservar un pasaje con antelación y asegurándoles que podrán viajar en un horario específico, de manera que no tengan que ir a la terminal tanto tiempo antes.

La propuesta que hacemos no pretende solucionar la escasez de servicios para las horas pico, pues esto requeriría la asignación de ingentes recursos. Nuestro proyecto está dirigido específicamente a reducir el tiempo de espera para el uso de los colectivos escasos, pudiendo destinar el tiempo que antes malgastaban haciendo cola en tareas productivas para ellos y para nuestro país.

El problema que enfrentamos en particular implica un desafío especial. Será necesario generar un sistema de reservas "independiente" del sistema de pago de la tarifa del servicio. En efecto, dado que el sistema SUBE, la forma más conveniente y barata de abonar el servicio, no permite la interacción con aplicaciones de terceros, no será posible el desarrollo de un único sistema que centralice el pago y la reserva del boleto, como tradicionalmente se observa en las aplicaciones relacionadas con la reserva de lugares para pasajeros en medios de transporte.

Este Proyecto Final de Carrera se basa en la construcción de una aplicación móvil (en adelante denominada "Rush") que permite reservar asientos en los colectivos que transitan a diario entre Santa Fe y Paraná, eliminando la incertidumbre sobre los viajes y atendiendo a una demanda de miles de usuarios que utilizan a diario este servicio. En relación al alcance del producto, se propuso establecer como funcionalidad principal la reserva del asiento, suponiendo que el costo del pasaje seguirá siendo más barato y conveniente si se abona por medio del sistema SUBE, y considerando que, como señalamos previamente, dicho sistema todavía no permite que aplicaciones de terceros debiten saldo de la tarjeta.

La posibilidad de saber por anticipado el horario en el que se podrá viajar reduce significativamente la necesidad de hacer colas, de manera que cada nuevo usuario de la

aplicación se convertiría en una persona que cuenta con un lugar ya reservado, reduciendo el tiempo de espera promedio del sistema.

Adicionalmente, la pandemia de Covid-19 declarada por la OMS durante la ejecución de este proyecto generó una preocupación generalizada por los ambientes en los que habitualmente se presentaban aglomeraciones de personas [4][5]. Como resultado, muchas actividades que pudiesen implicar un riesgo significativo para el contagio debieron suspenderse. Tanto el transporte de pasajeros aglomerados, como la espera en largas colas se tornaron negativos en la evaluación social de una actividad. En ese sentido, la puesta en funcionamiento de un sistema como "Rush", podría evitar la aglomeración de pasajeros en espera de viajar, impidiendo así la propagación de todo tipo de enfermedades infecciosas, de las que el Covid-19 es solo un ejemplo. De esta manera, un proyecto que fue originalmente pensado para reducir tiempos de espera, puede también resultar en una notable contribución a la salud pública.

b) Organización del Informe

El presente informe se estructura en el orden que a continuación se describe. En primer lugar, se introducen los conceptos y nociones básicas del problema a resolver, brindando un panorama completo del contexto en que se ha desarrollado este proyecto y el marco teórico en el que se sustenta. Se incluye allí un breve análisis de la prefactibilidad económica y social del proyecto en cuestión. Seguidamente, se procede a documentar los detalles del sistema que se propuso desarrollar para atender la situación problemática descripta. Como parte de dicha documentación se indican varios puntos, entre ellos, el proceso de desarrollo aplicado, la arquitectura del sistema y las funcionalidades del mismo, así como también las herramientas empleadas en el desarrollo y las incidencias que se presentaron a lo largo del proyecto. Se hace mención de los mecanismos de seguridad que se dispusieron para proteger al sistema, y las técnicas de testing utilizadas para garantizar la calidad del software.

Finalmente, se realiza un breve comentario acerca de la extensibilidad del proyecto y su potencial transformador en la sociedad, culminando con una conclusión que retome las actividades realizadas y las experiencias vivenciadas a lo largo del proyecto, para dar cierre a este informe.

2. Marco Teórico

a) Elementos conceptuales que enmarcan el problema

La problemática que este proyecto tiene como propósito mitigar no es de larga data. Hace poco más de un lustro, se permitía en el servicio bajo análisis que los pasajeros viajen parados en estos colectivos, abarrotando cada rincón posible de los pasillos del vehículo. Sin embargo, como estos servicios transitan por una autopista, viajar sin un asiento asignado y sin cinturón de seguridad, constituía una flagrante violación a las regulaciones de la Comisión Nacional de Regulación del Transporte (CNRT). Por tal motivo, las empresas debieron adecuarse a las normas prohibiendo el exceso de pasajeros, mejorando así su seguridad, pero dejando a muchos usuarios esperando en colas por largas horas, tal como se observa en la Figura 1.

Figura 1: Pasajeros esperando en la cola de la terminal de Paraná

Siendo que la problemática que se intenta resolver es, en esencia, una cola, no podemos obviar la existencia de estudios teóricos previos respecto a la dinámica de las mismas. La teoría de colas es un área temática que numerosos autores han investigado y de cuyos descubrimientos podemos nutrir nuestro análisis de la situación. Algunos trabajos como [6] y [7] introducen la temática estableciendo que las colas surgen por la naturaleza limitada de los recursos. Los autores explican que los mecanismos triviales para reducir el tiempo de espera en una cola son dos: disponer de más recursos o reducir los tiempos de servicio. Ninguna de las dos alternativas es viable bajo nuestras restricciones, pues reducir los tiempos de servicio implicaría violar los límites de velocidad de las rutas y disponer de más recursos no sería posible sin incursionar en una inversión impensable en el contexto actual.

Sin embargo, al igual que muchos otros estudios, la teoría de colas simplifica su análisis gracias a ciertos supuestos que, de no cumplirse, no sería posible sacar conclusiones triviales sobre la misma. Hillier y Lieberman en [8] mencionan que dos de los supuestos básicos de la

teoría de colas son “que ni el tamaño de la cola ni las costumbres de quienes esperan afecten la tasa de entradas”, es decir, que aun cuando las personas estén acostumbradas a que la cola sea larga, actúen de la misma manera que si fuese corta o mediana. Este comportamiento aleatorio está lejos de poder trasladarse a la espera de colectivos. Muchos usuarios tienen compromisos horarios en su destino, y si el servicio no les da ninguna garantía de arribar a tiempo, se ven obligados desarrollar algún mecanismo “defensivo” que les permita cumplir sus compromisos con un transporte deficiente. Es así que muchos pasajeros suelen llegar a las terminales con demasiado tiempo de anticipación, poblando las colas no sólo con personas que quieren viajar inmediatamente, sino también con viajeros que planean partir dentro de un par de horas. Incluso, en algunos casos, las redes sociales potencian aún más el inconveniente, haciendo públicas imágenes del tamaño de las colas e incitando a los usuarios que aún no han llegado a la terminal a apurarse, modificando así su comportamiento. Todos estos factores dejan en claro que el problema de colas a resolver no se ajusta a los supuestos de la teoría de colas, y por tal motivo, es posible que existan otras alternativas que permitan reducir los tiempos de espera sin afectar la cantidad de recursos ni los tiempos de servicio.

Uno de los factores clave que analizamos al momento de proponer esta solución es la previsibilidad de los usuarios. Si observamos el funcionamiento de cualquier otro viaje en colectivo de larga distancia, los tiempos de espera para abordar el vehículo son prácticamente despreciables. Esto se debe a que los pasajes se venden por anticipado, cada pasajero tiene un asiento reservado con antelación, y si su horario preferido no tiene más asientos disponibles, puede elegir otro horario y asegurarse que viajará en otro momento. En el transporte de pasajeros entre Santa Fe y Paraná no existe tal previsibilidad. Los usuarios se presentan en la terminal cuando desean, y si la capacidad no es suficiente, las colas crecen en consecuencia, originando todos los problemas que los restantes colectivos no presentan. Es así que nuestra propuesta consiste en replicar el sistema de reserva de asientos en estos colectivos por medio de una aplicación que brinde información sobre la disponibilidad de los vehículos y permita asegurarse un lugar en los viajes.

Al garantizarle a los usuarios el horario en el que van a viajar, hacer cola se vuelve totalmente innecesario. Cada pasajero debe presentarse en el horario de su reserva y abordar el colectivo. Nadie obtiene ninguna ventaja sobre otra persona por llegar antes a la terminal, sino que la seguridad de disponer de un asiento en el viaje puede conseguirse desde la comodidad de nuestro hogar.

Las ventajas del uso de esta aplicación no solo consisten en una considerable reducción de las colas, sino también la desaparición de todos los trastornos que las colas ocasionan: Tiempo ocioso en la terminal, largas esperas en condiciones climáticas adversas (lluvia, calor, frío excesivos), entre otras. Adicionalmente, los servicios de las empresas podrían mejorar gracias a funcionalidades adicionales que más tarde podrían incorporarse en la aplicación como: filtro de colectivos de acuerdo a servicios especiales (portaequipajes, accesibilidad, etc.), retroalimentación a la empresa sobre el estado de los colectivos y la calidad del servicio, entre otras.

No sólo los usuarios verían beneficiados sus intereses con el uso de esta aplicación. Las empresas también pueden ver provechoso mejorar su imagen al haber tomado cartas en el asunto para resolver la problemática, de la misma forma que podrían nutrirse de la información que recolecta la aplicación sobre sus clientes y sus hábitos de viaje. Asimismo, al recibir reservas por anticipado, serían capaces de estimar con mucha más precisión las fechas de mayor demanda y actuar en consecuencia de forma más expedita.

Si bien la teoría tradicional de colas no resultó adecuada para analizar esta problemática tan compleja, el no cumplimiento de uno de los supuestos abrió la posibilidad teórica que exista alguna solución diferente a las dos alternativas triviales antes mencionadas, proponiendo herramientas que determinen comportamientos sociales más convenientes para cada uno de los pasajeros y para su conjunto. El estudio en detalle del funcionamiento de otros colectivos similares que no presentan estos inconvenientes motivó el surgimiento de una solución creativa que facilitaría enormemente la vida de todos aquellos que día a día transitan entre ambas capitales para estudiar, trabajar y realizar todo tipo de actividades.

b) Análisis de prefactibilidad socio-económica e impactos

Una vez definida la propuesta de solución a la problemática planteada, corresponde realizar un análisis de prefactibilidad económica privada sobre la misma.

Según los datos recabados que se exponen en una tabla incluida más abajo, se venden anualmente, sin considerar los efectos actuales de la pandemia, casi 2.500.000 pasajes que solo 2 empresas ofrecen para transportar pasajeros entre Santa Fe y Paraná. Los estudiantes representan una proporción considerable de la totalidad de los usuarios, principalmente por la conveniencia de los boletos con descuento que, mensualizados, implican un costo mucho menor al de un alquiler de un inmueble cercano a la Facultad donde estudian. Los restantes usuarios se reparten entre diversos trabajadores, usuarios esporádicos y turistas que, en su mayoría, también se verían beneficiados por los efectos de una reducción significativa de las colas.

Estas colas se forman principalmente en las terminales de colectivos de Santa Fe y Paraná situadas respectivamente en Belgrano 2910 y Av. Ramírez 2350. Asimismo, se busca también reducir las colas que se forman en las distintas paradas intermedias de los viajes que recorren otros puntos de ambas ciudades.

Para el mismo consideramos el aprovechamiento económico del sistema a través de un emprendimiento unipersonal encuadrado en el Monotributo Nacional [9] y en el Convenio Multilateral para las Provincia de Santa Fe [10] y de Entre Ríos [11] además de la consideración de la Tasa de Seguridad e Higiene de la ciudad de Paraná [12]. Suponemos para este análisis que existe un vínculo contractual entre el administrador del sistema y las empresas de colectivos que prestan el servicio Santa Fe – Paraná.

I. Demanda e Ingresos

El mercado se divide aproximadamente en dos mitades, una de ellas servida por la empresa ERSA y la otra por la Empresa Etacer. La Tabla I muestra la utilización de servicio por pasajeros solo para la empresa ERSA por el año 2018 y los primeros meses de 2019.

Mes	Pasajeros
Enero	76364
Febrero	96285
Marzo	130833
Abril	136341
Mayo	129962
Junio	118034
Julio	88779
Agosto	110060
Septiembre	114416
Octubre	131123
Noviembre	103768
Diciembre	86562
Total Anual	1322527
Promedio Mensual	110211

Tabla I: Cantidad de pasajeros por mes de la empresa ERSA

Si suponemos un promedio aproximado mensual para cada empresa de 100.000 viajes, el total de viajes mensuales del mercado resultará de unos 200.000. Partiendo de un supuesto prudente que el servicio de reserva será utilizado solo en el 25% de los viajes, nos encontramos ante una cantidad a servir de 50.000 viajes/mes para la realización de las reservas.

Considerando un precio base del servicio de $ 5.- por reserva, los ingresos mensuales resultarían en unos 250.000 pesos. En la siguiente sección se hará un análisis de costos que determinará si este nivel de ingresos hace factible el negocio bajo los supuestos planteados.

Consideramos que si valorizamos la hora de espera del pasajero utilizando un concepto como el salario mínimo vital y móvil por hora que actualmente es $ 94.50 [13], un ahorro de solo 15 minutos de tiempo, arrojaría para la persona un resultado mayor a los $ 23.63, sin considerar otros beneficios particulares y sociales ligados a la incomodidad de la espera. Entendemos que, realizando esa comparación, el pago de $ 5.- es más que razonable.

II. Costos

Los costos fijos y variables más significativos ligados al emprendimiento se indican en la Tabla II y son los siguientes:

- El pago mensual de un Monotributo de aproximadamente 2500 de pesos.
- La comisión que carga la plataforma de pagos (MercadoPago) por cada cobro que realicemos a través de su plataforma. En caso de aceptar recibir el dinero a los 30 días del momento de la transacción se acerca al 2,5% más IVA del total cobrado. Dada la inexistencia de costos significativos erogables resulta razonable "esperar" financieramente la recepción del dinero dada la significativa reducción de la alícuota de comisión.
- El costo de los tributos provinciales y municipales variables sobre el monto de ventas estimado aproximadamente en un 5%
- En concepto de otros costos e imprevistos se estima un 15% del total de los costos antes identificados. En esto incluimos aquellos costos ocasionados por una elevada demanda de peticiones a la WebAPI o transacciones a la base de datos.

El análisis de costos realizado se resume en la Tabla II. Junto con la estimación de 250.000 pesos mensuales de ingreso, realizamos también un análisis de sensibilidad considerando la posibilidad de obtener ingresos un 25% mayores y un 25% menores de ingresos.

ANÁLISIS DE COSTOS			
	-25%	Estimación	+25%
Ventas	187,50	250,00	312,50
Costo variable tributario	-9,38	-12,50	-15,63
Costo Variable MercadoPago	-5,67	-7,56	-9,45
Costo Monotributo (**)	-2,50	-2,50	-2,50
Otros costos Imprevistos	-2,63	-3,38	-4,14
Total ganancia	167,32	224,05	280,79
Porcentaje sobre ingresos	89,24%	89,62%	89,85%

Tabla II: Análisis de costos del proyecto

(*) Datos indicados en miles de pesos.
(**) En las 3 estimaciones corresponde la categoría de monotributo B y por eso las 3 tienen el mismo valor.

III. Resumen de los impactos

El resultado neto relativo que surge de la comparación de los ingresos y los costos antes detallados es significativo y, al mismo tiempo, permite brindar un importante servicio que la sociedad demanda hace años. Asimismo, el emprendimiento sería capaz de hacer frente a potenciales costos administrativos que se requieran abonar a las empresas para asegurar la actualización de los datos relativos a viajes y tarifas.

Recordemos que las reservas se realizan para un día y horario específico y se garantizan con una tarjeta de crédito, de donde se cobra una multa solamente en el caso que el pasajero no se presente, montos que estarán destinados para ser transferidos a las empresas y no resultan ingresos propios.

Pensamos en una implementación paulatina de este sistema en la que, en primera instancia, se reserven unos pocos asientos, para luego incrementar la proporción de asientos sujetos a reserva y, eventualmente, destinar algunos colectivos enteros a clientes con reservas. Es probable que, si coexiste el sistema de reservas con la posibilidad de subirse libremente permanezcan las colas, pero ya no ocuparán cientos de metros de distancia, sino que serán un puñado de usuarios infrecuentes que esperarán algún asiento libre en el que ellos también puedan disfrutar su viaje.

a. Impacto en los Clientes

Los clientes serían los mayores beneficiarios de las ventajas provistas por la aplicación, pudiendo estos planificar sus viajes con mayores certezas desperdiciando menos tiempo en colas innecesarias. También, en el largo plazo, se podrían proveer algunos servicios adicionales como la posibilidad de abordar el ómnibus en diferentes paradas, informar sobre eventualidades como paros o desperfectos técnicos y asegurar reservas en colectivos que posean portaequipaje, aire acondicionado u otros servicios. Como resultado, se espera que el incremento en los servicios prestados, la reducción de las colas y la mayor comodidad de los usuarios, mejore significativamente la satisfacción de los clientes.

b. Impacto en los Choferes

Los choferes ocupan un rol crucial en el funcionamiento de esta aplicación, pues son ellos quienes registran el ingreso de pasajeros y validan sus reservas. Además, el uso del sistema permitiría controlar cuántos asientos se van ocupando, sin que el chofer necesite pararse para contar la cantidad de lugares libres. Esta comodidad se suma a las mejores condiciones de trabajo que gozarán los choferes al ver las colas reducidas en tamaño, lo que tornaría más infrecuentes los altercados con usuarios molestos por las largas esperas.

c. Impacto para la Empresa

La presencia de extensas colas en las terminales no solo es visible para los usuarios del servicio, sino también para los cientos de personas que transitan a diario ambas terminales. Reducir las colas no solamente mejorará la imagen de la empresa con sus clientes actuales, sino también con todos aquellos que hoy observan los cientos de metros de cola de usuarios esperando abordar sus colectivos. Muchos de los usuarios frustrados con el servicio podrían

volver a considerar viajar con esta alternativa que tiene la ventaja de ser mucho más económica en comparación con el automóvil (el pasaje vale casi la mitad del peaje sin considerar gasto de nafta) sin la desventaja de la incertidumbre que originan las colas.

Otra ventaja muy importante de la implementación de este proyecto es la mayor disponibilidad de información sobre los clientes y sus hábitos de viaje. De esta manera, la empresa podría anticiparse a la demanda y conocer los colectivos que se llenarán con más antelación, así como también adelantar el devengamiento de los pasajes reservados que ya implican un compromiso del usuario a viajar en el futuro.

Gran parte de la operatoria de la empresa se podría gestionar por medio del sistema que integraría la reserva de pasajes, la planificación de los colectivos y su capacidad, así como también la venta de pasajes por ventanilla que también podría hacerse en forma de reserva.

Finalmente, todo este desarrollo que realizamos propone una solución a la problemática de las colas en los colectivos sin que las empresas tengan que incursionar en el desarrollo de software ni enfrentarse a los riesgos que semejante proyecto implicaría. Siendo que el desarrollo ya está finalizado, la empresa solo debe tener voluntad de implantar el sistema y adaptar sus procesos de negocio para que el mismo comience a funcionar.

d. Impacto sobre la salud pública

El contexto de pandemia actual ocasionado por el COVID 19 hace inconcebible una cola de semejante magnitud como las que habitualmente se presentaban en este sistema de transporte. La implementación de Rush implicaría una reducción significativa de aglomeraciones de personas en las terminales, desacelerando la propagación, no solo del Coronavirus, sino también de muchísimas otras enfermedades peligrosas.

Al momento de finalizar la redacción del presente las autoridades nacionales han habilitado, aún con fuertes limitaciones, la reanudación del servicio de transportes, tornando más urgente la definición de una solución para este problema.

IV. Perspectivas de escalabilidad futura en el mercado

La incorporación de Rush a la operatoria de transporte no debe necesariamente permanecer limitada a los servicios que conectan Santa Fe y Paraná. Otros colectivos similares de media distancia podrían aprovechar la aplicación como un servicio más para los pasajeros. Además, los colectivos de larga distancia también pueden implementar no solo la reserva, sino también la compra de pasajes por medio de aplicación (puesto que SUBE no está involucrada). Esto permitiría que los pasajeros puedan recibir notificaciones sobre sus próximos viajes y abordar los colectivos sin necesidad de imprimir ningún pasaje en papel. La comodidad que brindaría una aplicación de estas características en los servicios de larga distancia ya está disponible en empresas de colectivos de otros países del mundo y no pensamos que existan obstáculos significativos para implementarla en Argentina.

3. Funcionalidades y características del sistema

Toda la arquitectura del sistema interactúa entre sí para proveer las funcionalidades que se expondrán en esta sección. Si bien dichas funcionalidades solo se pueden apreciar visualmente en la aplicación móvil, no se puede desestimar el rol del resto de los módulos de la arquitectura que se desarrollan con más detalle en la sección 4b.

a) Inicio de sesión (Aplicación Android)

Para darle la bienvenida al usuario se dispuso una pantalla de inicio (Figura 2) con un único botón que permite iniciar sesión haciendo uso de una cuenta de Google. Siendo que la aplicación se ejecutará sobre el sistema Android, el cual requiere estar asociado a una cuenta de Google, solicitar dicha cuenta para iniciar sesión no supone ningún impedimento para que cualquier usuario acceda a la aplicación.

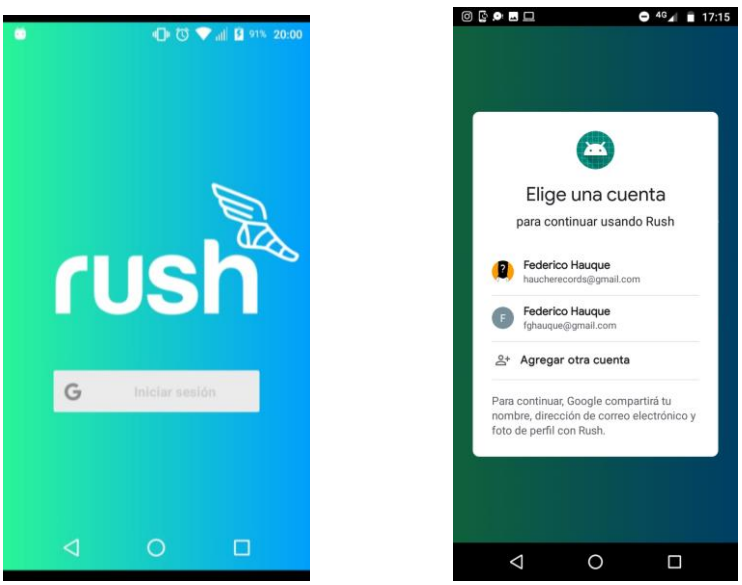


Figura 2: Inicio de sesión en la aplicación Android

Si disponemos de más de una cuenta en el servicio, se nos solicita indiquemos cual usaremos para acceder a la aplicación, y posteriormente de forma automática, somos redirigidos al menú principal de la aplicación.

b) Menú principal

El menú principal de la aplicación nos da la bienvenida una vez que fuimos identificados como usuarios del sistema (Figura 3). Cuatro grandes botones nos dan acceso a las funcionalidades principales del sistema: “Nueva Reserva”, “Mis Reservas”, “Historial”, “Mi billetera”.

Figura 3: Menú principal de la aplicación

Nueva Reserva: Permite buscar viajes, reservar y pagar por un asiento.

Mis Reservas: Muestra en pantalla un listado de las reservas realizadas para viajes futuros y permite observar detalles, cancelarlas y generar un código de validación para abordar el colectivo.

Historial: Ver un listado de las reservas pasadas o canceladas y consultar sus detalles.

Mi Billetera: Gestiona, da de alta y de baja las tarjetas de crédito asociadas a la plataforma de pago y permite elegir una tarjeta por defecto para pagar los viajes en la aplicación.

c) Mi Billetera

Si se desea gestionar la billetera de la aplicación, se muestra la siguiente pantalla en la que se identifica al usuario logueado (Figura 4) y se le permite ver el listado de las tarjetas cargadas en el sistema, así como también, agregar una tarjeta para los pagos posteriores.

Figura 4: Pantalla de “Mi Billetera”

Asumiendo que es la primera vez que iniciamos sesión y no tenemos ninguna tarjeta asociada, debemos dar de alta nuestro primer medio de pago. Para ello se solicitan los datos de la tarjeta, que deberán ser validados antes de poder usarse para cualquier pago (Figura 5).

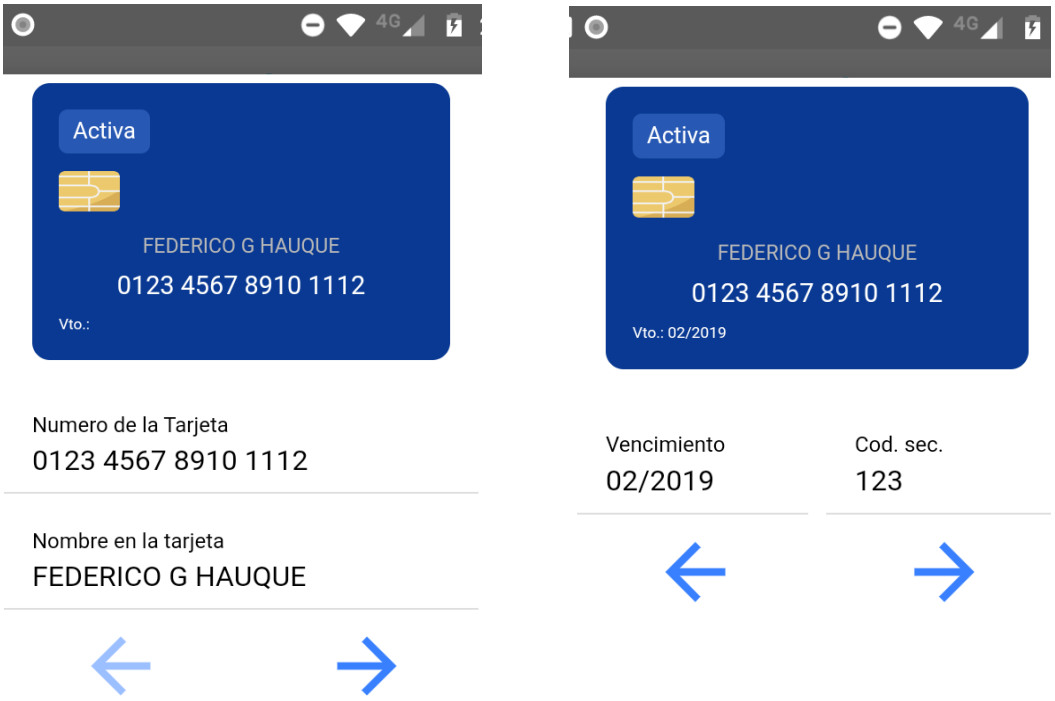


Figura 5: Carga de datos de una nueva tarjeta

Si la validación de la tarjeta es exitosa, ya podemos verla en el listado de “Mis tarjetas” que muestra todos los medios de pago asociados a nuestra cuenta de usuario. De todo el listado, una de las tarjetas debe estar marcada como activa, lo que significa que la misma será el medio de pago predeterminado para abonar reservas.

En cualquier momento, el usuario puede cambiar su tarjeta activa simplemente eligiéndola en el listado de “Mis tarjetas”.

d) Nueva Reserva

Para realizar una nueva reserva, se debe primero buscar el viaje que se pretende reservar (Figura 6). Para ello se debe indicar como parámetros de búsqueda la terminal de origen, la terminal de destino y la fecha de viaje prevista. Luego de presionar en “Buscar” se obtiene un listado de los viajes que pasan por ambas terminales en la fecha indicada. Se indica para cada viaje el horario y terminal de salida, así como también la terminal de destino y la hora prevista de arribo a la misma. Es posible observar los viajes que ya no tienen más asientos disponibles como parte de la lista, pero obviamente, el botón de “Reservar” no está disponible para estos viajes.

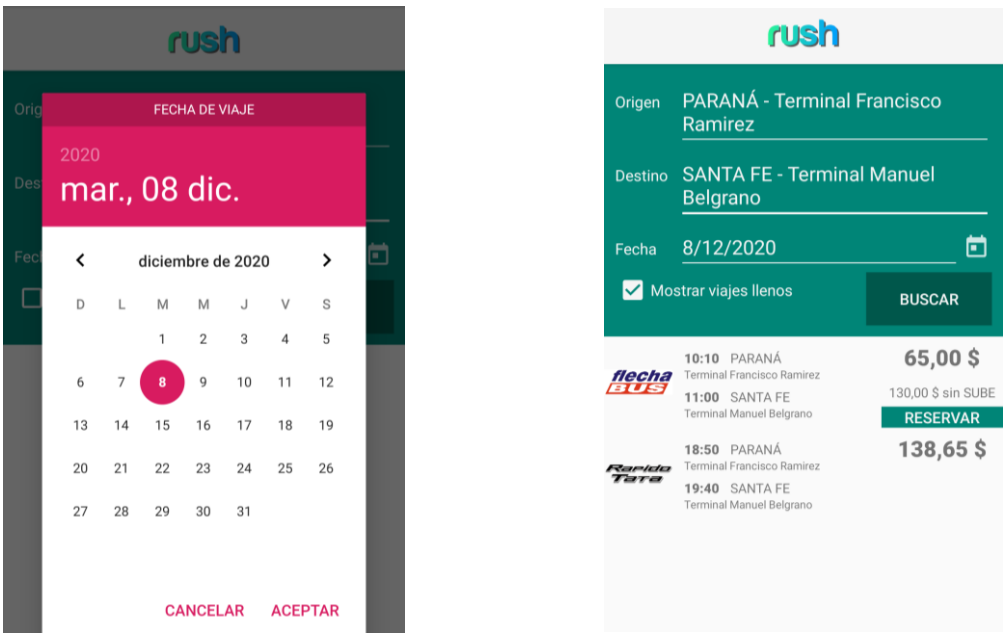


Figura 6: Búsqueda de viajes

Una vez que se decidió el horario a reservar, se procede a efectuar la reserva, para lo cual se solicita confirmación del pago con la tarjeta activa. De no existir ninguna tarjeta activa, se solicita que se elija una o se carguen los datos de una primera tarjeta. Si hay una tarjeta activa, se muestran en pantalla sus datos, acompañados de los importes de la autorización que se va a realizar (Figura 7). Se solicita el ingreso del código de seguridad de la tarjeta (CVV) para confirmar la autorización y finalmente efectuar la reserva.

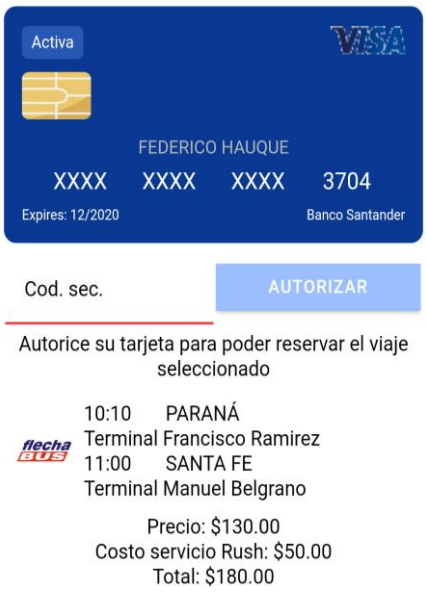


Figura 7: Confirmación del pago de la reserva

e) Mis Reservas

Una vez que se creó la reserva, la misma ya se puede encontrar en el listado de “Mis reservas”. En este listado solamente aparecen las reservas realizadas por el usuario logueado para viajes futuros que no hayan sido cancelados (Figura 8).

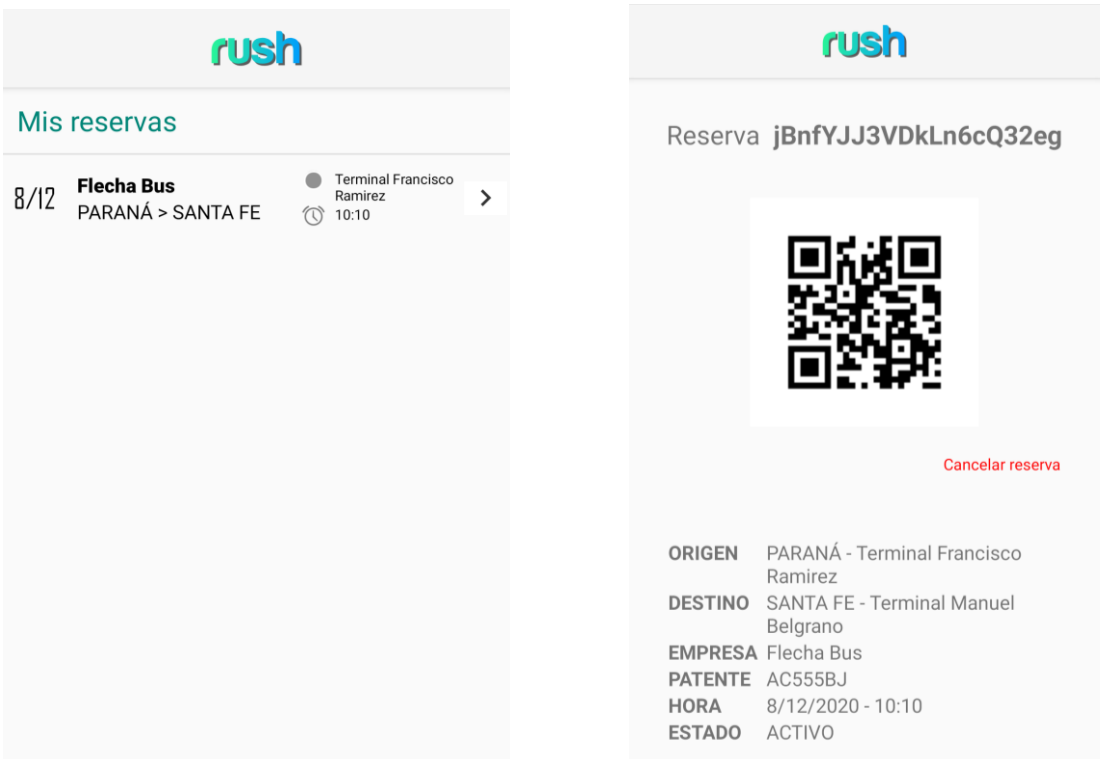


Figura 8: Listado de “Mis Reservas”

Si se selecciona una de las reservas del listado, se pueden observar los detalles de la misma, incluyendo un código QR que puede usarse para validar la reserva al momento de abordar el colectivo (Figura 8). Debajo del mismo se ubica la opción para cancelar la reserva.

Al momento de cancelar una reserva, se debe aplicar la política de cancelación de la empresa. La misma especifica si se debe cobrar un porcentaje del boleto o no dependiendo con cuánta antelación se está efectuando la reserva. Las reservas que se cancelan con mucha anticipación no originan ningún cargo en la tarjeta del pasajero. Ahora bien, las cancelaciones realizadas demasiado cerca de la fecha de partida pueden requerir el pago de un porcentaje del boleto. Dicho pago se realiza de forma automática junto con la cancelación.

Rush no posee una única política de cancelación global, sino que permite que cada empresa defina sus propias reglas para el cobro de cancelaciones. Para ello la base de datos almacena en el documento de cada empresa un campo para su política de cancelación. En el mismo se definen rangos de horas y porcentajes a cobrar en cada uno. Por ejemplo, se puede definir un 30% del boleto a cobrarse si la cancelación se realiza entre 48 y 24 hs. de la partida del viaje.

Para el rango de 72 a 48 hs. ese porcentaje debe ser menor, y así sucesivamente se definen rangos y porcentajes hasta llegar a un techo de horas por encima del cual nunca se cobrará ninguna cancelación.

f) Multado de Reservas

Siendo que Rush está pensado para colectivos que se pagan por medio del sistema SUBE, en un contexto normal, la empresa no cobrará nada hasta que el pasajero pase su tarjeta por el lector de SUBE dentro del colectivo. Si el pasajero no viaja, la empresa no cobraría nada, y habría un asiento vacío desaprovechado. Por este motivo, Rush ejecuta periódicamente una tarea que recopila la información de todas las reservas pasadas que no fueron usadas y les aplica una multa sobre el valor autorizado en la tarjeta (que es equivalente al precio del pasaje sin subsidios). Este mecanismo permite disuadir a los pasajeros de realizar reservas que no vayan a utilizar. Y si a pesar de esto, un usuario reserva un asiento que no usa, la empresa gana igual el mismo dinero que si el pasajero hubiera viajado.

El multado de reservas se ejecuta todos los días a las 8 de la mañana y, para evitar problemas de concurrencia, solamente procesa las reservas correspondientes a viajes que hayan finalizado 48 horas antes del momento de la ejecución. Si algún cobro falla por algún error de la plataforma de pagos, su estado en la base de datos no se modifica, y en la próxima ejecución de la tarea periódica, se intentará volver a procesarla.

g) Historial

Al presionar en esta opción, se observa un listado de las reservas históricas realizadas por el usuario logueado (Figura 9). Las reservas que fueron usadas en algún viaje se listan en color negro, mientras que las que fueron canceladas o multadas se listan en color rojo.

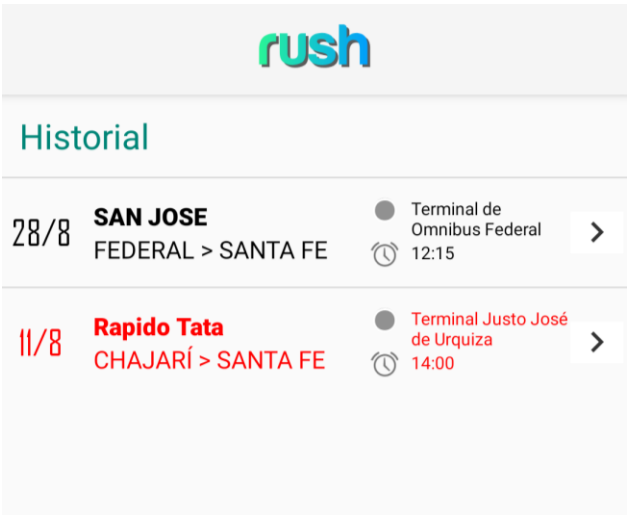


Figura 9: Pantalla del “Historial de reservas”

Al igual que en el listado de “Mis Reservas”, se puede seleccionar una reserva y ver sus detalles, pero por ser reservas pasadas, no está disponible la opción para cancelarlas.

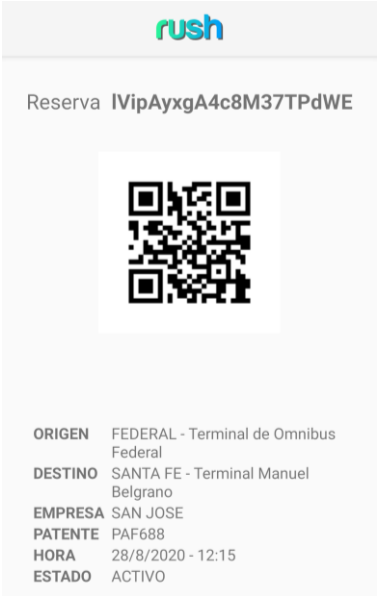


Figura 10: Detalles de reserva pasada

h) Inicio de sesión (Interfaz de administración)

Las empresas pueden gestionar sus viajes a través de una interfaz de administración. Para acceder a la misma, los empleados administrativos de cada empresa deben autenticarse en la misma con un email y una contraseña (Figura 11).

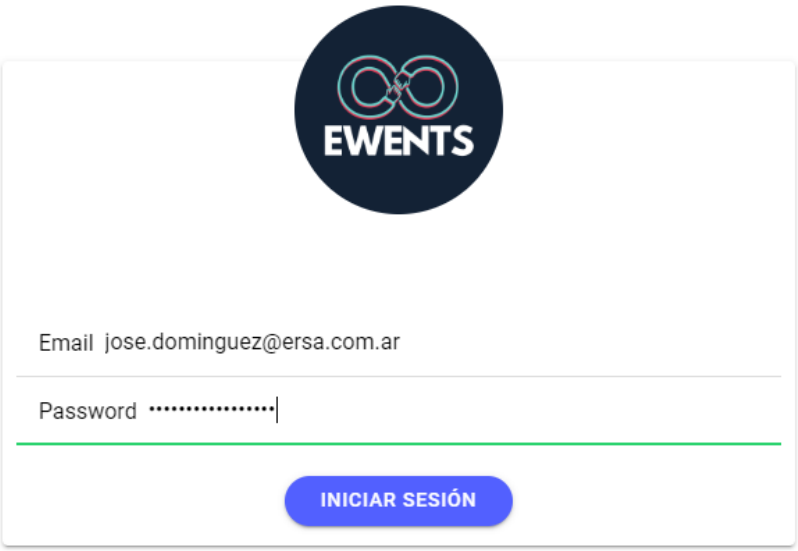


Figura 11: Inicio de sesión en la interfaz de administración

i) Listado de viajes

Los empleados de cada empresa pueden consultar la lista de viajes cargados para la misma (Figura 12). El botón flotante de la parte inferior derecha sirve para dar de alta un nuevo conjunto de viajes.

Figura 12: Listado de viajes de una empresa

I. Crear nuevo viaje

Para crear un nuevo viaje se comienza solicitando datos del colectivo (obligatorio) y chofer (opcional) que estarán asignados al mismo (Figura 13).

Figura 13: Asignación de chofer y colectivo

Luego se debe ingresar los datos de la partida del viaje, que son la terminal de origen junto con la fecha y hora de salida (Figura 14).

Figura 14: Carga de la terminal de origen

Antes de ingresar las siguientes paradas se pregunta si todos los trayectos tendrán o no el mismo precio y si es posible pagar con el sistema SUBE (Figura 15).

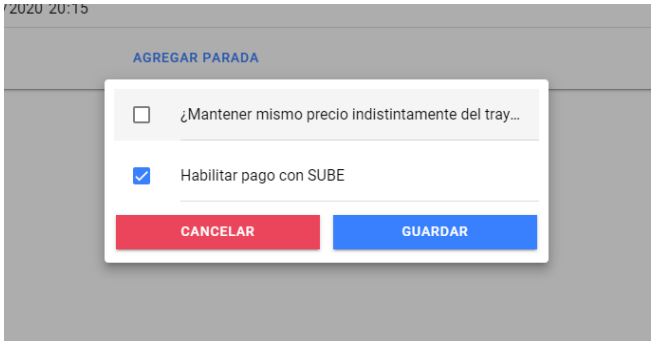


Figura 15: Configuración de precios

Al agregar las demás paradas, se solicitan los mismos datos de la terminal y hora de arribo, pero también se pide el ingreso del precio que tiene el trayecto desde la última parada cargada (Figura 16). Si el viaje se puede pagar con SUBE, se pide también este otro precio por separado.

Figura 16: Carga de trayectos

Finalmente, en la última pantalla se muestra toda la información del viaje a crear. Se muestra una foto del colectivo y chofer (si se eligió uno), los precios de todos los tramos y se puede marcar en un calendario todas las fechas en las que queremos que se repita este viaje (Figura 17). Una vez que estamos seguros de todos los datos, podemos presionar el botón verde de guardar y se crean todos los viajes en las fechas indicadas.

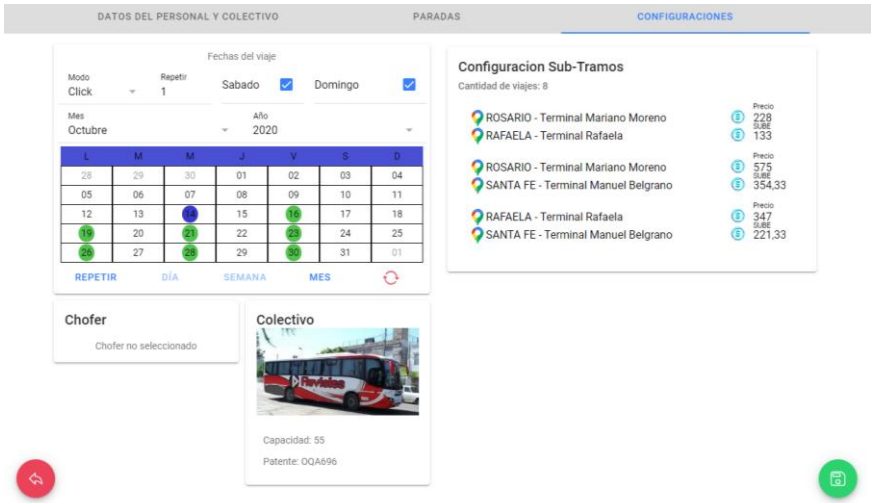


Figura 17: Carga de fechas del viaje

4. Documentación

a) Metodología de desarrollo

Para este proyecto, la intención fue aplicar una metodología de desarrollo que sea capaz de adaptarse cómodamente no sólo a la naturaleza de las actividades a realizar, sino también al equipo de trabajo. Algunos de los principios en los que se sustenta esta nueva metodología fueron tomados de otros procesos de desarrollo y combinados, de manera que la organización del proyecto posea las prácticas más aplicables de cada metodología conocida.

Si bien tanto Scrum como el Modelado ágil no constituyen procesos de desarrollo prescriptivos, ambos resultaron un buen punto de partida para la definición de un proceso de desarrollo que pensábamos que se ajustaría a las necesidades del equipo. Hicimos ciertas modificaciones a las propuestas de ambos procesos en pos de la comodidad de los integrantes, sin que esto fuese en detrimento de la calidad del producto. Sin embargo, la coyuntura del contexto en el que se llevó a cabo este proyecto nos obligó a realizar numerosos ajustes que entendemos que nos ayudaron a mantener un mínimo nivel de orden para continuar nuestras tareas de la forma más organizada posible.

Dada la disponibilidad horaria de los integrantes y la preferencia por las reuniones de seguimiento, nos pareció adecuado aplicar algunas de las características del proceso Scrum para el desarrollo de este proyecto. Sin embargo, dada la reducida cantidad de participantes, no tenía sentido la asignación de roles para las reuniones. No se eligió ningún scrum master, sino que todos los integrantes del grupo colaboraron en el análisis de la marcha del proyecto, consensuando las decisiones que afectarían el avance de las diferentes tareas antes de la próxima reunión.

El desarrollo se realizó mayormente de forma distribuida, con reuniones Scrum periódicas en las que se planteaban las dificultades y se intentaba resolverlas. Las actividades de testing se hicieron en 2 instancias. Al momento de desarrollar una nueva funcionalidad, el autor del código preparaba una batería de pruebas unitarias y se aseguraba de su correcto funcionamiento ejecutando casos de prueba al integrar dicha funcionalidad con las ya desarrolladas. El siguiente paso era subir el *pull request* con los cambios a GitHub[1], donde los demás integrantes podrían acceder al código, controlar si el alcance de la batería de pruebas unitarias era adecuado y ejecutar otros casos de prueba para verificar su correcta integración. Si algún integrante consideraba que se necesitaban cambios menores, se indicaba en un comentario. Si se necesitaban cambios considerables se planteaba su discusión en la siguiente reunión scrum. Por el contrario, si no había ninguna objeción al nuevo código propuesto, los integrantes lo aprobaban y se procedía a hacer un "merge" a la rama de desarrollo del proyecto.

[1] GitHub – https://github.com

El equipo siempre tuvo predilección por las reuniones scrum presenciales, puesto que en las mismas no sólo planteábamos las dificultades que teníamos en nuestras tareas, sino que también resolvíamos dichos inconvenientes programando ciertas partes grupalmente. Asimismo, en dichas reuniones, nos dedicábamos a observar el código de los pull requests de los demás integrantes, sabiendo que contábamos con su presencia para hacerles alguna consulta si no entendíamos alguna de las modificaciones que propusieron. Lamentablemente, a lo largo del proyecto se pudieron realizar sólo unas pocas reuniones presenciales durante los primeros sprints. Los 3 integrantes vivimos en ciudades diferentes, lo que, sumado al advenimiento de las medidas impuestas por el distanciamiento social propiciadas por la pandemia mundial antes mencionada, terminó haciendo inviable aplicar la metodología planteada inicialmente.

Como respuesta a la imposibilidad de reunirnos físicamente, el grupo debió reformular las reuniones scrum de manera virtual. Es así que se definió un día a la semana en el cual, haya o no haya avances para comentar, el grupo debía reunirse para definir los próximos pasos del proyecto. Sin embargo, estas reuniones fueron más cortas y nunca lograron tener la misma productividad que las reuniones presenciales. Cada integrante debió encontrar el momento para verificar los pull requests ajenos y diferir sus consultas al instante en que el autor pueda responderlas.

La estimación de los tiempos de desarrollo se hizo en horas reloj, resultando más cómodo que los puntos de historias de usuario, que muchas veces son confusos. Se definieron prioridades en las tareas, de manera que se pueda ordenar más claramente qué requerimientos son más urgentes y cuales deben estar listos antes de poder desarrollar otros.

A diferencia de la mayoría de las metodologías ágiles, este proceso utilizó modelado UML, pero se siguieron varios preceptos del "Modelado Ágil" propuestos por Ambler [14][15] y seguidos por Pressman [16]. En el panfleto introductorio del Modelado Ágil [17] se enuncian varios de los principios que mantendremos en este proyecto, entre los cuales nos sentimos identificados principalmente con los siguientes:

- *Modelar con un propósito:* No deben producirse modelos que representen aspectos del sistema cuya utilidad sea dudosa. Se debe modelar la estructura esencial del modelo, pero no hacer hincapié en detalles inconducentes.
- *Múltiples modelos:* Podría parecer irrisorio modelar lo mismo de formas diferentes. Sin embargo, es necesario reconocer que los modelos evolucionan a lo largo del proyecto y no podemos pretender lograr un sólo modelo único y perfecto en medio del proceso de desarrollo. En un primer momento se realizan modelos detallados, pero luego se realizan croquis sencillos en cada iteración, facilitando la evolución y difiriendo las tareas de documentación.
- *Documentación tardía:* Concentrar los esfuerzos de documentación una vez que el modelo ya evolucionó y el proyecto fue puesto en marcha.
- *Priorización de requerimientos*: Presente en muchos modelos de desarrollo, permite distribuir mejor la carga de trabajo, dándole mayor peso a las tareas que se enfocan en funcionalidades críticas.

Desafortunadamente, las estimaciones de tiempos de desarrollo estuvieron muy alejadas de la realidad, no solo por el imprevisto de la pandemia, sino por la necesidad de rehacer ciertas partes del sistema. Cuando nos percatamos de la imposibilidad de cumplir el plan de proyecto, realizamos una priorización de los requerimientos pendientes de desarrollo y los distribuimos en diferentes sprints. En las primeras iteraciones llevábamos muy poco control y se presentaron numerosos retrasos adicionales. Uno de los más importantes fue la aparición del problema de haber desarrollado un backend que consumía demasiada memoria del servidor, excediendo la cuota gratuita. La necesidad de volver a implementar el backend en una tecnología más liviana derivó en demoras ocasionadas por la codificación y prueba de funcionalidades que ya habían sido desarrolladas.

Tanto las funcionalidades pendientes de desarrollo, como las que debieron rehacerse por completo fueron priorizadas nuevamente, pero en esta oportunidad, no fueron distribuidas en sprints inmediatamente, sino que se decidió cuáles desarrollar al inicio de cada sprint. En un principio las estimaciones realizadas seguían siendo demasiado imprecisas, pero luego de ajustar la cantidad de horas destinadas a cada tarea, pudimos planificar con mayor certeza las últimas etapas del proyecto.

b) Arquitectura y diagramas principales

En esta sección se presenta la arquitectura del sistema, indicando los componentes que lo integran y el rol que desempeña cada uno. Seguidamente, se exponen los diagramas que se utilizaron para definir el modelo de dominio, la base de datos y las transiciones de estados de las entidades fundamentales.

I. Arquitectura

La arquitectura que sostiene el sistema desarrollado consta de 3 capas. La aplicación móvil que corre sobre el sistema operativo Android en forma nativa, la base de datos, la interfaz de administración desarrollada en Ionic y la WebAPI que actúa de intermediario en alguna de las operaciones entre los otros 3 componentes. Esta arquitectura se ilustra en la Figura 18 y a continuación se brindan más detalles sobre sus componentes.

a. Clientes Android

Las interfaces disponen en diferentes pantallas todos los elementos gráficos que le permiten al usuario interactuar con el resto del sistema. Estos se comunican con gestores que manejan la lógica de negocio local y se conectan, cuando es necesario a la WebAPI. Tanto los gestores como las interfaces disponen de un modelo del dominio idéntico al de la base de datos que permite manipular las entidades fundamentales del negocio.

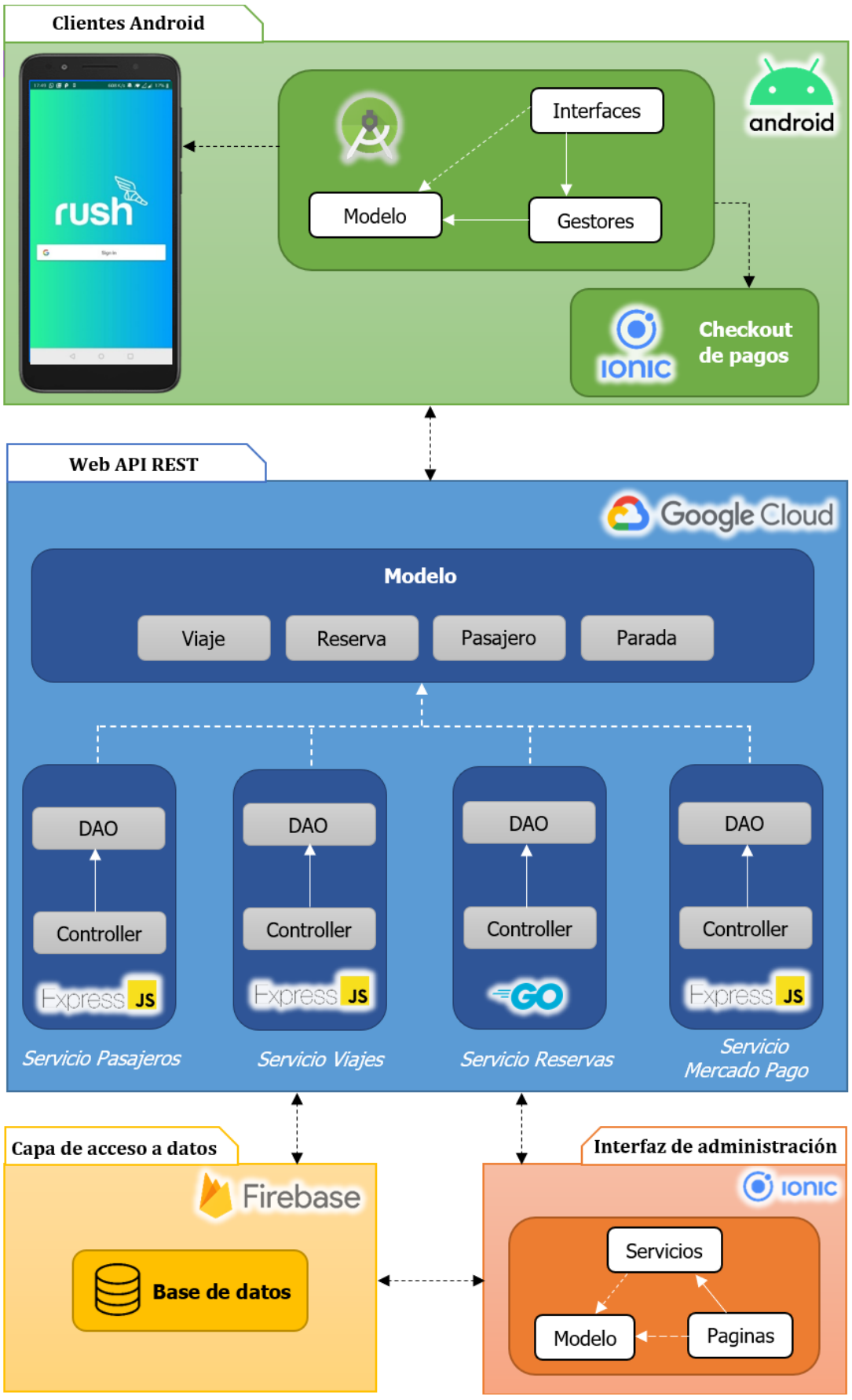


Figura 18: Arquitectura del sistema

Dentro de la aplicación se incluye también un módulo que se encarga de la gestión de las tarjetas de crédito de un usuario, así como también recibe y valida los datos para las autorizaciones y captura de pagos. El mismo se desarrolló de forma independiente como una aplicación web basada en el framework Ionic que se ejecuta en forma local en el dispositivo Android, desacoplándose del resto del desarrollo que corre en forma nativa sobre el sistema operativo.

a. WebAPI REST

Toda comunicación entre el teléfono y la base de datos, así como también toda interacción con la plataforma de pagos debe pasar primero por el Web API REST que se construyó para funcionar como intermediario de estas acciones.

El Web API funciona sobre la plataforma Google Cloud[2] y proporciona 4 servicios con diferentes incumbencias en el manejo de peticiones. Si bien se podría haber desarrollado un solo servicio que atienda todas las peticiones, pero que se habría visto cargado de miles de líneas de código difíciles de probar y depurar. La separación de servicios en base a diferentes incumbencias aumenta la legibilidad del código y favorece la cohesión de sus componentes.

El servicio de usuarios se encarga de crear usuarios y validar los inicios de sesión. Paralelamente, los servicios de viajes y reservas se conectan a la base de datos para obtener o persistir información de dichas entidades. Antes de almacenar cualquier registro, controlan la validez de los datos provistos, asegurando robustez al sistema. Asimismo, el servicio MercadoPago[3] se encarga de verificar algunos datos de pagos, enviarlos al API de la plataforma de pagos, y gestionar los diferentes errores que la misma pueda arrojar.

Finalmente, el Servicio de MercadoPago también posee procedimientos que se ejecutan periódicamente para encontrar las reservas de viajes pasados que no fueron validadas y aquellas que quedaron pendientes por cobrar. Cada vez que se ejecutan estos procedimientos, se intentan realizar todos los cobros pendientes.

b. Interfaz de administración

La interfaz de administración es una aplicación web desarrollada con Ionic que tiene por objeto brindar las funcionalidades administrativas a las empresas para que puedan gestionar sus viajes. Al igual que el resto de las capas superiores, esta aplicación posee su modelo y consta de diversas páginas en los que se ubican las funcionalidades para gestionar los viajes de una empresa. Las páginas se comunican con el backend y la base de datos a través de un conjunto de servicios inyectados como dependencias de cada recurso que los necesita.

[2] Google Cloud - https://cloud.google.com
[3] MercadoPago - API Developers site: https://www.mercadopago.com.ar/developers

Si bien la interfaz de administración nunca formó parte del plan de proyecto, el grupo se propuso desarrollar dos de sus funcionalidades más importantes para poder exhibir un funcionamiento más integral del sistema. Estas fueron el alta (individual y masiva) de viajes para una empresa y la cancelación de los mismos.

c. Base de Datos

Para el almacenamiento de la información referida a viajes y reservas del sistema optamos por utilizar una base de datos en línea alojada en el servicio Cloud Firestore de Firebase[4]. La misma nos provee con una base de datos de tipo NoSQL y resulta muy fácil de integrar en aplicaciones Android.

Los motivos que nos llevaron a optar por una base NoSQL van mucho más allá de las ventajas inherentes de la tecnología respecto a la facilidad de manejo de documentos estructurados en lugar de las tablas con datos planos que impone el modelo relacional. La aplicación que se desarrolla en este proyecto está pensada para uso masivo, lo que supone una elevada demanda de memoria al sistema que, de no ser manejada adecuadamente, podría tener un impacto muy negativo en los costos de mantenimiento. Numerosos autores como [18] ponderan las facilidades que poseen las bases NoSQL respecto a la escalabilidad de las mismas y el valor agregado que supone la posibilidad de incrementar el tamaño de la base de datos con mínimos costos. Esto nos asegura que a medida que nuestro sistema crezca, el aumento del volumen de datos manejado no supondrá un dolor de cabeza para el mantenimiento de la base de datos, y siendo que Firebase cobra un precio acorde al volumen de datos y transacciones, podemos estimar de forma mucho más simple la rentabilidad de este proyecto.

II. Diagrama de clases de análisis

Al principio del desarrollo debimos establecer claramente un modelo de análisis que nos permita sentar las bases de las entidades fundamentales que constituían el modelo del sistema (Figura 19). El mismo se realizó inicialmente como un mero diagrama borrador en UML y fue evolucionando durante el desarrollo, incorporando detalles en cada nueva etapa. Antes de profundizar en el desarrollo de alguna nueva parte del sistema, nos reunimos a revisar el modelo para evaluar si el mismo era capaz de cumplir los requerimientos a desarrollar en el sprint que estaba por comenzar y, si eran necesarios cambios, se buscaba la mejor forma de realizarlos y se evaluaba el potencial impacto de tales cambios en aquello que ya estaba desarrollado.

La entidad fundamental de este modelo es el viaje, que tiene asignado un colectivo y chofer para recorrer un conjunto de paradas. Tanto el chofer, como el colectivo son entidades independientes en el sistema, y ambas están asociadas a una empresa que posee dicho colectivo y emplea al chofer.

[4] Firebase - https://firebase.google.com

Otra entidad fundamental del modelo es la reserva, que no está asociada directamente al viaje sino a un trayecto específico del viaje (que se representa como otra entidad). Si un viaje parte de Paraná y llega a Buenos Aires haciendo paradas en Santa Fe, Rosario, San Nicolás y Pacheco, se pone en evidencia una característica del complejo problema a resolver. El tramo Santa Fe-Buenos Aires no tiene el mismo precio que el de Rosario-San Nicolas, asimismo se vuelve necesario llevar un conteo de la cantidad de asientos que quedan libres en cada tramo, de manera que se puedan filtrar los viajes llenos al momento de comprar. Estos atributos no corresponden ni al viaje ni a las paradas, sino a una entidad aparte que denominamos trayecto, y que representa un tramo del viaje entre 2 paradas específicas. Cada viaje tiene entonces tantos trayectos como combinaciones de pares de paradas existan. En el ejemplo antes mencionado, no solo estarán presentes los tramos que vayan de Rosario a San Nicolás y de San Nicolás a Buenos Aires, sino también habrá tramos más largos con paradas intermedias, como sería el caso de Rosario a Buenos Aires.

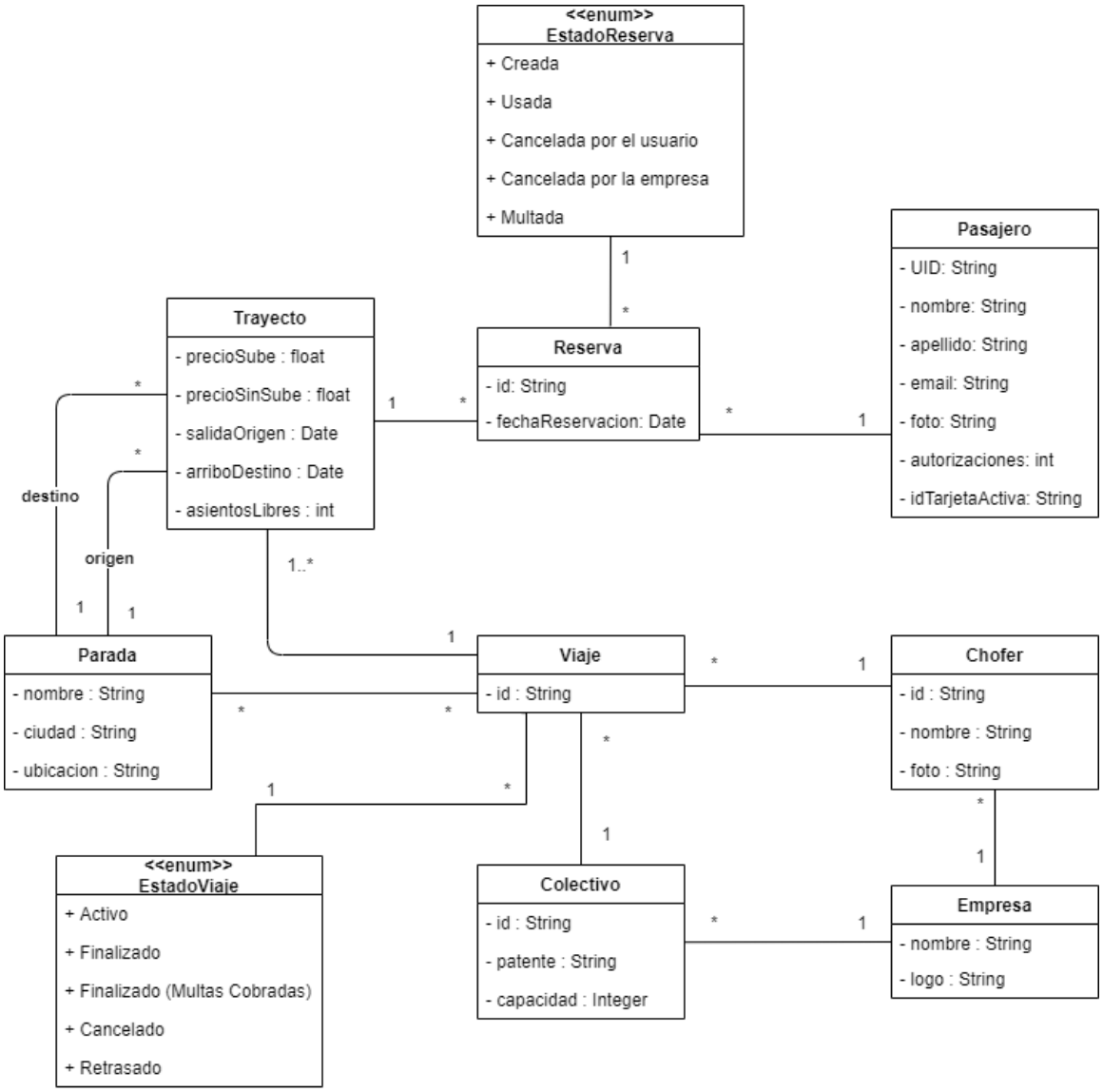


Figura 19: Diagrama de clases de análisis

Es por esto que decidimos que lo que se reserve no sea un viaje, sino un trayecto específico del viaje. Dicha reservación se almacena asociada al pasajero que la realizó, que también es otra entidad aparte. El modelo del pasajero posee toda la información identificatoria de un usuario de la aplicación, acompañada de algunos datos de su configuración de pagos.

III. Diagrama de base de datos

Una de las partes que más dificultades nos ocasionó al documentar los modelos fue la base de datos. Aunque numerosos autores no desaprovechan la oportunidad para generar nuevas propuestas, de momento no existe un estándar que haya alcanzado consenso suficiente para modelar de forma generalizada datos para bases de tipo NoSQL [19]. Adicionalmente, el modelo de la base de datos debía realizarse de forma tal que suponga la menor cantidad de peticiones de Firebase, de manera que se minimicen lo más posible los costos que se cargarían a nuestra tarjeta de crédito por la utilización del servicio.

Tuvimos entonces que pensar no solo el *qué* sino también el *cómo* del modelado. Para lo cual planteamos un modelo muy básico que consta de 6 entidades fundamentales: viajes, empresas, pasajeros, choferes, colectivos y reservas (véase Figura 20). Los pasajeros poseen un conjunto de reservas, al igual que sucede con las empresas que poseen conjuntos de choferes y colectivos. El viaje finalmente, es la entidad que más atributos posee, y también está asociada con casi todas las otras entidades, pues posee referencias a las instancias de los choferes que conducirán, el colectivo que se empleará y las reservas que se hicieron para ese viaje.

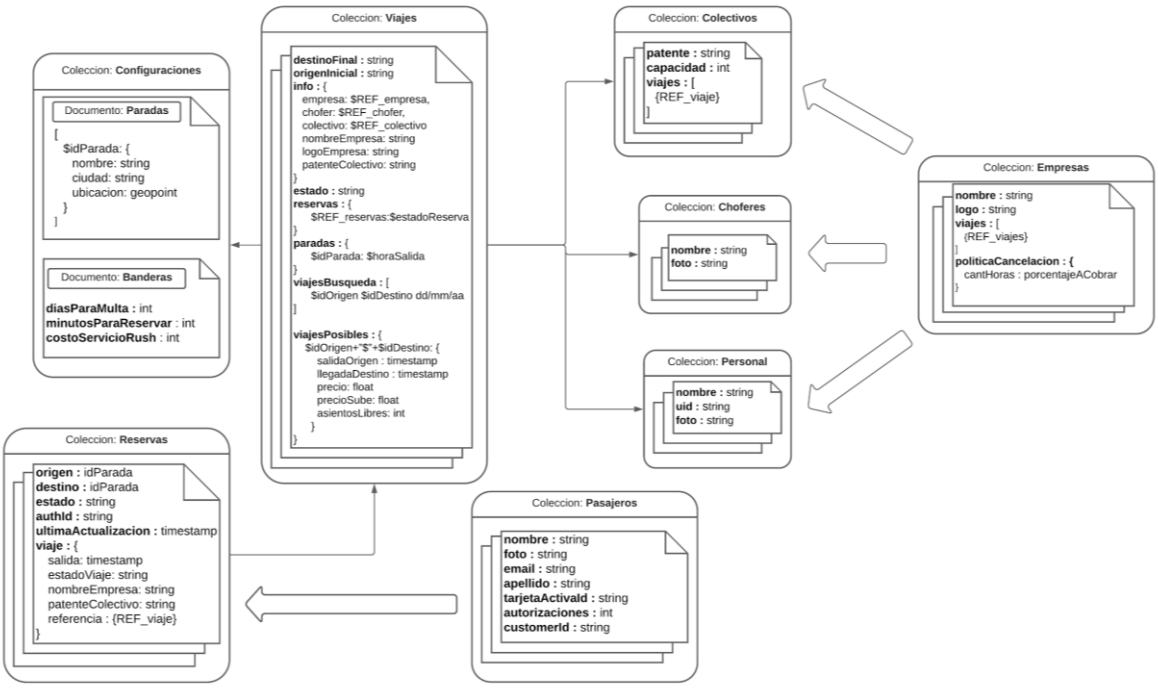


Figura 20: Diagrama de la base de datos

Adicionalmente, se creó una colección especial en la base de datos que almacena configuraciones varias. En él se puede setear la cantidad de días que deben transcurrir para que se cobre una multa, así como también la cantidad de horas antes del viaje que se autoriza la realización de reservas y la antelación mínima para efectuar una cancelación. Asimismo, un documento de configuración incluye un listado completo de las paradas que pueden tener los viajes. Esto permite obtener un listado completo de la totalidad de paradas para agilizar su búsqueda sin necesidad de hacer excesivas llamadas a la base de datos (que implicarían un costo mayor). Cada vez que se busque una parada en el formulario de Nueva Reserva o se deba consultar el Id de la parada de un viaje, se dispondrá del listado completo en forma local en cada cliente y la búsqueda se podrá hacer mucho más rápida, sin que esto implique un consumo de memoria demasiado elevado, pues el listado de paradas no llega a ocupar 2 MB de espacio.

IV. Máquinas de estados de las entidades fundamentales

Hay 2 entidades fundamentales que se caracterizan por poseer estados. Tanto las reservas como los viajes transitan por diferentes que, si bien están estrechamente relacionados, poseen ciertas diferencias que resulta importante destacar.

Es importante comprender qué representan cada uno de los estados por los que puede pasar cada una de las entidades para así interpretar adecuadamente por qué el sistema distingue entre estados y por qué debe actuar diferente ante diferentes circunstancias. Es por esto que a continuación se presentan las máquinas de estados correspondientes a los viajes y las reservas.

a. Viajes

Los viajes transitan únicamente por dos estados definidos en la aplicación: ACTIVO y CANCELADO (véase Figura 21). Los viajes cancelados son aquellos que la empresa decidió no realizar por motivos de fuerza mayor. Sin embargo, la semántica de los viajes activos es mucho más amplia e involucra dos subestados: “Pendiente” y “Realizado”.

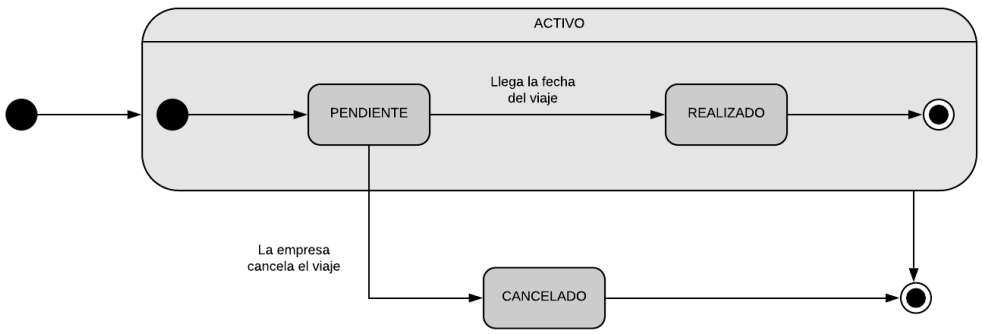


Figura 21: Máquina de estados de la entidad Viaje

Desde que el viaje es creado y hasta que se llega a la fecha prevista de partida, el mismo permanece en el subestado “Pendiente”. Una vez pasada la fecha de partida, el viaje pasa a considerarse como “Realizado”. Solamente pueden cancelarse los viajes pendientes. En el sistema no se distinguen dos estados distintos entre “Pendiente” y “Realizado” porque lo mismo implicaría llamar innecesariamente a la base de datos a actualizarse con datos que tranquilamente pueden inferirse a partir de la fecha de viaje.

b. Reserva

Las reservas tienen una máquina de estados más compleja (véase Figura 22). Al crearlas se mantienen en el estado “CREADA”. Si el usuario decide cancelar dicha reserva, la misma pasa al estado “CANCELADA_SIN_PAGAR” que indica que el usuario canceló correctamente la reserva, pero aún el sistema no ha verificado si corresponde cobrar una parte de la misma por haberlo hecho demasiado tarde. Una vez que se realizó el cobro pertinente, la misma pasa al estado “CANCELADA”. Si por el contrario, la empresa es la que decide cancelar el viaje y, en consecuencia, todas sus reservas, la misma pasa directamente al estado “CANCELADA” porque no corresponde comprobar ni cobrar nada.

Si el pasajero se presenta en la terminal el día pactado y valida su reserva con el chofer, la misma pasa al estado “USADA_SIN_PAGAR”. Luego de cobrar la comisión que corresponde a Rush por el pasaje, la misma queda en estado “USADA”.

Si la reserva no se usa ni se cancela, el pasajero merece ser multado por haberse reservado un asiento que jamás usó. Un procedimiento se ejecuta periódicamente para encontrar las reservas de viajes pasados que deben multarse. Una vez que se cobra exitosamente dicha multa, la reserva se marca en estado “MULTADA”. Si hay algún fallo en el cobro, permanece en estado “CREADA” y se la intentará cobrar en la próxima ejecución del procedimiento.

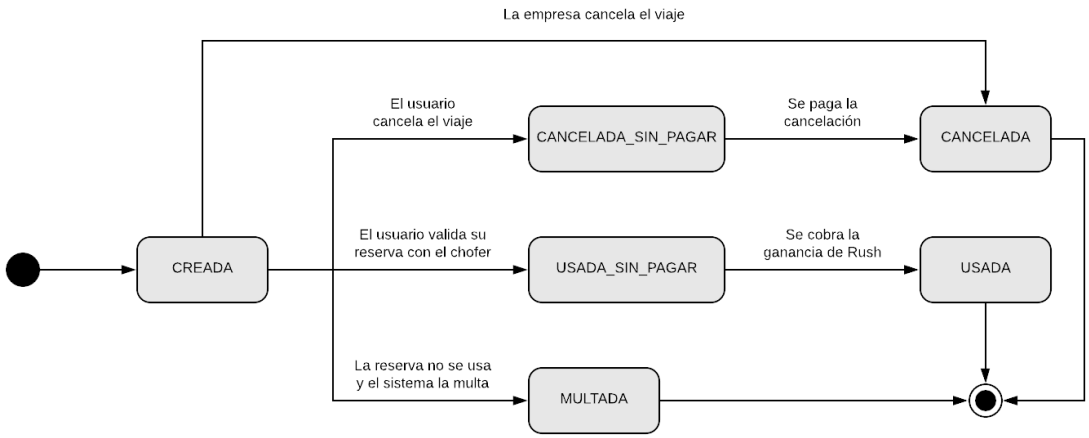


Figura 22: Máquina de estados de la entidad Reserva

El motivo de la existencia de los estados intermedios “CANCELADA_SIN_PAGAR” y “USADA_SIN_PAGAR” se debe a la existencia de otros procedimientos que periódicamente buscan las reservas que los usuarios usaron y cancelaron pero que por algún motivo no se pudieron cobrar. Estos procedimientos reintentan los cobros que fallaron y, si corresponde, actualizan la base de datos con los estados finales “USADA” y “CANCELADA”.

c) Herramientas empleadas

A lo largo del desarrollo del proyecto, fue necesario emplear un conjunto muy variado de tecnologías, herramientas y librerías que permitan no solo satisfacer los requerimientos del sistema, sino también puedan integrarse en armonía como parte del sistema. Los recursos empleados incluyen no solo lenguajes de programación, sino también frameworks, plataformas de gestión de cambios, entornos de desarrollo, APIs y librerías de testing. El listado completo de las herramientas utilizadas son las siguientes.

GOLANG	**GoLang:** Lenguaje empleado para el desarrollo del Servicio de Reservas del backend.
express	**Express JS:** Framework empleado en el desarrollo de los servicios del backend de Viajes, MercadoPago y Pasajeros.
Java	**Java + Android:** Empleados en el desarrollo de las aplicaciones clientes del sistema.
Firebase	**Firebase:** Tecnología empleada para la persistencia de toda la base de datos del sistema.
IONIC	**Ionic:** Framework empleado para el desarrollo de la interfaz gráfica de la billetera del cliente Android y la plataforma de administración del sistema.
ANGULAR	**Angular:** Framework de base para la organización de los componentes de Ionic en las interfaces de billetera y administración.
mercado pago	**MercadoPago:** API empleada para la gestión de las funcionalidades de autorizaciones y pagos de tarjetas en el sistema.
GitHub	**GitHub:** Plataforma empleada para la gestión de cambios del proyecto.
JUnit	**JUnit:** Herramienta para la automatización de prueba unitaria empleada en gestores y DAOs de la aplicación Android.
espresso	**Espresso:** Herramienta para la automatización de pruebas de UI empleado en las pantallas de la aplicación Android.
mockito	**Mockito:** Herramienta para la generación de mocks de objetos ajenos a la clase involucrada en un test unitario. Empleado en todos los tests

	de Android.
	**Jest:** Librería empleada para automatizar el testing del backend.
	**Sinon JS:** Empleado para la generación de objetos espías y mocks en el testing del backend.
	**Android Studio:** IDE utilizado para el desarrollo de la aplicación de Android.
	**Visual Studio Code:** IDE empleado para el desarrollo del backend y las páginas en Ionic.
	**Google Sign-In:** API empleada para la autenticación de usuarios en la aplicación.
	**Google Cloud:** Plataforma de desarrollo web empleada para el deploy en línea del backend del sistema.

d) Implementación

En esta sección se hace un breve resumen de las principales decisiones que se tomaron al momento de implementar el proyecto, haciendo especial hincapié en los patrones empleados para organizar las clases del modelo. Adicionalmente, se presenta el pseudocódigo de los principales endpoints del backend, de manera que se brinde un panorama de todas las operaciones que la WebAPI realiza.

I. Clientes Android

a. Modelo

En los clientes Android, los objetos del modelo no se representan con clases como es habitual en muchos otros proyectos. En su lugar, se emplean los objetos DocumentSnapshot que provee Firebase y permiten acceder y persistir los datos de forma directa. Sin embargo, para acceder a los atributos de cada objeto es necesario saber de antemano el nombre del campo en que se almacenan dentro del Snapshot. Si bien podrían obtenerse dichos datos indicando los nombres con strings planos, la mantenibilidad del código se volvería muy tediosa si se presentare un cambio en la estructura de los objetos. Es por esto que sí fue necesario construir un modelo que consta de varias interfaces que incluyen los nombres de los campos accesibles del Snapshot. De esta manera, no solo se disponía de un ayuda memoria de la estructura de cada entidad, sino también nos asegurábamos la posibilidad de efectuar cambios en tales estructuras de manera mucho más sencilla.

Excepcionalmente, se utilizan algunos objetos Java para representar las entidades de "Pasajero" y "Parada". En ambos casos, requerimientos propios del lenguaje y las APIs involucradas no permitían acometer ciertas acciones, por lo que no hubo otra alternativa que mapearlos a objetos Java.

b. Conexión al backend y base de datos.

Los DAOs de la aplicación actúan como intermediarios entre la aplicación y las interfaces externas que permiten acceder a datos. Las clases DAO están desarrolladas como clases Java y todas responden al patrón Singleton. Las mismas pueden comunicarse tanto con el backend, como también con la base de datos en forma directa. En el caso de las altas, bajas y modificaciones de datos, todas las solicitudes deben pasar por el backend que controlará que las mismas sean consistentes y no tengan faltantes de datos. Sin embargo, las consultas de datos no solo no requieren ninguna comprobación previa (salvo por motivos de seguridad implementados en la base de datos), sino que también es muy improbable que su ejecución cambie y requiera actualizarse. Además, es completamente innecesario incorporar a una consulta la sobrecarga de una conexión al backend que no tenga ninguna lógica más que la de simplemente obtener los datos. Es por tal motivo que las consultas a la base de datos son realizadas directamente en el código de la aplicación Android.

c. Gestores

Los gestores de la aplicación manejan toda la lógica de negocios que procesa los datos que luego son exhibidos en las interfaces. Se comunica constantemente con los DAOs para solicitar, recibir y posteriormente procesar información de la base de datos. Al igual que los DAOs, los gestores también se implementan en clases Java que responden al patrón Singleton.

d. Interfaces

Las interfaces de la aplicación se construyeron haciendo uso de dos herramientas. Por un lado, el editor incorporado de Android Studio permitió construir las pantallas principales de la aplicación, incluyendo la de bienvenida, el menú principal y la búsqueda de viajes y reservas. Por otro lado, las interfaces que se presentan en la ventana emergente del menú "Mi billetera" se desarrollaron haciendo uso del framework Ionic que se ejecuta en forma local dentro de la aplicación. La ventana emergente que confirma la autorización de las reservas también fue desarrollada en Ionic y tuvo que implementar una comunicación con la aplicación nativa para informar a Android que debe cerrar la ventana emergente una vez que la autorización resulte exitosa.

II. Servicios del back-end

Todos los servicios del back-end constan de un Controller, un DAO y un Modelo. El Controller se limita exclusivamente a recibir requests y determinar cuál response corresponde

devolver. Los DAO se encargan de toda la lógica de negocios restante, que siempre implica al menos una comunicación con la base de datos. Hay un método del DAO para cada endpoint.

El flujo de todos los controller es similar y se explica con el siguiente seudocódigo:

```
endpointRush(request, response){
      if(!validarTokenFirebase(request))
            response.send(BAD_REQUEST)
      if(response.body != null){
            resultado = metodoDaoParaEsteEndpoint(request)
                  .catch( response.send(INTERNAL_SERVER_ERROR))
            switch(resultado){
                  case 1 : response.send(OK)
                  case 2 : response.send(BAD_REQUEST)
                  case 3 : response.send(NOT_FOUND)
                  case 4 : response.send(INTERNAL_SERVER_ERROR)
            }
      }
}
```

a. Servicio Pasajero

El servicio de pasajero es el encargado de registrar los nuevos usuarios en el sistema haciendo uso de la API de Google, así como también es capaz de autenticar a los usuarios al momento de iniciar sesión. El mismo consta de dos endpoints, uno para cada funcionalidad.

Endpoint **/NewPasajero** (Method: POST)

Propósito: Crea un nuevo usuario en la aplicación extrayendo sus datos de Google y almacenándolos en la Base de Datos.

Entrada: Datos del usuario (nombre, apellido, email y foto)

```
newPasajero(datosUsuario){

  var newUsuario
  newUsuario.apellido = datosUsuario.apellido
  newUsuario.foto = datosUsuario.foto
  newUsuario.nombre = datosUsuario.nombre
  newUsuario.email = datosUsuario.email

  firebase.guardarPasajero(newUsuario)
  response.send(OK)

}
```

Endpoint **/Pasajero** (Method: GET)

Propósito: Obtiene el pasajero actualmente logueado y devuelve todos sus datos

Entrada: Token de autenticación del pasajero.

```
getPasajero(id){

  var pasajero = firebase.traerPasajero(id)
      if(pasajero == null){
         response.send(NO_CONTENT)
  }

  response.send(OK, pasajero)
}
```

b. Servicio Mercadopago

El servicio de MercadoPago es el intermediario entre el sistema y la API de MercadoPago. Posee diferentes endpoints que permiten realizar todas las transacciones previstas en las historias de usuario y es el único servicio que posee acceso a la API de pagos. Todos los componentes del sistema que requieran mostrar o utilizar información sobre autorizaciones, capturas de fondos y medios de pago deben solicitarla a este servicio.

El manejo de tarjetas en este servicio involucra la existencia de una tarjeta particular llamada "activa", que constituye el medio preferido de pago al momento de autorizar reservas. Dicha tarjeta puede elegirse libremente entre todas las tarjetas cargadas de un usuario.

Además de los endpoints que se ejecutan a demanda, este servicio también posee dos métodos adicionales denominados "CronJob" que se ejecutan periódicamente para cobrar las multas y pagos pendientes. Los mismos se implementaron gracias al paquete "cron" [20] que justamente está destinado a programar ejecuciones periódicas en Node.js.

Endpoint **/CargarTarjeta** POST

Propósito: Permite asociar una nueva tarjeta al sistema para realizar pagos.

Entrada: Token de la nueva tarjeta, ID del usuario.

```
cargarTarjeta(tokenTarjeta, userId){
      var pasajero = firebase.traerPasajero(userId)
      if(pasajero == null){
         response.send(NOT_FOUND)
  }

  var email = pasajero.email;
  if(email == null){
         response.send(INTERNAL_SERVER_ERROR)
  }
```

```
      var customerIdMercadoPago = mp.obtenerCustomerId(email)
      if(customerIdMercadoPago  == null){
         response.send(INTERNAL_SERVER_ERROR)
  }

      var tarjeta = mp.nuevaTarjeta(tokenTarjeta,customerIdMercadoPago)
      if(tarjeta  == null){
         response.send(INTERNAL_SERVER_ERROR)
  }

      var autorizacion = mp.autorizar(tarjeta)
      var cancelacion = null
      if(autorizacion  == null){
         response.send(INTERNAL_SERVER_ERROR)
  }else if(autorizacion.autorizado == true){
            cancelacion = mp.cancelarAutorizacion(autorizacion)
      }else{
            response.send(CONFLICT)
      }

   if(cancelacion  == null || autorizacion.cancelado == false){
         response.send(INTERNAL_SERVER_ERROR)
  }

  if (mp.obtenerTarjetasDeUsuario(pasajero.customerId).length == 1){
            pasajero.tarjetaActiva = tarjeta.id
            actualizarDatosPasajero(pasajero)
  }
      response.send(OK)
}
```

Endpoint **/Autorizar** POST

Propósito: Permite generar una nueva autorización de la tarjeta para crear una reserva.

Entrada: ID de usuario, Token de la tarjeta, Datos de la reserva que se crea (ID del viaje, terminal de origen y terminal de destino)

```
autorizar(userId, tokenTarjeta, datosReserva){

  if(datosReserva.viajeId == null && datosReserva.origen == null &&
     datosReserva.destino == null){
         response.send(BAD_REQUEST)
  }

  var pasajero = firebase.traerPasajero(userId)
  if(pasajero == null){
         response.send(NOT_FOUND)
  }
  if(pasajero.email == null){
         response.send(INTERNAL_SERVER_ERROR)
  }
  if(pasajero.tarjetaActiva == null){
         response.send(INTERNAL_SERVER_ERROR)
  }
```

```
      var customerIdMercadoPago = mp.obtenerCustomerId(pasajero.email)
      if(customerIdMercadoPago  == null){
         response.send(INTERNAL_SERVER_ERROR)
  }

  var tarjetas = mp.obtenerTarjetasDeUsuario(pasajero.customerId)
      if(tarjetas == null){
         response.send(INTERNAL_SERVER_ERROR)
  }

  var tarjetaActiva = tarjetas.find(t-> t.id == pasajero.tarjetaActiva)

  var viaje = firebase.obtenerViaje(datosReserva.viajeId)
  if(viaje == null){
         response.send(NOT_FOUND)
  }

  var origen = datosReserva.origen
  var destino = datosReserva.destino
  var monto = viaje[origen + "$" + destino].precio

  var auth = mp.autorizar(tokenTarjeta, tarjeta, customerId, monto)
if(auth == null || auth.id = null){
         response.send(INTERNAL_SERVER_ERROR)
  }

      datosReserva.authId = auth.id
      datosReserva.userId = userId

      var resultadoCreacion = http.post(URL_CREAR_RESERVA, datosReserva)

      if(resultadoCreacion == null || resultadoCreacion.status != 200){
            var cancelacion = mp.cancelarAutorizacion(autorizacion)
            if(cancelacion  == null || autorizacion.cancelado == false){
                  response.send(CONFLICT)
            }else{
                  response.send(INTERNAL_SERVER_ERROR)
            }
  }
      response.send(OK)
}
```

Endpoint **/Cards** GET

Propósito: Permite obtener el listado de tarjetas del usuario logueado.

Entrada: ID de usuario

```
obtenerTarjetas(userId){

  var pasajero = firebase.traerPasajero(userId)
      if(pasajero == null){
         response.send(NOT_FOUND)
  }
  if(pasajero.customerId == null){
         response.send(INTERNAL_SERVER_ERROR)
```

```
  }

  var tarjetas = mp.obtenerTarjetasDeUsuario(pasajero.customerId)
      if(tarjetas == null){
         response.send(NO_CONTENT)
  }
      response.send(OK, tarjetas)

}
```

Endpoint **/TarjetaActiva** GET

Propósito: Permite obtener la tarjeta seleccionada como medio de pago preferido por el usuario.

Entrada: ID del usuario

```
getTarjetaActiva(userId){

  var pasajero = firebase.traerPasajero(userId)
  if(pasajero == null){
         response.send(NOT_FOUND)
  }
  if(pasajero.tarjetaActiva == null){
         response.send(INTERNAL_SERVER_ERROR)
  }

  var tarjetas = mp.obtenerTarjetasDeUsuario(pasajero.customerId)
  if(tarjetas.length == 0){
         response.send(NO_CONTENT)
  }

  var tarjetaActiva = tarjetas.find(t-> t.id == pasajero.tarjetaActiva)

  response.send(OK, tarjetaActiva)
}
```

Endpoint **/CambiarTarjetaActiva** POST

Propósito: Permite elegir un medio de pago preferido diferente al que estaba definido anteriormente.

Entrada: ID de usuario, ID de la nueva tarjeta activa

```
cambiarTarjetaActiva(userId, nuevaTarjetaActiva){

  var pasajero = traerPasajero(userId)
  if(pasajero == null){
         response.send(NOT_FOUND)
  }

  var tarjetas = mp.obtenerTarjetasDeUsuario (pasajero.customerId)
```

```
  if(tarjetas == null || tarjetas.length == 0){
        response.send(CONFLICT)
  }

      if (tarjetas.find(t-> t.id == nuevaTarjetaActiva) == null){
      response.send(CONFLICT)
  }

  firebase.cambiarTarjetaActiva(userId, nuevaTarjetaActiva)

      response.send(OK)
}
```

Endpoint **/CancelarAutorizacion** POST

Propósito: Permite cancelar una autorización sin cobrar ninguna multa. Se lo llama cuando una empresa decide cancelar un viaje y debe cancelar unilateralmente todas las reservas de ese viaje.

Entrada: ID de usuario, ID de la reserva cuya autorización se cancela

```
cancelarAutorizacion(userId, reservaId){

  var pasajero = firebase.traerPasajero(userId)
  if(pasajero == null){
        response.send(NOT_FOUND)
  }

  var reserva = firebase.traerReserva(userId, reservaId)
      if(reserva == null){
        response.send(NOT_FOUND)
  }
  if(reserva.authId == null){
        response.send(CONFLICT)
  }

      var cancelacion = mp.cancelarAutorizacion(autorizacion)
      if(cancelacion  == null || autorizacion.cancelado == false){
        response.send(INTERNAL_SERVER_ERROR)
  }
      response.send(OK)
}
```

Endpoint **/CobrarGanancia** POST

Propósito: Permite cobrar el margen de ganancia que le corresponde a Rush sobre la autorización realizada.

Entrada: ID de usuario, ID de la reserva cuya ganancia se cobra

```
cobrarGanancia(userId, reservaId){
```

```
  var pasajero = firebase.traerPasajero(userId)
  if(pasajero == null){
         response.send(NOT_FOUND)
  }

  var reserva = firebase.traerReserva(userId, reservaId)
      if(reserva == null){
         response.send(NOT_FOUND)
  }
  if(reserva.authId == null){
         response.send(INTERNAL_SERVER_ERROR)
  }

  var ganancia = firebase.obtenerGanancia()
      if(ganancia == null){
         response.send(INTERNAL_SERVER_ERROR)
  }

  var captura = mp.capturaParcial(autorizacion, ganancia)
      if(captura == null || captura.captured = true){
         response.send(INTERNAL_SERVER_ERROR)
  }
      response.send(OK)
}
```

Endpoint **/CancelarPagoReserva** POST

Propósito: Se utiliza cuando un usuario decide cancelar su reserva. Para ello, primero verifica con cuanta anticipación se está realizando la cancelación. Si es muy tarde, corresponde cobrar una multa (que constituye un porcentaje del valor del pasaje). Si es muy temprano, la cancelación puede ser gratuita y simplemente se cancela la autorización.

Entrada: ID de usuario, ID de la reserva que se cancela.

```
cancelarPagoReserva(userId, reservaId){

  var pasajero = firebase.traerPasajero(userId)
  if(pasajero == null){
         response.send(NOT_FOUND)
  }

  var reserva = firebase.traerReserva(userId, reservaId)
      if(reserva == null){
         response.send(NOT_FOUND)
  }

      var viaje = firebase.traerViaje(reserva.viaje)
      if(viaje == null){
         response.send(NOT_FOUND)
  }

      var empresa = firebase.traerEmpresa(viaje.empresa)
      if(empresa == null){
         response.send(NOT_FOUND)
```

```
  }

      //Se obtiene un porcentaje a cobrar dependiendo la política de
    cancelación de la empresa
      var porcentajeAcobrar = obtenerPorcentaje(reserva,empresa.politica)

      var autorizacion = mp.traerAutorizacion(reserva.authId)
      if(autorizacion  == null || autorizacion.captured == true ||
      autorizacion.cancelled == true || autorizacion.monto  == null){
         response.send(INTERNAL_SERVER_ERROR)
  }
  if(porcentajeAcobrar == 0){
            var cancelacion = mp.cancelarAutorizacion(autorizacion)
            if(cancelacion  == null || autorizacion.cancelado == false){
                  response.send(INTERNAL_SERVER_ERROR)
            }
  }else{
            var monto = autorizacion.monto * porcentajeAcobrar / 100
            var captura = mp.capturaParcial(autorizacion, monto)
            if(captura == null || captura.captured == false){
                  response.send(INTERNAL_SERVER_ERROR)
            }
  }

  response.send(OK)
}
```

CronJob **MultarReservas**

Propósito: Se encarga de identificar las reservas que no fueron ni usadas ni canceladas y que corresponde que sean multadas.

Frecuencia de ejecución: 1 vez al día.

Entrada: (Los cron-job no tienen entrada)

```
multarReservas(){

  var reservas = firebase.traerReservasAMultar()
  reservas.forEach( reserva -> {
            var autorizacion = mp.traerAutorizacion(reserva.authId)
            if(autorizacion  == null){
                  console.log("No se encontró la autorización")
            }
            if(autorizacion.captured  == true){
                  firebase.actualizarEstado(reserva.id, "MULTADA")
            }else{
                  var captura = mp.capturaTotal(autorizacion)
                  if(captura == null){
                       console.log("No se pudo capturar la autorización")
                  }
                  if(captura.captured == true){
                       firebase.actualizarEstado(reserva.id, "MULTADA")
                  }
            }
```

```
   }
}
```

CronJob **CobrarPendientes**

Propósito: Permite cobrar las reservas usadas o canceladas que hayan quedado pendientes de cobro por algún error en la comunicación de servicios.

Frecuencia de ejecución: 1 vez al día.

Entrada: (Los cron-job no tienen entrada)

```
cobrarPendientes(){

  var reservas = firebase.traerReservasPendientesDeCobro()
  reservas.forEach( reserva -> {
            var autorizacion = mp.traerAutorizacion(reserva.authId)
            if(autorizacion  == null){
                  console.log("No se encontró la autorización")
            }
            var nuevoEstado
            if(reserva.estado  == "CANCELADA_SIN_PAGAR"){
                  nuevoEstado = "CANCELADA"
            }else{
                  nuevoEstado = "USADA"
            }
            if(autorizacion.captured  == true){
                  firebase.actualizarEstado(reserva.id, nuevoEstado)
            }else{
                  var montoCobro = 0
                  if(reserva.estado  == "CANCELADA_SIN_PAGAR"){
                     var viaje = firebase.traerViaje(reserva.viaje)
                     if(viaje == null){
                        console.log("No se encontró el viaje")
                     }

                     var empresa = firebase.traerEmpresa(viaje.empresa)
                     if(empresa == null){
                        console.log("No se encontró la empresa")
                     }
                     var porcentajeAcobrar =
                        obtenerPorcentaje(reserva, empresa.politica)
                     var monto = autorizacion.monto
                     var montoCobro = monto * porcentajeAcobrar / 100

                  }else{
                        var montoCobro = ganancia
                  }
                  if(montoCobro == 0){
                    var cancelacion =
                                mp.cancelarAutorizacion(autorizacion)
                    if(autorizacion.cancelado == true){
                       firebase.actualizarEstado(reserva.id, nuevoEstado)
                    }
                  }else{
```

```
                    var captura =
                        mp.capturaParcial(autorizacion, montoCobro)
                    if(captura.captured == true){
                       firebase.actualizarEstado(reserva.id, nuevoEstado)
                    }
            }
  }
}
```

c. Servicio Reserva

El servicio de Reservas provee las funcionalidades esenciales para manipular las reservas a través de su ciclo de vida. Posee 4 endpoints destinados a crear y validar reservas, así como también cancelarlas de forma individual.

Endpoint **/CrearReserva** POST

Propósito: Permite crear una nueva reserva para el usuario logueado.

Entrada: Datos de la reserva a crear.

```
crearReserva(datosReserva){

  if(datosReserva.viajeId == null && datosReserva.origen == null &&
     datosReserva.destino == null && datosReserva.estado == null &&
         datosReserva.userId == null && datosReserva.authId == null){
         response.send(BAD_REQUEST)
  }

  firebase.runTransaction({

     var viaje = firebase.traerViaje(viajeId)
     if(viaje == null){
            response.send(NOT_FOUND)
     }

     if(viaje.estado != 'ACTIVO'){
            response.send(INTERNAL_SERVER_ERROR)
     }

     var origen = datosReserva.origen
     var destino = datosReserva.destino

     var trayecto = viaje.traerTrayecto(origen, destino)
     if(trayecto != null){
            response.send(INTERNAL_SERVER_ERROR)
     }
     if(trayecto.fecha.before(now())){
            response.send(INTERNAL_SERVER_ERROR)
     }
     if(trayecto.asientosLibres < 1){
            response.send(INTERNAL_SERVER_ERROR)
     }
```

```
          datosReserva.ultimaActualizacion = now()
          var idReserva = firebase.crearReserva(userId, datosReserva)
          if(idReserva  == null){
             response.send(INTERNAL_SERVER_ERROR)
      }

          trayecto.actualizarAsientosLibres(-1, origen, destino)
          viaje.actualizarTrayecto(trayecto)
          viaje.agregarReserva(reserva)
          firebase.actualizarViaje(viaje)

    })
    response.send(OK, idReserva)
}
```

Endpoint **/CancelarReserva** POST

Propósito: Permite que un usuario pueda cancelar una reserva propia.

Entrada: ID del usuario e ID de la reserva a cancelar.

```
cancelarReserva(userId, reservaId){

  var pasajero = firebase.traerPasajero(userId)
  if(pasajero == null){
         response.send(NOT_FOUND)
  }

  var reserva = firebase.traerReserva(userId, reservaId)
      if(reserva == null){
         response.send(NOT_FOUND)
  }
  if(reserva.authId == null || reserva.estado != 'CREADA'||
            reserva.salida.before(now())){
         response.send(INTERNAL_SERVER_ERROR)
  }

reserva.estado = 'CANCELADA_SIN_PAGAR'
       var result = firebase.actualizarReserva(userId, reserva)
       if(result == null){
          response.send(INTERNAL_SERVER_ERROR)
  }
  var postBody = {userId, reservaId}

  var response = http.post(URL_CANCELAR_RESERVA_MERCADO_PAGO, postBody)
  if(response != null && response.status == 200){
            reserva.estado = 'CANCELADA'
       firebase.actualizarReserva(userId, reserva)
  }
       response.send(OK)
}
```

Endpoint **/ValidarReserva** POST

Propósito: Permite validar una reserva al momento de abordar el viaje para que la misma quede marcada como USADA en la base de datos.

Entrada: ID del usuario e ID de la reserva a validar.

```
validarReserva(userId, reservaId){

  var pasajero = firebase.traerPasajero(userId)
  if(pasajero == null){
         response.send(NOT_FOUND)
  }

  var reserva = firebase.traerReserva(userId, reservaId)
      if(reserva == null){
         response.send(NOT_FOUND)
  }
  if(reserva.authId == null || reserva.estado != 'CREADA'){
         response.send(INTERNAL_SERVER_ERROR)
  }

  reserva.estado = 'USADA_SIN_PAGAR'
      var result = firebase.actualizarReserva(userId, reserva)
      if(result == null){
         response.send(INTERNAL_SERVER_ERROR)
  }
  var postBody = {userId, reservaId}

  var response = http.post(URL_COBRAR_GANANCIA_MERCADO_PAGO, postBody)
  if(response != null && response.status == 200){
      reserva.estado = 'USADA'
      firebase.actualizarReserva(userId, reserva)
  }
      response.send(OK)
}
```

d. <u>Servicio Viaje</u>

El servicio de Viajes permite a las empresas dar de alta viajes tanto en forma individual como masiva y cancelarlos cuando se lo considere conveniente. La cancelación de cualquier viaje implica también la cancelación de todas las reservas asociadas a la misma.

Endpoint **/NewViaje** POST

Propósito: Permite que las empresas puedan crear uno o más viajes con un conjunto de trayectos y horarios definidos.

Entrada: Datos del nuevo viaje.

```
newViaje(datosViaje){

    var empresa = firebase.obtenerEmpresa(datosViaje.empresa)
```

```
    if(empresa == null){
        response.send(NOT_FOUND)
    }

    var colectivo = firebase.obtenerColectivo(datosViaje.colectivo)
    if(colectivo == null){
        response.send(NOT_FOUND)
    }

    if(datosViaje.chofer){
        var chofer = firebase.obtenerChofer(datosViaje.chofer)
        if(chofer == null){
            response.send(NOT_FOUND)
        }
    }

    if(datosViaje.fechas.find(fecha => fecha < now()) != null){
        response.send(BAD_REQUEST)
    }

    var paradasDB = firebase.obtenerParadas()
    if(paradasDB == null){
        response.send(INTERNAL_SERVER_ERROR)
    }

    if(datosViaje.paradas.find(parada=> {
                        !paradasDB.includes(parada)})!=null){
        response.send(BAD_REQUEST)
    }

    var viajeDB;
    var batchedWrite = firebase.batch();

    for(var i = 0; i < datosViaje.fechas.length; i++){
        viajeDB = mapearViajeDB(datosViaje, i)
        batchedWrite.write(firebase.newDocument(), viajeDB)
    }

    var result = batchedWrite.commit()

    if(result == null){
        response.send(INTERNAL_SERVER_ERROR)
    }


    response.send(OK)
}
```

Endpoint **/CancelarViaje** POST

Propósito: Permite cancelar un viaje creado previamente. Si existen reservas asociadas al viaje cancelado, también se las cancelan en forma masiva.

Entrada: ID del viaje a cancelar

```
cancelarViaje(idViaje){
```

```
    var viaje = firebase.obtenerViaje(idViaje)
    if(viaje == null){
        response.send(NOT_FOUND)
    }
    if(viaje.estado != "ACTIVO"){
        response.send(INTERNAL_SERVER_ERROR)
    }

    var batchedWrite = firebase.batch();

    batchedWrite.write(viaje, estado, "CANCELADO")

    viaje.reservas.forEach( reserva => {
        var postBody = {reserva.user, reserva.id}
        var response = http.post(URL_CANCELAR_AUTORIZACION, postBody)
        if(response != null && response.status == 200){
            batchedWrite.write(reserva, estado, "CANCELADA")
        }else{
            response.send(INTERNAL_SERVER_ERROR)
        }
    })

    var result = batchedWrite.commit()

    if(result == null){
        response.send(INTERNAL_SERVER_ERROR)
    }

    response.send(OK)
}
```

e) Testing

En esta sección se detallan las prácticas realizadas durante la fase de pruebas del sistema. Se hace mención de las tecnologías empleadas en el testing unitario, tanto en la aplicación Android como en el backend. Asimismo, se comenta la labor realizada en las pruebas de integración y se hace mención a las pruebas del sistema que se encuentran documentadas en los anexos.

I. Testing Unitario.

Una etapa primordial para cualquier proceso de desarrollo ágil es la construcción de test unitarios que permitan corroborar con rapidez el correcto funcionamiento del código luego de efectuar cambios. La realización de alteraciones (previstas e imprevistas) en el código es una característica inherente de todas las metodologías ágiles. Pero si no se impone un mínimo orden a este ritmo de cambio tan vertiginoso, se corre riesgo de introducir errores que, al acumularse tiendan a volver las pruebas de software en un proceso mucho más caótico y tedioso.

Dado el contexto de la pandemia del coronavirus, los integrantes de este equipo estuvimos trabajando en el proyecto a distancia, con muy poco contacto interpersonal, y en numerosas oportunidades nos vimos obligados a alterar código de la autoría de alguno de nuestros

compañeros sin poder consultarles si nuestros cambios tendrían algún efecto secundario no deseado. En un escenario en el que no hubiésemos planteado pruebas unitarias, dichos cambios habrían conllevado largas horas destinadas a localizar errores que habrían podido ser identificados de forma mucho más temprana. Sin embargo, gracias a la aplicación de pruebas unitarias, no se realizaron cambios sin que dichos test avalen su correcto comportamiento.

Tal como lo indica [21] en el primero de sus 3 principios del testing de software: "Las pruebas sólo confirman la existencia de los errores que detectan, pero no pueden asegurar que no haya otros errores". La prueba unitaria facilita el testing y la localización temprana de errores, pero no la soluciona por completo. Siempre habrá casos de prueba no tenidos en cuenta que puedan ocasionar problemas, incluso cuando el testing unitario es profundo e incluya miles de casos. Por ese motivo, se acostumbra combinar esta técnica junto con otras que brinden una perspectiva integral de la robustez del sistema y permitan identificar aún más errores, que, una vez solucionados, den lugar a un software de calidad.

a. Testing en la aplicación Android

Para efectuar pruebas unitarias en este proyecto se utilizaron las librerías JUnit y Mockito. La primera proveyó las herramientas básicas para organizar las pruebas y verificar las aserciones, mientras que la segunda permitió aislar las pruebas de unidad por medio del uso de mocks que sustituyan instancias de otras clases ajenas a la testeada. Dichos mocks contaban con métodos especiales llamados "stubs" que se configuraban para retornar diferentes valores que permitan testear todos los flujos de ejecución de la clase testeada.

Un inconveniente que se presentó en este proceso fue la necesidad de adaptar el código del programa a una estructura que conviva con el test y provea funcionalidades para que el mismo se ejecute sin problemas. Es así que se tuvieron que eliminar casi todas las llamadas a métodos de clase (que Mockito no permite "stubbear"), agregar métodos únicamente visibles en entorno de testing y adaptar los constructores para que instancien más objetos que ofrezcan la funcionalidad de los métodos de clase. Si bien esto implicó ensuciar el código de la aplicación con funciones innecesarias, la robustez lograda por la disponibilidad de las pruebas unitarias probó ser una ventaja considerable que aceleró el avance del proyecto.

El testing unitario no se limitó únicamente a la lógica de negocios de la aplicación, sino que también investigamos tecnologías para hacer testing a fondo sobre las interfaces de usuario. De esta manera pudimos disponer de un panorama general del funcionamiento del sistema ante cualquier cambio en el código. Utilizamos la librería Espresso para llevar a cabo estas pruebas dado que la misma se integra a la perfección con el IDE Android Studio y posee ciertas herramientas sumamente útiles para el testing de interfaces. La librería permite acceder a los componentes de la interfaz, simular interacciones con los mismos (presionar botones, mover barras de desplazamiento, entre otras) y verificar su correcto comportamiento con aserciones.

La ejecución de tests de interfaz presentó ciertos inconvenientes que debimos enfrentar. Uno de ellos fue la dificultad que posee la librería para manejar ejecuciones en las que se debe esperar para obtener una respuesta. Por ejemplo, al buscar una reserva, podemos simular una

búsqueda ingresando los parámetros y presionando el botón “Buscar”, pero se debe esperar a que la búsqueda finalice para poder realizar las aserciones correspondientes. Espresso permite manejar la espera de recursos, pero debimos combinar esta solución con la utilización de numerosos “mocks” de Mockito para “stubbear” los métodos que devuelven información de la base de datos. De esta manera, la espera de la búsqueda se acortaba y siempre se traía el mismo set de datos de prueba, para luego realizar las aserciones en la interfaz, verificando que se impriman correctamente en pantalla. Otro de los inconvenientes que se presentaron en los tests en Espresso es la lentitud de los mismos, característica inherente de este tipo de pruebas dado que, para poder realizarse, requieren una compilación total de la aplicación y su ejecución en un dispositivo. Es así que siempre resultaba más conveniente ejecutar una batería de tests y resolver varios errores a tener que esperar varias compilaciones para corregir los bugs de a uno.

Estas implementaciones requirieron tiempo no sólo para comprender adecuadamente las tecnologías de testing, sino también para mantener actualizados los tests durante el proceso de desarrollo. Mantener los tests actualizados es una tarea muy demandante, pues no solo implica incorporar nuevos tests cuando se implementan funcionalidades, sino también alterar los tests existentes cuando el flujo de ejecución cambia. En reiteradas oportunidades durante el proyecto, se agregaron, eliminaron y refinaron numerosos tests para que representen de la mejor forma posible el funcionamiento esperado de las clases del sistema.

b. Testing en el Backend

El testing en el backend se realizó con librerías y recursos diferentes. El servicio de Reservas desarrollado en Go se puso a prueba con la librería de testing que el lenguaje provee, mientras que los servicios MercadoPago y de Usuarios (desarrollados en Express.js) requirieron el uso del paquete “Sinon” que provee la funcionalidad de “stubbing” y “spies”. Los “spies” actúan como “mocks” que almacenan información sobre las llamadas y argumentos que recibieron y los retornos que devolvieron. Sinon también provee aserciones personalizadas para sus “spies” y permite verificar que la interacción de la clase con el objeto espía haya sido la que se espera de ella.

Al igual que en Android, en el testing del backend también se debió adaptar el código para que conviva con las pruebas unitarias. Se debieron hacer numerosos cambios notorios como el desacoplamiento de las funciones del controlador y la incorporación de métodos que permitan sustituir objetos que se comunican con las APIs externas esenciales como Firebase y MercadoPago.

II. Pruebas de Integración.

A diferencia del testing unitario, las pruebas de integración no se hicieron de forma automatizada. En lugar de esto, cada vez que alguno de los integrantes realizaba un pull request, el resto del grupo debía compilar el proyecto para probar la nueva funcionalidad con casos de prueba variados.

Este enfoque funcionó muy bien porque no solo permitió que se hagan tests más confiables y objetivos, sino que también mantuvo a todo el grupo en pleno conocimiento de la evolución del código y sus potenciales falencias.

III. Pruebas del Sistema.

Para realizar las pruebas finales del sistema completo se construyó un conjunto de datos de prueba en la base de datos (Ver anexo I). Estos datos se usarían para probar la aplicación con los casos de prueba (Ver anexo II) que se definieron para evaluar el funcionamiento de la aplicación.

Las pruebas se hicieron una a la vez, en diferentes ejecuciones de la aplicación y en diferentes teléfonos celulares con diferentes resoluciones de pantalla. Los resultados pueden observarse en los Anexos III y IV. Afortunadamente las fallas detectadas en esta última etapa fueron prácticamente insignificantes. Muy probablemente esto fue producto de todo el esfuerzo puesto en las etapas anteriores del testing.

f) Seguridad

En esta sección se analizan los diferentes mecanismos de seguridad empelados tanto para la autenticación de usuarios como para la comunicación con la WebAPI y la base de datos. Además, se menciona la tecnología utilizada como plataforma de pagos y los parámetros de seguridad que se requieren para comunicarnos con la misma.

I. Inicio de sesión

Para la autenticación de usuario empleamos las funcionalidades provistas por la API de Google. Si bien esto requiere que el usuario posea una cuenta en dicho servicio, sabemos que nadie se vería imposibilitado de acceder a Rush, puesto que todos los celulares Android están asociados a una cuenta de Google.

Elegimos esta modalidad de inicio de sesión por varios motivos, el primordial es la facilidad que brinda utilizar un mecanismo de autenticación confiable ya implementado y listo para usar. Dicha ventaja es incomparable con lo engorroso que supondría desarrollar nuestro propio mecanismo de seguridad. Asimismo, la envergadura de la aplicación no amerita un desarrollo de seguridad personalizado, pues se requiere muy poca información del usuario, y la misma puede obtenerse de forma rápida y sencilla de Google, sin necesidad de completar formularios con datos que las APIs pueden obtener de forma automática.

II. Gestión de pagos

Siendo que es la primera vez que los tres integrantes de este grupo nos enfrentamos a la necesidad de desarrollar un sistema que requiera el cobro de un servicio, decidimos hacer uso

de la API de MercadoPago, que ya posee implementados todos los requerimientos de seguridad. Por medio del manejo de diversos tokens de las tarjetas e IDs de usuarios y autorizaciones, el sistema se desliga tanto del almacenamiento de los medios de pago como de la realización de los cobros.

Si bien la API empleada presenta numerosas falencias en su documentación y entorno de testing, al día de la fecha no han trascendido fallas masivas en su seguridad que pudiesen suponer algún riesgo mayor para nosotros o para los usuarios del sistema. Además, si algún día se presentase alguna contingencia que pusiese los fondos de los usuarios en peligro, MercadoPago es el desarrollador de casi la totalidad del código que maneja la seguridad de la plataforma de pagos y, por ende, es solidariamente responsable ante cualquier daño que algún usuario pudiese sufrir.

III. Autenticación en el backend

El backend es pasible de recibir llamadas a endpoints que puedan alterar los datos de viajes, pasajeros y reservas. Sin embargo, no todas esas llamadas pueden provenir de clientes legítimos. Resulta indispensable filtrar las llamadas realizadas desde orígenes conocidos de aquellas que pueden haber sido enviadas para vulnerar la integridad de los datos. Firebase propone para esto una alternativa bastante sencilla. Su API permite la generación de tokens temporales para la sesión de cada usuario, que pueden validarse en el backend para corroborar que la llamada que contiene el token proviene de un usuario legítimo [22].

El mecanismo de autenticación funciona de la siguiente forma, antes de llamar al backend, el cliente Android debe solicitar a la API de Firebase la generación de un token de la sesión. Dicho token se adjunta en la cabecera de la request que se envía al backend. Una vez recibida la consulta, el controlador correspondiente llama a otro método de la API de Firebase que valida el token recibido. Si el mismo se corresponde con una sesión activa, se procede a procesar la request, en el caso contrario, se rechaza la misma devolviendo una respuesta con código 401 (Unauthorized). Los tokens de Firebase no se almacenan ni se reutilizan porque solo son válidos por el plazo de una hora desde su generación. Cada vez que se desea realizar una nueva request, se debe solicitar a la API un nuevo token para que la validación cumpla su objetivo.

En el caso de las llamadas al Servicio Viaje, se realiza un paso más para garantizar la seguridad de los datos. Estos endpoints solo deberían poder ser accedidos por los usuarios de la interfaz de administración del sistema. Sin embargo, la API de Firebase solo permite validar sesiones de usuarios, sin distinguir qué permisos tiene cada uno. Es por esto que las llamadas al Servicio Viaje deben pasar por un filtro adicional que consulta en la base de datos si el usuario validado tiene permisos de administrador. De no poseer tales permisos, se rechaza la request de la misma manera que si la sesión no existiese.

IV. Control de acceso y reglas de la base de datos

Firebase permite el control de acceso a los diferentes documentos de la base de datos por medio del uso de reglas. Las mismas se definen en la base de datos con un lenguaje que permite especificar grupos de documentos a los que se le asignará un conjunto de permisos específicos. Pueden consultarse más detalles sobre la naturaleza de este mecanismo de seguridad en [23]. Las reglas se definieron de manera que sólo se autorice el acceso de acuerdo a las restricciones indicadas en la Tabla III.

Documentos	Accesos Autorizados
Pasajeros	● Solo el usuario puede leer y editar sus datos personales.
Reservas	● El usuario que creó la reserva y los administradores de la empresa que organiza el viaje reservado pueden leer y modificar la reserva. ● Solamente el usuario puede crear nuevas reservas para sí mismo.
Viajes	● Solamente los admins de una empresa pueden crear viajes. ● Solamente los admins de la empresa que creó el viaje pueden editarlo. ● Todos los usuarios pueden leer los viajes.
Empresas	● Solamente los admins de la empresa pueden editar los datos de la empresa. ● Todos los usuarios pueden leer las empresas.
Choferes	● Solamente los admins de la empresa pueden crear o editar los choferes. ● Todos los usuarios pueden leer los choferes.
Colectivos	● Solamente los admins de la empresa pueden crear o editar los colectivos. ● Todos los usuarios pueden leer los colectivos.
Configuraciones	● Todos los usuarios pueden leer las configuraciones globales.

Tabla III: Reglas de acceso definidas en la base de datos

Tal como lo indica [23], si se solicitan permisos para algún documento que no tenga una regla específica para tal fin, la operación será rechazada.

5. Gestión del proyecto

En esta sección se presentan las tareas realizadas en el marco de la gestión del proyecto. Se hace hincapié en la metodología empleada para la revisión de código y la política de gestión de cambios y configuraciones. Adicionalmente, se comentan las decisiones que se tomaron para mitigar el impacto de algunos riesgos que se materializaron durante el transcurso del proyecto.

a) Política de gestión de cambios y configuraciones

La gestión de cambios y configuraciones se llevó a cabo empleando la Plataforma GitHub en la que almacenamos 4 proyectos independientes: La aplicación Android, la WebAPI del backend, la interfaz de administración y la interfaz de billetera virtual. La gestión de cambios y configuraciones se organizó de la siguiente forma:

- Por cada funcionalidad que se proponía desarrollar, se creaba una issue (o bug si era un error que debía corregirse) indicando claramente la funcionalidad a desarrollar, su contexto y diferentes alternativas para realizar su implementación.
- Por cada nueva funcionalidad que se desarrollaba, se creaba una nueva rama. Una vez finalizado el desarrollo que incumbe a dicha rama, se abría un "pull request" y se instaba a los restantes integrantes del grupo a revisar el código de acuerdo a los parámetros indicados en la sección b.

Adicionalmente a estas prácticas triviales, el equipo incorporó las siguientes ideas para imponer aún más orden en la gestión de cambios.

I. Pull Requests apilados

Una vez iniciado el proyecto, no tardaron en aparecer problemas en la gestión de los Pull request. Los tres integrantes del grupo trabajamos independientemente en porciones de código separadas y generamos numerosos pull request con miles de líneas de código por verificar. Naturalmente las revisiones de código se prolongaban demasiado y el desarrollo continuaba sin realizar los correspondientes merge, generando nuevos pull request con líneas de código adicionales a las aún no revisadas. Esta dificultad está muy presente en distintos proyectos de usuarios poco experimentados en GitHub, y algunos autores como [24] han documentado alternativas para resolverla. La situación pudo controlarse de forma temprana cuando se propuso una nueva política de manejo de los pull request que consista en ir apilando diferentes ramas de acuerdo a funcionalidades atómicas que se desarrollen y desapilándolas con pull requests pequeños con pocas líneas de código por revisar en cada uno. De esta manera, se evita que los códigos a revisar sean demasiado extensos, situación propicia para que, entre tantas líneas de código, se pasen por alto errores. Así, las revisiones pueden hacerse más a conciencia, concentrándose en una única funcionalidad.

Como resultado, se multiplicó la cantidad de ramas y se destinó más tiempo en la organización de los pull request. Sin embargo, la revisión del código se volvió mucho más eficaz, cómoda y manejable, permitiendo localizar de forma mucho más temprana errores e ineficiencias que pudiesen haber puesto en jaque el proyecto.

II. Claridad y trazabilidad del control de cambios - Plantillas para issues y PR

Otra dificultad emergente en la gestión de cambios es la variabilidad en la información provista en los detalles de cada issue y pull request. No todos los integrantes del grupo consideraban importante redactar los mismos detalles en cada issue, lo que generaba ciertas inconsistencias y ambigüedades. Los pull request no siempre indican claramente los cambios propuestos ni hacían referencia a la issue que resolvían, motivo por el que la trazabilidad de los cambios se vio comprometida. Afortunadamente ninguno de estos inconvenientes supuso un retraso en las primeras etapas del desarrollo, pero para evitar que ello sucediese luego, se comenzaron a evaluar ciertas alternativas que permitan unificar criterios. GitHub promueve el uso de plantillas para estandarizar el contenido de los issues y pull requests [25]. Es así que se configuraron en el proyecto plantillas para la redacción de cada issue y pull request. Cada una indicaba una serie de datos obligatorios y opcionales que debían incluirse antes de publicar cualquier elemento de gestión de cambios.

En el caso de las issues, se debía indicar el contexto de la nueva funcionalidad, describir el requerimiento lo más detalladamente posible y proponer alternativas para la implementación del mismo. La plantilla que definimos para esto puede observarse en la Figura 23.

```
## Título del Issue
<!--- Usar un título que sirva como resumen general de la nueva
funcionalidad -->

## Contexto
<!--- Describir de forma clara y concisa por qué se necesita esta
funcionalidad, o el problema que soluciona -->

## Descripción general
<!--- Describir la solución al problema o la funcionalidad en sí, sin
entrar en muchos detalles técnicos -->

## Implementación
<!--- (Opcional) Describir la forma de implementar la solución o
funcionalidad de forma clara -->

## Alternativas
<!--- (Opcional) Describir brevemente alternativas descartadas para
solucionar este problema/implementar esta funcionalidad -->
1.
2.
3.
```

Figura 23: Plantilla para la creación de una Issue

En el caso de los pull request, se debían indicar todos los cambios que se efectuaron al código de la rama a la que se pretende aplicarle merge. Era obligatorio indicar una referencia a las issues que está resolviendo este pull request y mencionar que cambios fueron realizados en los testing unitarios. Finalmente, si el pull request incluía cambios sobre alguna interfaz gráfica, era necesario documentarlos con capturas de pantalla de las modificaciones y una breve mención de los componentes que fueron alterados. El template que impone esta estructura puede observarse en la Figura 24.

```
## Título del Pull request
<!--- Usar un título que sirva como resumen de los cambios propuestos en
este PR -->

## Cambios que se proponen
<!--- Listar todos los cambios involucrados en este PR -->
-
-
-

## Issues relacionadas
<!--- Etiquetar todas las issues relacionadas, usando las palabras claves
"Fixes" o "Resolves" si corresponde -->

## Testing
<!--- Describir cómo han sido testeados los cambios involucrados en este
PR -->

## Screenshots
<!--- (Opcional) Si en este PR hay cambios relacionados con UI/UX, incluir
capturas de pantalla ilustrando los mismos -->
```

Figura 24: Plantilla para la creación de una Issue

La incorporación de las nuevas plantillas permitió disponer de información suficiente y consistente en cada pull request y en cada issue. El tiempo insumido para completar tales datos fue mayor, pero no ocasionó atrasos considerables. Gracias a esto, el proyecto pudo avanzar de forma más ordenada y se empezó a contar con el valor agregado de poder establecer una clara trazabilidad entre los cambios en el código (pull request) y los requerimientos (issues).

b) Revisión del código

Tal como indicamos en la sección 4a, en un principio las revisiones de código se realizaban en el marco de reuniones presenciales del grupo, en las que se analizaban los pull requests y se comentaban posibles errores o ineficiencias con sus autores. Con la llegada de la pandemia, no pudimos realizar más estas reuniones, por lo que cada uno de los integrantes del equipo debió encontrar el momento que le resulte más conveniente para revisar el código de sus compañeros. Si surgía alguna duda o comentario para realizar, se lo documentaba en el pull request y, de no existir errores, se lo aprobaba. Hasta no contar con la aprobación de la totalidad del grupo no se realizaba ningún merge.

Sin embargo, dado que no todos prestamos atención a los mismos detalles del código, decidimos crear una checklist que clarifique en qué puntos hay que hacer hincapié para encontrar errores comunes y poner atención sobre los mismos. Cada vez que revisamos una porción de código ajeno debíamos responder las siguientes preguntas de manera de asegurarnos que la integración de la nueva funcionalidad no ocasione problemas inesperados. Los puntos definidos fueron los siguientes:

1) ¿El código de la rama a mergear compila?
2) ¿Se está intentando mergear sobre la rama correcta?
3) ¿La funcionalidad a mergear cumple satisfactoriamente el requerimiento?
4) ¿Los test unitarios abarcan la totalidad de posibles ejecuciones?
5) ¿La funcionalidad desarrollada respeta los modelos de análisis y diseño?
6) ¿Existe alguna otra funcionalidad que pudo ser afectada por este cambio?
7) ¿Existe algún test unitario de otra funcionalidad que pudo ser afectado por este cambio?
8) ¿La solución maneja correctamente las excepciones?
9) ¿La solución hace un manejo eficiente de los recursos?
10) ¿La solución tiene problemas de concurrencia?
11) ¿Se están aplicando correctamente todos los mecanismos de seguridad?

c) Gestión de riesgos

Al inicio del proyecto contábamos con una lista de riesgos priorizada que nos sirvió de guía para monitorizar las potenciales amenazas que podrían poner en jaque el proyecto. Dicha lista puede observarse en el Anexo V. Al término de cada sprint revisamos la misma para verificar si se accionó alguno de los disparadores y decidir si correspondía o no ejecutar algún plan de contingencia.

A lo largo del desarrollo del proyecto se presentaron desafíos causados por riesgos que sí estábamos gestionando, acompañados de riesgos que nunca fueron previstos y cuyos planes de contingencia debieron ejecutarse de forma reactiva.

I. Riesgos planificados y gestionados

a. Desconocimiento de las tecnologías a emplear

Desde el primer momento, se intentaron seleccionar las tecnologías más conocidas por los integrantes del equipo, de manera que no se requiera destinar tanto tiempo para capacitarnos. Sin embargo, dos tecnologías, respecto de las cuales teníamos poco o nulo conocimiento, supusieron desde el principio un desafío. Estas fueron la plataforma de pagos y el testing unitario.

Si bien ya tuvimos alguna experiencia breve con el testing unitario durante el cursado de la carrera, nunca nos propusimos aplicarlo en la casi totalidad de las clases de un proyecto. Mantener una batería de pruebas confiable y robusta es un desafío que no solo requería que aprendamos las tecnologías necesarias, sino que también, adaptemos nuestras prácticas al momento de codificar para que no obstruyan la ejecución de los tests [26]. La tarea de tornó aún mas interesante cuando nos propusimos aplicar testing unitario a las interfaces graficas con Espresso. Esta tecnología se asemejaba considerablemente a las que ya empelamos en otros proyectos para hacer pruebas, pero adoptaba un enfoque diferente al centrarse únicamente en la verificación del comportamiento de componentes gráficos.

El completo desconocimiento con el que iniciamos el proyecto nos obligó a adelantar el inicio de las capacitaciones en testing. Esto fue muy beneficioso porque debimos alterar gran parte del código para poder testearlo, y sólo debimos hacerlo con las pocas clases que se habían desarrollado al principio. Al momento de continuar el desarrollo, las pautas de codificación ya estaban establecidas de manera que el testing no tenga que cambiar nada luego. Sin embargo, hacer testing desde una etapa tan temprana tiene sus desventajas. Mantener actualizada la batería de pruebas ante cada mínimo cambio en la implementación de la clase es un desafío que requiere mucho tiempo y desalienta la incorporación de mejoras al código.

Por otro lado, la implementación de la plataforma de pagos supuso otro desafío considerable. Nunca habíamos interactuado con un recurso tan crítico como el dinero de terceros, y es por eso que desde el principio decidimos proteger el cronograma del proyecto asignando una elevada cantidad de horas a la implementación de esta funcionalidad. Al igual que con el testing unitario, empezamos a investigar la plataforma mucho antes de comenzar a codificar la solución. De esta manera pudimos experimentar con las tecnologías hasta que finalmente nos sentimos lo suficientemente seguros como para iniciar el desarrollo de la billetera virtual y la pantalla de confirmación de pagos.

Tanto para la plataforma de pagos como para el testing unitario, la estrategia de adelantar la capacitación nos permitió ir mejorando nuestras habilidades con las tecnologías a usar, lo que nos dio más confianza al momento de encarar las tareas de desarrollo que mayor incertidumbre nos generaban y evitar retrasos innecesarios en el proyecto.

b. Aumento desproporcionado de los gastos de servicios en la nube

A pesar de conocer la existencia de este riesgo, debimos reaccionar ante el mismo sin un plan de contingencia. Al momento de planificar, previmos una probabilidad de ocurrencia muy baja que nos motivo a tolerar el riesgo. Sin embargo, al momento de poner en marcha el backend en Google Cloud, notamos una dificultad que amenazaba con ocasionarnos costos elevados no previstos. El consumo de memoria de la aplicación era excesivamente alto. Algo que resultaba totalmente incomprensible si se tiene en cuenta que algunos servicios casi no tenían lógica de negocios. No consideramos admisible que haya servicios tan básicos cuyo consumo de memoria exceda los límites gratuitos provistos por la plataforma, por lo que decidimos investigar sobre las potenciales causas, y descubrimos que el desperdicio en

memoria de los servicios Spring modernos oscila entre el 30% y el 90% incluso en proyectos sumamente sencillos. Algunos autores como Lakshmanan en [27] afirman que el motivo de tal carga sobre la memoria es producto de malas prácticas de código que hacen ineficiente el manejo de muchas peticiones. Sin embargo, nuestro sistema probó que Lakshmanan puede estar equivocado, pues la mera puesta en marcha del servicio de Pasajeros, uno de los más sencillos del sistema, excedió el límite gratuito de 256 MB de RAM, aun cuando todavía no se habían recibido peticiones. Otros usuarios que desarrollan proyectos sencillos también ven inflado el consumo de memoria de sus aplicaciones debido a este framework, al que acusan de "bloatware" en numerosos documentos e hilos de redes sociales [28] [29].

Es así que se optó por volver a desarrollar los backend en Spring utilizando otros frameworks: Express.js y Go. Se replicó exactamente la misma lógica de negocios que se había desarrollado en el servicio de Spring y la performance mejoró de forma notable. En el caso del servicio de Pasajeros, que antes había ocasionado problemas, se logró reducir a más de la mitad el consumo de memoria, y cerca de 10 veces el tiempo de arranque. A pesar del retrabajo requerido, se optó por la migración total de los restantes servicios a otros framework, de manera que siempre que se pueda, se evite pagar por excesos innecesarios en el consumo de memoria. Hubo un atraso considerable en el proyecto a causa de esta decisión, pero el ahorro logrado en los consumos dolarizados que suponen estas plataformas sigue siendo sumamente significativo.

c. Otros riesgos planificados

Varios de los otros riesgos que analizamos al momento de la planificación se materializaron, pero no tuvieron impactos significativos. Durante el transcurso del proyecto, los 3 integrantes incrementaron su carga horaria laboral, reduciendo considerablemente la cantidad de tiempo disponible para el proyecto y otras tareas. El cumplimiento de las horas semanales destinadas al proyecto se vio comprometido, pero otros acontecimientos ocasionaron retrasos mucho mayores que estos. Algunas tareas fueron subestimadas, pero la mayor parte de los retrasos se dio por factores ajenos a las estimaciones realizadas. La interfaz gráfica también podía ocasionar retrasos teniendo en cuenta que una mala experiencia de usuario no solo puede requerir cambios en los componentes gráficos, sino también en la lógica de negocios que sirve a esas interfaces. La estrategia adoptada para evitar tales problemas fue construir las interfaces siguiendo los consejos de algunos referentes que se dedican al desarrollo de experiencias de usuario.

Al mismo tiempo, seguimos interesados en ofrecerles la plataforma que desarrollamos a las empresas de transporte. El riesgo de perder el interés de los potenciales clientes no compromete la viabilidad técnica del proyecto, pero sí su podría poner en jaque su futura rentabilidad, de manera que correspondía que también apliquemos una estrategia para mitigar este riesgo. Luego de una exhaustiva investigación, logramos localizar otras empresas de colectivos que tienen funcionamiento similar a las de los colectivos que transitan entre Santa Fe y Paraná. De esta manera, acrecentamos el universo de potenciales clientes y tenemos más alternativas a la hora de ofrecer nuestro producto terminado.

II. Riesgos imprevistos

La gestión de riesgos hace el mayor esfuerzo posible para anticiparse a un conjunto de amenazas que no siempre pueden identificarse con facilidad. En algunos casos es imposible prever determinados riesgos, especialmente cuando su probabilidad de ocurrencia es insignificante. En esta sección analizaremos dos de los riesgos que emergieron durante el desarrollo del proyecto, pero que no fueron previstos en el plan.

a. Pandemia de CoViD-19

Es innegable que la pandemia mundial de CoViD-19 tomó por sorpresa a toda la comunidad internacional, convirtiéndose rápidamente en una amenaza para todo tipo de proyectos que requieran algún tipo de contacto presencial. El desarrollo de Rush se vio afectado por esta contingencia debido a que nuestra metodología de trabajo preveía reuniones presenciales periódicas. Sustituirlas por reuniones virtuales implicó una merma considerable en la productividad, que rápidamente se hizo notar en los atrasos que se produjeron en el cronograma del proyecto.

Muchas dificultades se presentaron al intentar continuar el proyecto durante las medidas de distanciamiento social. Entre ellas estuvieron la imposibilidad de mantener reuniones prolongadas, coincidir en nuestros tiempos libres y mantener una comunicación fluida cuando había dificultades con el servicio de internet. La estrategia que aplicamos consistió en mantener reuniones mucho mas breves, pero muy frecuentes. De esta manera, todo el equipo podía tener un panorama del avance del proyecto que, aunque sufrió notables atrasos por esta contingencia, nunca se puso en peligro su continuidad, pues cada integrante siempre estuvo pendiente de los avances del proyecto.

b. Necesidad de cambios significativos en la arquitectura del sistema

Inicialmente se había planificado que la aplicación Android se conecte en forma directa a la base de datos en Firebase. En algunos proyectos realizados con antelación se encaró el desarrollo con esta arquitectura y se lograba cumplir los requerimientos. Sin embargo, algunos de los integrantes de este proyecto, participaron previamente en el desarrollo de Prode of Thrones [30]. Dicha aplicación fue desarrollada con mucha prisa para lanzarse a la par de una serie televisiva. En consecuencia, no tardó en aparecer la necesidad de realizar modificaciones a la estructura de la base de datos mientras la misma estaba en producción. Tales cambios debían estar acompañados de una actualización de la aplicación que debía distribuirse en forma inmediata por Google Play, pues de lo contrario se corría riesgo de producirse un mal funcionamiento en las versiones anteriores que esperaban una estructura de datos diferente. Si bien Rush no se desarrolló con la misma urgencia que Prode of Thrones, se empezaron a considerar arquitecturas alternativas que no requieran una actualización masiva inmediata cuando hay alteraciones en la base de datos.

La propuesta que se terminó llevando a cabo fue la construcción de una WebAPI que funcione como intermediaria entre la base de datos y los clientes Android. De esta manera, cuando sucedan cambios en la estructura de la base de datos, se requerirá una única actualización de la WebAPI que manipule dichos cambios adecuadamente, sin necesidad de actualizar los clientes de cada terminal que tenga la aplicación.

La nueva arquitectura propuesta para la aplicación supuso un replanteo de gran parte de la estructura del proyecto, así como también, implicaba un retraso significativo de la finalización del mismo, puesto que se estaba incorporando una capa adicional para el manejo de datos no prevista en el plan inicial. Sin embargo, no dudamos en embarcarnos en la iniciativa puesto que, indudablemente el mayor tiempo de desarrollo impactaría positivamente en la calidad del producto final.

d) Otras incidencias y sucesos en el desarrollo

Durante el desarrollo del proyecto sucedieron un conjunto de acontecimientos que impactaron significativamente en el ritmo de avance del mismo. Experiencias de proyectos previos, acontecimientos mundiales como la pandemia del CoViD-19, así como también numerosas propuestas de modificaciones a la arquitectura de la aplicación ocasionaron retrasos que, aunque implicaron una mejora sustancial en la calidad del desarrollo, devinieron en una postergación de la fecha prevista de finalización del proyecto.

I. Fallas funcionales y documentación incompleta de la API de MercadoPago

Si bien MercadoPago provee una API de desarrollo pensada para los medios de pago argentinos y es relativamente sencilla de implementar, resulta también muy cuestionada por la pobre calidad de su documentación. Numerosas funcionalidades avanzadas que empleamos en este proyecto poseen documentación pobre, desactualizada o simplemente inexistente. Al momento de redactar estas líneas todavía no existe documentación oficial sobre la captura de autorizaciones en node.js, funcionalidad crucial para la gestión de cobros del sistema.

Paralelamente, algunas funcionalidades del API sencillamente no reciben el mantenimiento adecuado, especialmente en los entornos de testing. La captura parcial de fondos de una autorización nunca funcionó durante las pruebas de nuestro sistema y resultó imposible evaluar su eficacia hasta que el mismo no pasó a producción.

II. Manejo de información estática en memoria.

Si bien mucha de la información que maneja el sistema se obtiene a demanda de la base de datos, existe cierto conjunto de datos cuya variabilidad es tan infrecuente que se consideró la posibilidad de almacenarlas en memoria para mejorar significativamente la performance. Es así que la información referida al costo del servicio de la aplicación, la información sobre las paradas y demás configuraciones que no varían con tanta frecuencia se obtienen de la base de

datos una única vez al iniciar la aplicación. Luego, dichos datos se obtienen de la memoria de forma sumamente expedita.

Lamentablemente, la presunción que dicha información permanezca en memoria durante toda la ejecución de la aplicación no es más que un error ingenuo. El teléfono puede poner a Rush en segundo plano, y ante una eventual falta de memoria para otras aplicaciones, se podría perder información esencial para la ejecución. Es así que, para mantener la mejora de performance obtenida y no comprometer la viabilidad de la ejecución del sistema, se incluyó en cada sentencia que requiera datos de memoria una instrucción que verifique si dichos datos están disponibles y, ante una ausencia de los mismos, los obtenga de manera asincrónica.

Si bien el código mutó notablemente a causa de estas contingencias de baja probabilidad, los cambios permitieron asegurar un sistema más robusto y preparado para muchas de las tantas eventualidades que pudieran atentar contra la experiencia de usuario en los equipos más modestos.

6. Conclusiones

Durante las diversas etapas de este proyecto pudimos, a pesar de las contingencias de la pandemia, llevar adelante el desarrollo de un producto de software que cumple las expectativas que plasmamos en el plan de proyecto inicialmente presentado. A lo largo de este camino se presentaron numerosos obstáculos a sortear, varios de los cuales nunca habían sido previstos y tuvieron un impacto significativo en el cronograma. Sin embargo, las diversas adaptaciones que hicimos a la metodología de trabajo, permitieron finalmente alcanzar un resultado de calidad.

Las reuniones presenciales del grupo eran uno de los ejes fundamentales de la metodología, pero durante la mayor parte del desarrollo nos vimos imposibilitados de realizarlas a causa de las medidas de distanciamiento social. Es así que la metodología tuvo que modificarse, llevando a cabo las reuniones de forma virtual, en pos de poder continuar el proyecto en un contexto inusual y completamente imprevisto. Sin embargo, y aún con todos los retrasos que la pandemia ocasionó, el grupo pudo adaptarse a las circunstancias, permitiendo que el proyecto avance y alcance su finalización en un plazo considerable respecto al que inicialmente nos habíamos propuesto.

Asimismo, se presentaron otras de las dificultades en el transcurso del proyecto, como lo fueron las modificaciones tardías en los diagramas fundamentales, la falta de documentación de la API de pagos, el elevado consumo de memoria del backend desarrollado en Spring, entre otros. Cada inconveniente supuso un desafío al ingenio de todos los integrantes del grupo que debimos abocarnos a proponer alternativas para destrabar el avance del proyecto.

La capacidad de sortear obstáculos que comúnmente se presentan en los proyectos de software, así como también la posibilidad de adaptarse a contextos desfavorables imposibles de prever son algunas de las enseñanzas más significativas que nos deja este proyecto. Si bien ya habíamos tenido que enfrentar estas dificultades en otras circunstancias, nunca nos vimos envueltos en un proyecto de una magnitud comparable, ni en un contexto de tanta incertidumbre. La experiencia que tuvimos fue sumamente positiva y enriquecedora y no dudamos que podremos muy pronto sacar provecho de todo lo que aprendimos en esta última etapa de nuestra carrera de grado.

El producto de software desarrollado fue nuestra principal motivación desde el principio. Rush puede ser de gran utilidad no solo para la industria del transporte, sino también para la gran masa de usuarios que día a día transitan entre Santa Fe y Paraná y hace años que están alzando su voz para que se les brinde un mejor servicio. Rush es una respuesta a estas reiteradas demandas. Atendimos una problemática compleja con una solución osada, pero sencilla y la misma está disponible para cualquier empresa que desee incorporarla. El proyecto terminó, pero la mejora de este sistema de transporte continúa siendo una deuda pendiente.

7. Extensibilidad

Si bien la aplicación estuvo concebida inicialmente para atender la problemática específica del transporte interurbano entre las ciudades de Santa Fe y Paraná, el diseño de la solución se pensó para permitir su utilización en muchos otros servicios de transporte similares. Si bien son pocos los servicios en el país que presentan características similares a las de los colectivos que transitan entre Santa Fe y Paraná, existen otros trayectos que también conectan dos provincias, no permiten reservas y se abonan con SUBE como es el caso de Corrientes-Resistencia.

Es cierto que, en el contexto de la pandemia por el coronavirus, estos servicios se hallan suspendidos y no hay indicios de potenciales fechas de reanudación de los mismos [31] [32]. Incluso ciertos funcionarios han declarado que planean posponer el reinicio de estas actividades hasta que estén dadas las condiciones de distanciamiento necesarias [33]. Es obvio que la existencia de una larga cola de espera para abordar un colectivo constituye un peligro para la salud de quienes esperan, más aún cuando es imposible mantener un control sobre el tamaño de la cola, siendo que todos compiten por conseguir un lugar en el próximo vehículo.

Las empresas podrían sacar mucho provecho de Rush para reiniciar sus operaciones de forma más temprana, sin comprometer la salud de sus pasajeros, al permitirles reservar su pasaje desde la comodidad de su casa y abordar el colectivo sin exponerse a un potencial contagio en la espera. Una vez que la pandemia haya sido superada y con Rush implantado, no pensamos que nadie crea conveniente volver al antiguo sistema de espera en cola.

El diseño de la aplicación también se pensó para que, con ciertas modificaciones, pueda adaptarse a la venta pasajes y no solo reserva de asientos. Teniendo información de los precios y el sistema de pagos implementado, basta con simplemente distinguir cuáles servicios ofrecen reserva y cuales sólo venta, así como también cuáles reservas se deben cobrar de forma completa y cuales deben permanecer autorizadas hasta que se usen o se multen.

También, la funcionalidad de autorizaciones de MercadoPago sólo funciona para hacer reservas en el corto plazo. Por política de la plataforma, toda autorización que haya pasado más de una semana sin capturarse, se cancela automáticamente. Esto restringe de forma muy significativa el horizonte de tiempo para el cual se pueden hacer reservas, lo que sin dudas podría mejorarse con una implementación de otra API que se adapte mejor a los requerimientos de un servicio de transporte que usualmente permite la venta de pasajes con hasta 6 meses de anticipación.

Son muchas las funcionalidades que se podrían adaptar para incrementar el alcance de esta aplicación que, si bien nació como solución a un problema muy específico, posee mucho potencial para instalarse y crecer en el sistema de transporte.

8. Referencias

[1] *"Terminal de ómnibus: largas colas y hasta dos horas de espera para viajar a Paraná en el finde extendido"* (2019) - Nota de Aire de Santa Fe 12/10/2019, Disponible en: https://bit.ly/2McXFRC - Consultado el 14/10/2019

[2] *"Volvieron las largas colas para ir a Santa Fe"* (2019) - Nota de El Diario de Paraná 14/4/2019, Disponible en: https://bit.ly/2IRe24f - Consultado el 14/10/2019

[3] *"Continúan las largas colas para viajar entre Paraná y Santa Fe"* (2019) Audio de Radio Nacional 1/4/2019, Disponible en: https://bit.ly/2VIqbO1 - Consultado el 14/10/2019

[4] *"Información sobre el aislamiento social, preventivo y obligatorio"* (2020) Ministerio de Salud de la República Argentina, Disponible en: https://bit.ly/3bC4FSm - Consultado el 26/04/2020

[5] *"Q&A on coronavirus"* (2020) Reporte de la Organización Mundial de la Salud, Abril 2020, Disponible en: https://bit.ly/3cL80i0 - Consultado el 26/04/2020

[6] Beasley J.E. (2004) "Queueing Theory" en "Operations Research Notes - Imperial College" Disponible en: https://bit.ly/29DTTQP - Consultado el 08/10/2019

[7] Lin A. (2018) "An Introduction to Queuing Theory: The mathematical study of waiting in line" - Disponible en: https://bit.ly/2MBYv9N - Consultado el 08/10/2019

[8] Hillier & Lieberman (2017) "Introducción a la investigación de operaciones", Sexta edición.

[9] Administración Federal de Ingresos Públicos (2020) *"Categorías del Monotributo"* - Disponible en: https://bit.ly/33Uwwe6 - Consultado el 21/09/2020

[10] Administración Provincial Impositiva Santa Fe (2020) *"Código Fiscal - Impuesto sobre los Ingresos Brutos"*, Cap. II - Disponible en: https://bit.ly/3hUS4Mm - Consultado el 21/09/2020

[11] Administradora Tributaria de Entre Ríos (2018) *"Código Fiscal - Ley Impositiva"* Disponible en: https://bit.ly/3hL6T4c - Consultado el 21/09/2020

[12] Honorable Concejo Deliberante de la Ciudad de Paraná (2019) *"Ordenanza Nº 9893"* - Disponible en: https://bit.ly/35V8m5J - Consultado el 21/09/2020

[13] Ministerio de Trabajo, Empleo y Seguridad Social (2020) *"Resolución 4/2020"*, Consejo Nacional del Empleo, la Productividad y el Salario mínimo, vital y móvil - Disponible en: https://bit.ly/2IMg2O0 - Consultado el 21/10/2020

[14] Ambler S. (2002) *"Agile Modeling: Effective practices"* Wiley Computer Publishing

[15] Ambler S., Rahn A. (2002) "Panfleto Introductorio sobre el Modelado Ágil" Ronin International, Disponible en: https://bit.ly/2N57j9w - Consultado el 01/11/2019

[16] Pressman R. (2010) *"Software Engineering: A Practitioner's approach, 7th edition"* McGraw Hill

[17] Ambler S. (2002) *"Panfleto del Modelado Ágil"*, Disponible en https://bit.ly/2Y8QCQo - Consultado el 26/04/2020

[18] Fowler A. (2012) *"10 Advantages of NoSQL over RDBMS",* Disponible en https://bit.ly/2zphy4g - Consultado el 24/05/2020

[19] Vera H., Boaventura W., Holanda M., Guimaraes V., Hondo F. (2015) *"Data Modeling for NoSQL Document-Oriented Databases"* Proceedings of the 2nd Annual International Symposium on Information Management and Big Data - SIMBig

[20] Nwamba C. (2018) *"Node.js Cron Jobs By Examples"*, Disponible en: https://bit.ly/2PMxv9w - Consultado el 12/08/2020

[21] Cagley T. (2014) *"Testing Principles Part 1: This is not Pokémon"* Software Process and Measurement Blog - Disponible en: https://bit.ly/3g98Sjf - Consultado el 22/05/2020

[22] *"Verifica tokens de ID"* Documentación de Firebase - Disponible en: https://bit.ly/3kxeIMx - Consultado el 19/09/2020

[23] *"Lenguaje de las reglas de seguridad"* Documentación de Firebase - Disponible en: https://bit.ly/35Refko - Consultado el 19/09/2020

[24] Koonce G. (2016) *"Stacked Pull Requests: Keeping GitHub Diffs Small"*, Disponible en https://bit.ly/2zAIDBh - Consultado el 29/04/2020

[25] *"Utilizar plantillas para promover informes de problemas y solicitudes de extracción útiles"*, Disponible en https://bit.ly/3f0ySNf - Consultado el 30/04/2020

[26] Kolawa A. y Huizinga D. (2007). *"Automated Defect Prevention: Best Practices in Software Management"*. Wiley-IEEE Computer Society Press. p. 75. ISBN 978-0-470-04212-0.

[27] Lakshmanan R. (2019) "Memory wasted by spring boot applications", SAP Community blogs, Technical Articles, Disponible en: https://bit.ly/3058nkz - Consultado el 03/06/2020

[28] Light Platform (2017) "Spring is bloated", Light Platform Documentation Disponible en: https://bit.ly/2XVDSeh - Consultado el 03/06/2020

[29] User vishnu_gupt (2016) "Why are Java Web Apps so much bloated?", Java News Community in Reddit Disponible en: https://bit.ly/3eZmOLp - Consultado el 03/06/2020

[30] Lopez L. (2019) *"Prode of Thrones - La app para hacer predicciones sobre la última temporada de Game of Thrones!"* Hilo de Reddit disponible en https://bit.ly/2KGw75M - Consultado el 26/04/2020

[31] Oviedo M. (2020) *"No habrá colectivos entre Paraná y Santa Fe hasta el miércoles 25"* Nota de Diario UNO Entre Ríos - Disponible en: https://bit.ly/3awhpdq - Consultado el 17/08/2020

[32] (2018) *"Solicitarán mejoras para el servicio del Chaco-Corrientes"* Nota en diario El Litoral de Corrientes - Disponible en: https://bit.ly/314TTkU - Consultado el 17/08/2020

[33] Ojeda F. (2020) *"El transporte Chaco – Corrientes será uno de los últimos servicios que autorizará el gobierno correntino"* Nota en diario Chaco Día por Día - Disponible en: https://bit.ly/31XEjqw - Consultado el 17/08/2020

9. Anexos

I. Datos de prueba

La prueba del sistema que aquí se presenta fue llevada a cabo en la fecha 13/09/2020 a las 18 hs. Es por eso que se crearon datos de prueba de viajes para verificar que las condiciones de fecha están funcionando correctamente

a. Colección de Empresas

Empresas		
Nombre	**Campo**	**Contenido**
ERSA - Fluviales	**Logo**	
	Política de Cancelación	**72 hs.:** 10% **48 hs.**: 20% **24 hs.**: 30%
	Choferes	José López
		Eduardo Domínguez
	Colectivos	**Patente:** OQA696 **Capacidad:** 55
	Personal	Federico Hauque
		Tomás Fleitas
ETACER	**Logo**	ETACER
	Política de Cancelación	**96 hs.:** 10% **72 hs.**: 30% **48 hs.**: 50%
	Choferes	José González
	Colectivos	**Patente:** AA733BJ **Capacidad:** 88
		Patente: AB655ZN **Capacidad:** 60

	Personal	Federico Hauque
		Tomás Fleitas
Flecha Bus	**Logo**	
	Política de Cancelación	**72 hs.:** 10% **48 hs.**: 20% **24 hs.**: 30%
	Choferes	Andrés Segovia
		José Roncaglia
	Colectivos	**Patente:** AC555BJ **Capacidad:** 77
	Personal	Federico Hauque
		Tomás Fleitas
Rápido Tata	**Logo**	
	Política de Cancelación	**96 hs.:** 10% **72 hs.**: 30% **48 hs.**: 50%
	Choferes	Andrés Martínez
	Colectivos	**Patente:** AC156FJ **Capacidad:** 80
	Personal	Federico Hauque
		Tomás Fleitas

b. Colección de Configuraciones

Configuraciones		
Documento	**Campo**	**Contenido**

Banderas	**costoServicioRush**	50 $
	diasParaMultar	2 días
	horasParaCancelar	48 hs.
	minutosParaCancelar	120 min.
Paradas	BuenosAires	**Terminal:** Retiro **Ubicación:** 34° 35′ S - 58° 22′ O
	Chajarí	**Terminal:** Justo José de Urquiza **Ubicación:** 30° 46′ S - 57° 59’ O
	Federal	**Terminal:** Terminal Federal **Ubicación:** 30° 57′ S - 58° 48’ O
	Paraná	**Terminal:** Francisco Ramírez **Ubicación:** 31° 44′ S - 60° 31’ O
	Rafaela	**Terminal:** Nueva Terminal Rafaela **Ubicación:** 31° 16′ S - 61° 29’ O
	Rosario	**Terminal:** Mariano Moreno **Ubicación:** 32° 57′ S - 60° 38’ O
	SantaFe	**Terminal:** Manuel Belgrano **Ubicación:** 31° 38′ S - 60° 42’ O

c. Colección de Viajes

Viajes		
Documento	**Campo**	**Contenido**
viaje_1	**destinoFinal**	SantaFe
	origenInicial	Chajarí
	estado	ACTIVO
	colectivo	Pat: AC 156 FJ
	empresa	Rápido Tata
	chofer	null
	reservas	Lista Vacía

	trayectos	**Chajarí > Federal** **asientosLibres:** 30 **llegadaDestino:** 08/12/2020 - 15:40 **precio:** 135,00 **precioSube:** 98.36 **salidaOrigen:** 08/12/2020 - 14:00 **Chajarí > Paraná** **asientosLibres:** 9 **llegadaDestino:** 08/12/2020 - 18:50 **precio:** 284,99 **precioSube: 1**90.36 **salidaOrigen:** 08/12/2020 - 14:00 **Chajarí > Santa Fe** **asientosLibres:** 47 **llegadaDestino:** 08/12/2020 - 19:40 **precio:** 464,99 **precioSube:** 390.55 **salidaOrigen:** 08/12/2020 - 14:00 **Federal > Paraná** **asientosLibres:** 41 **llegadaDestino:** 08/12/2020 - 18:50 **precio:** 296,33 **precioSube:** null **salidaOrigen:** 08/12/2020 - 15:40 **Federal > Santa Fe** **asientosLibres:** 65 **llegadaDestino:** 08/12/2020 - 19:40 **precio:** 448,75 **precioSube:** null **salidaOrigen:** 08/12/2020 - 15:40 **Federal > Santa Fe** **asientosLibres:** 2 **llegadaDestino:** 08/12/2020 - 19:40 **precio:** 138,65 **precioSube:** null **salidaOrigen:** 08/12/2020 - 18:50
viaje_2	**destinoFinal**	Rafaela
	origenInicial	Paraná

	estado	ACTIVO
	colectivo	Pat: AC 555 BJ
	empresa	Flecha Bus
	chofer	null
	reservas	Lista Vacía
	trayectos	**Paraná > Rafaela** **asientosLibres:** 24 **llegadaDestino:** 08/12/2020 - 13:30 **precio:** 250,00 **precioSube:** 120.00 **salidaOrigen:** 08/12/2020 - 10:10 **Paraná > Santa Fe** **asientosLibres:** 23 **llegadaDestino:** 08/12/2020 - 11:00 **precio:** 130,00 **precioSube:** 65.00 **salidaOrigen:** 08/12/2020 - 10:10 **Santa Fe > Rafaela** **asientosLibres:** 0 **llegadaDestino:** 08/12/2020 - 13:30 **precio:** 210,00 **precioSube:** 115.38 **salidaOrigen:** 08/12/2020 - 11:00
viaje_3	**destinoFinal**	BuenosAires
	origenInicial	SantaFe
	estado	CANCELADO
	colectivo	Pat: AC 156 FJ
	empresa	Rapido Tata
	chofer	null
	reservas	Lista Vacía

	trayectos	**Rosario > Buenos Aires** **asientosLibres:** 80 **llegadaDestino:** 15/12/2020 - 13:20 **precio:** 265,00 **precioSube:** null **salidaOrigen:** 15/12/2020 - 09:30 **Santa Fe > Buenos Aires** **asientosLibres:** 80 **llegadaDestino:** 08/12/2020 - 13:20 **precio:** 598,00 **precioSube:** 233,00 **salidaOrigen:** 15/12/2020 - 07:00 **Santa Fe > Rosario** **asientosLibres:** 80 **llegadaDestino:** 08/12/2020 - 09:30 **precio:** 222,00 **precioSube:** 128,00 **salidaOrigen:** 15/12/2020 - 07:00
viaje_4	**destinoFinal**	SantaFe
	origenInicial	Paraná
	estado	ACTIVO
	colectivo	Pat: OQA 696
	empresa	ERSA - Fluviales
	chofer	null
	reservas	Lista Vacía
	trayectos	**Paraná > Santa Fe** **asientosLibres:** 51 **llegadaDestino:** 14/09/2020 - 00:40 **precio:** 160,00 **precioSube:** 76.88 **salidaOrigen:** 13/09/2020 - 23:50
viaje_5	**destinoFinal**	SantaFe
	origenInicial	Paraná

	estado	ACTIVO
	colectivo	Pat: OQA 696
	empresa	ERSA - Fluviales
	chofer	null
	reservas	Lista Vacía
	trayectos	**Paraná > Santa Fe** **asientosLibres:** 51 **llegadaDestino:** 13/09/2020 - 10:50 **precio:** 160,00 **precioSube:** 76.88 **salidaOrigen:** 13/09/2020 - 10:00

d. Colección de Pasajeros

Pasajeros		
Documento	**Campo**	**Contenido**
pasajero_1	**nombre**	Federico Hauque
	email	fghauque@gmail.com
	customerId	Dato oculto por motivos de seguridad
	idTarjetaActiva	Dato oculto por motivos de seguridad
pasajero_2	**nombre**	Luis Santiago Re
	email	lsantire2@gmail.com
	customerId	Dato oculto por motivos de seguridad
	idTarjetaActiva	Dato oculto por motivos de seguridad
pasajero_3	**nombre**	Tomas Fleitas
	email	tomas.fleitas@nextper.com

	customerId	Dato oculto por motivos de seguridad
	idTarjetaActiva	Dato oculto por motivos de seguridad

- Subcolección de reservas del usuario que realiza pruebas

Reservas		
Documento	**Campo**	**Contenido**
reserva_1	**authId**	Dato oculto por motivos de seguridad
	destino	BuenosAires
	estado	CREADA
	origen	Rosario
	patenteColectivo	AC 156 FJ
	horaSalida	01/12/2020 - 09:30
	referenciaViaje	viajes/GruZYp8ThMi7AfMz8tFZ
reserva_2	**authId**	Dato oculto por motivos de seguridad
	destino	Santa Fe
	estado	CREADA
	origen	Parana
	patenteColectivo	AC 555 BJ
	horaSalida	08/12/2020 - 10:10
	referenciaViaje	viajes/test1
reserva_3	**authId**	Dato oculto por motivos de seguridad
	destino	Santa Fe
	estado	USADA

	origen	Federal
	patenteColectivo	PAF 688
	horaSalida	28/08/2020 - 12:15
	referenciaViaje	viajes/test2
reserva_4	**authId**	Dato oculto por motivos de seguridad
	destino	Santa Fe
	estado	CANCELADA
	origen	Chajari
	patenteColectivo	AC 156 FJ
	horaSalida	11/12/2020 - 14:00
	referenciaViaje	viajes/test2
reserva_5	**authId**	Dato oculto por motivos de seguridad
	destino	Buenos Aires
	estado	CANCELADA
	origen	Santa Fe
	patenteColectivo	AC 156 FJ
	horaSalida	8/12/2020 - 07:00
	referenciaViaje	viajes/zhOHunGPf7Id1zi346Py
reserva_6	**authId**	Dato oculto por motivos de seguridad
	destino	Rosario
	estado	CREADA
	origen	Santa Fe

	patenteColectivo	AC 156 FJ
	horaSalida	13/09/2020 - 23:50
	referenciaViaje	viajes/zhOHunGPf7Id1zi346Py

II. Casos de prueba

a. Búsqueda de viajes

# de caso	Entrada	Viajes a encontrar
1	**Origen:** Paraná **Destino:** Rafaela **Fecha:** 08/12/2020	Se debe obtener viaje_2.
2	**Origen:** Rosario **Destino:** Buenos Aires **Fecha:** 15/12/2020	Ninguno (El único que cumple los filtros está cancelado)
3	**Origen:** Santa Fe **Destino:** Rafaela **Fecha:** 08/12/2020	Ninguno (No hay asientos libres en este trayecto)
4	**Origen:** Paraná **Destino:** Santa Fe **Fecha:** 08/12/2020	Se debe obtener viaje_1 y viaje_2.
5	**Origen:** Paraná **Destino:** Santa Fe **Fecha:** 13/12/2020	Se debe obtener solamente el viaje_4 (el viaje_5 pasó en un horario anterior)

b. Realización de reservas

# de caso	Entrada	Resultado
6	**Viaje:** viaje_2 CVV: Correcto (123)	Reserva creada exitosamente
7	**Viaje:** viaje_2 **CVV:** Incorrecto (12345)	No se puede crear la reserva
8	**Viaje:** viaje_4 CVV: Correcto (123)	Reserva creada exitosamente
9	**Viaje:** viaje_4 **CVV:** Incorrecto (12345)	No se puede crear la reserva

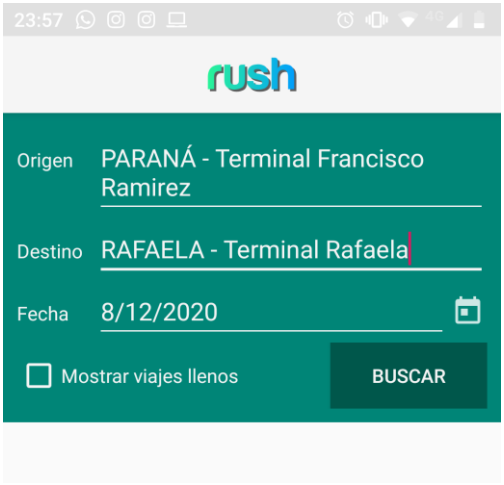
III. Casos de prueba en Android

c. Búsqueda de viajes

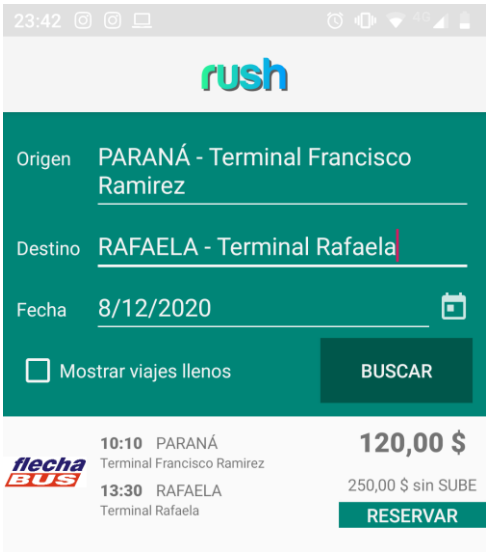
Caso # 1

Entrada:

Origen: Paraná
Destino: Rafaela
Fecha: 08/12/2020



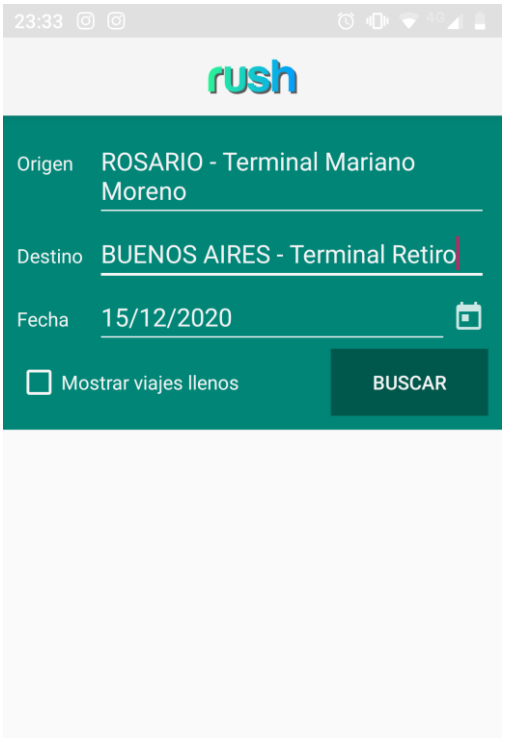
Salida:



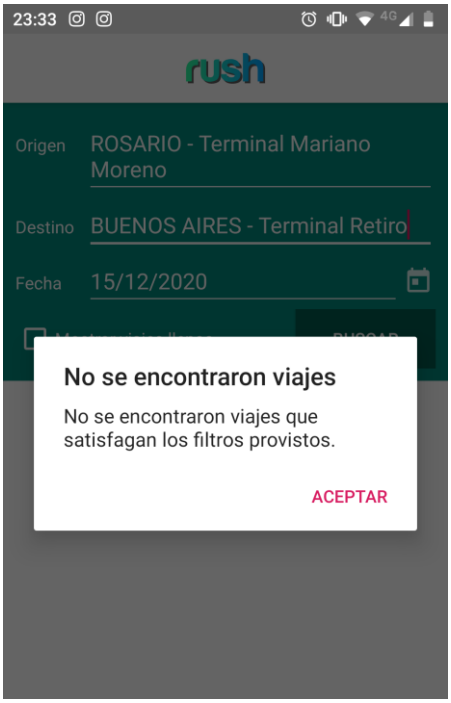
Caso #2

Entrada:

Origen: Rosario
Destino: Buenos Aires
Fecha: 15/12/2020



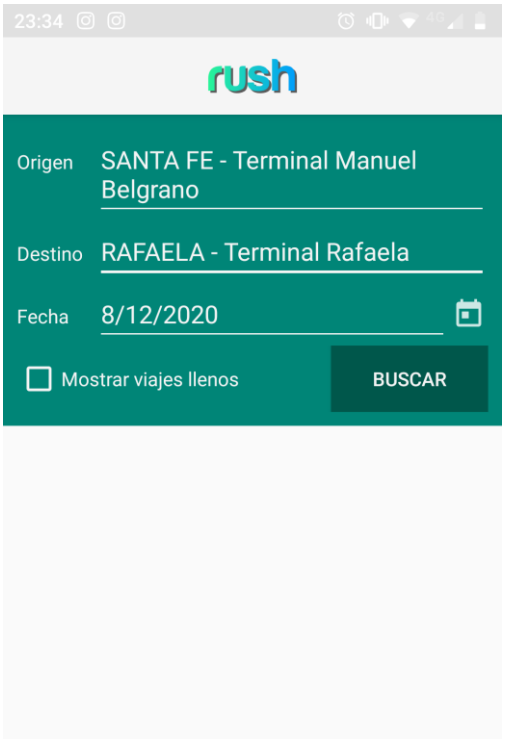
Salida:



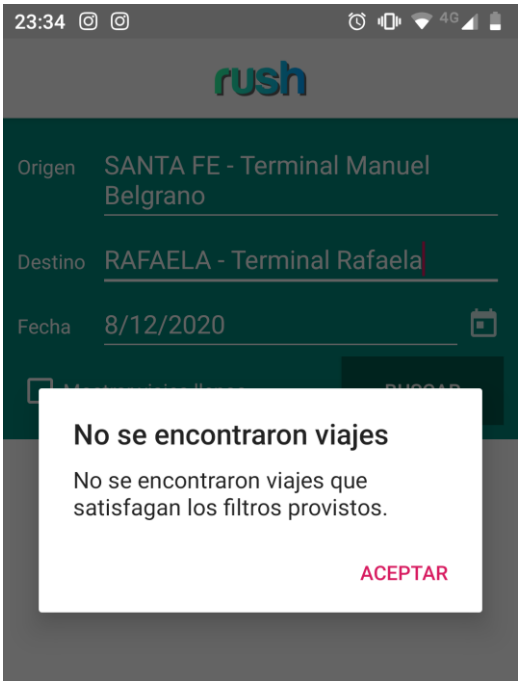
Caso #3

Entrada:

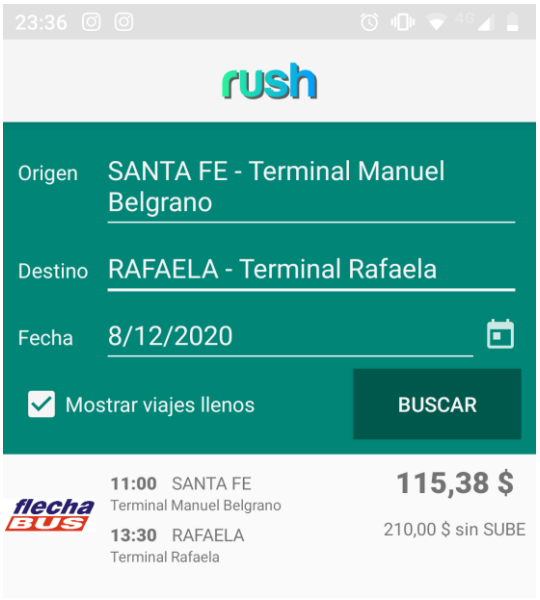
Origen: Santa Fe
Destino: Rafaela
Fecha: 08/12/2020



Salida:



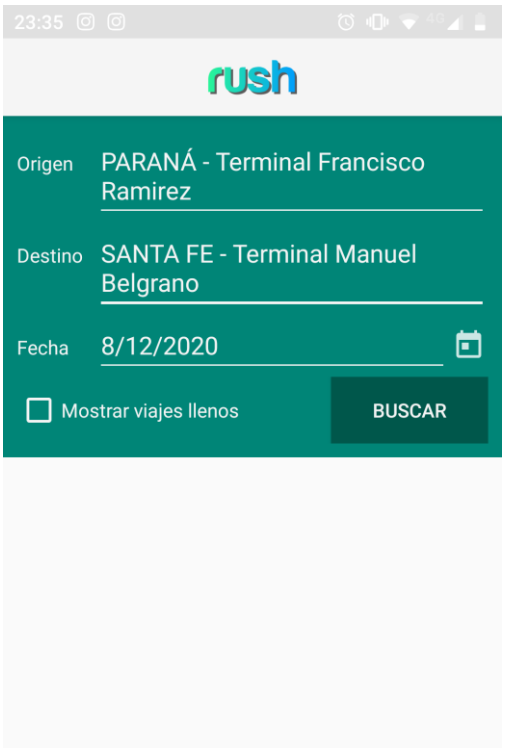
Como hay un viaje que cumple con estos filtros, pero no tiene asientos libres, si se tilda la opción para ver viajes llenos, aparece el viaje en el listado correctamente.



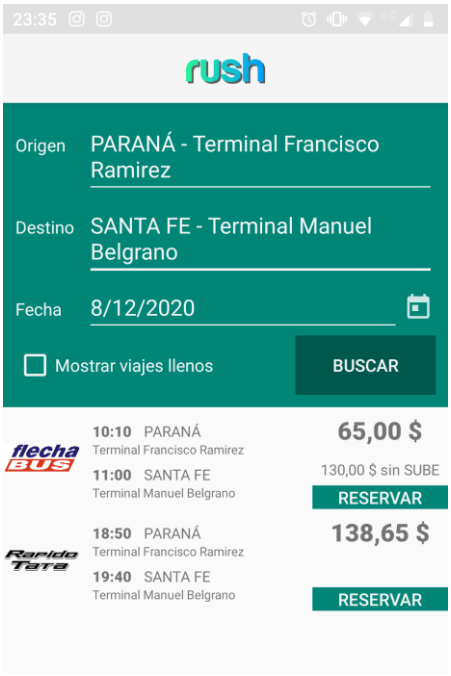
Caso #4

Entrada:

Origen: Paraná
Destino: Santa Fe
Fecha: 08/12/2020



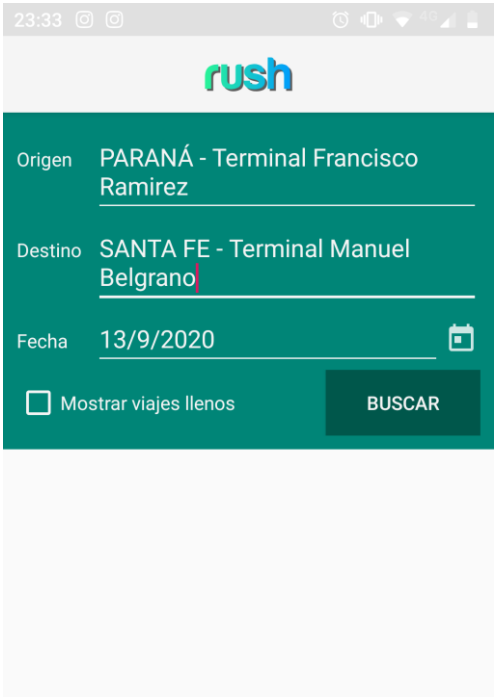
Salida:



Caso #5

Entrada:

Origen: Paraná
Destino: Santa Fe
Fecha: 13/09/2020



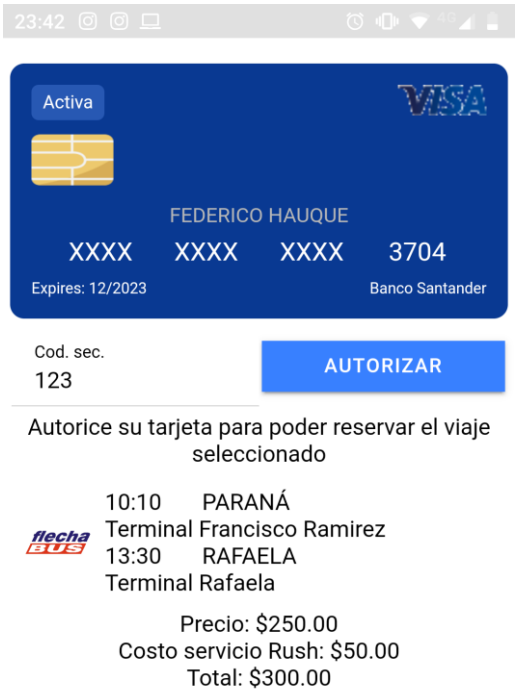
Salida:

d. Realización de reservas

Caso #1

Entrada:

Viaje: viaje_2
CVV: Correcto



Salida: Creación exitosa de la reserva

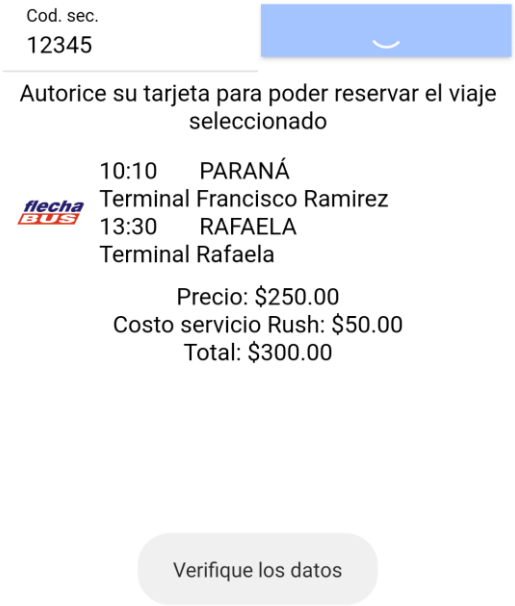
Caso #2

Entrada:

Viaje: viaje_2

CVV: Incorrecto

Salida: (se pide verificar los datos)



Caso #3

Entrada:

Viaje: viaje_4

CVV: Correcto

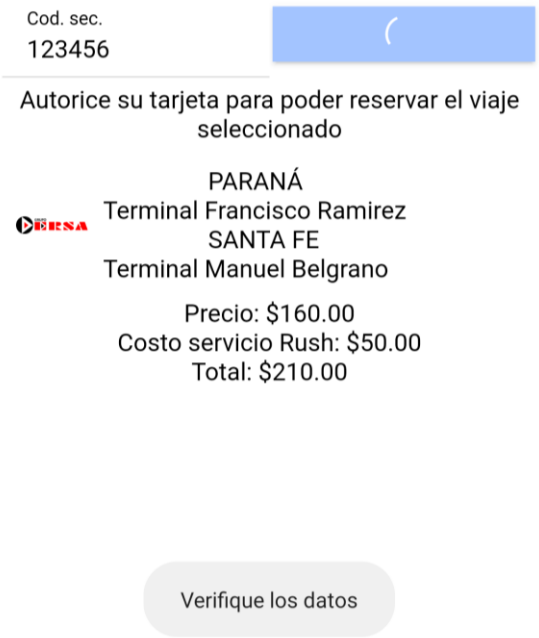
Salida: Creación exitosa de la reserva

Caso #4

Entrada:

Viaje: viaje_4
CVV: Incorrecto

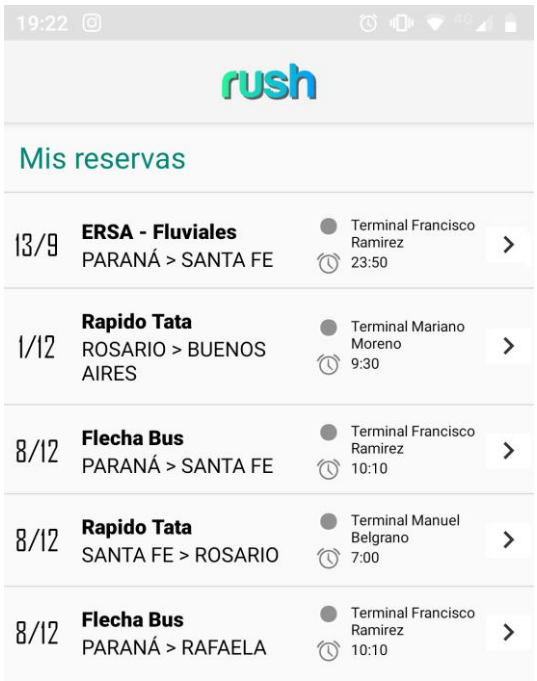
Salida: (se pide verificar los datos)



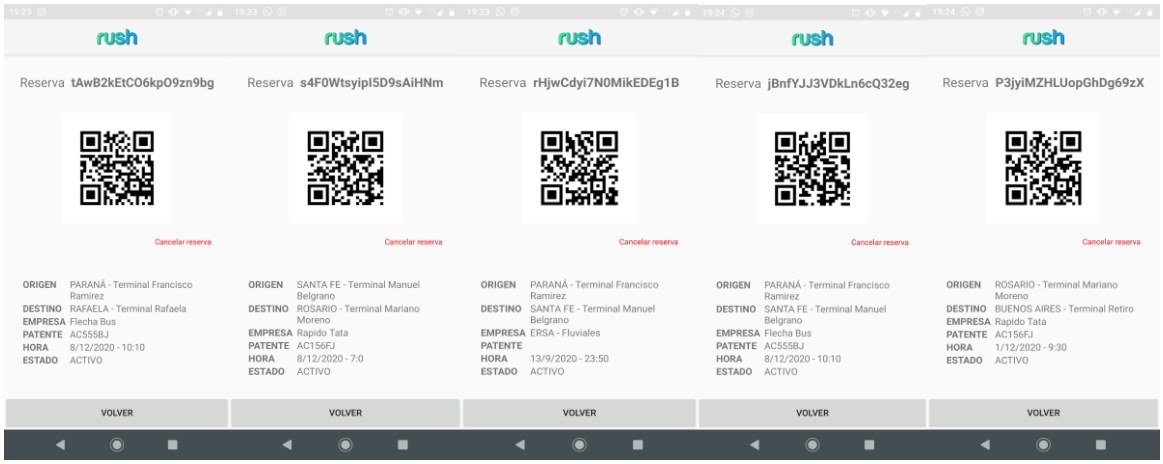
e. Mis Reservas

A continuación, se mostrarán los listados que incluyen las reservas que ya estaban en los datos de prueba y las reservas que se crearon como parte de las pruebas.

Mis Reservas:



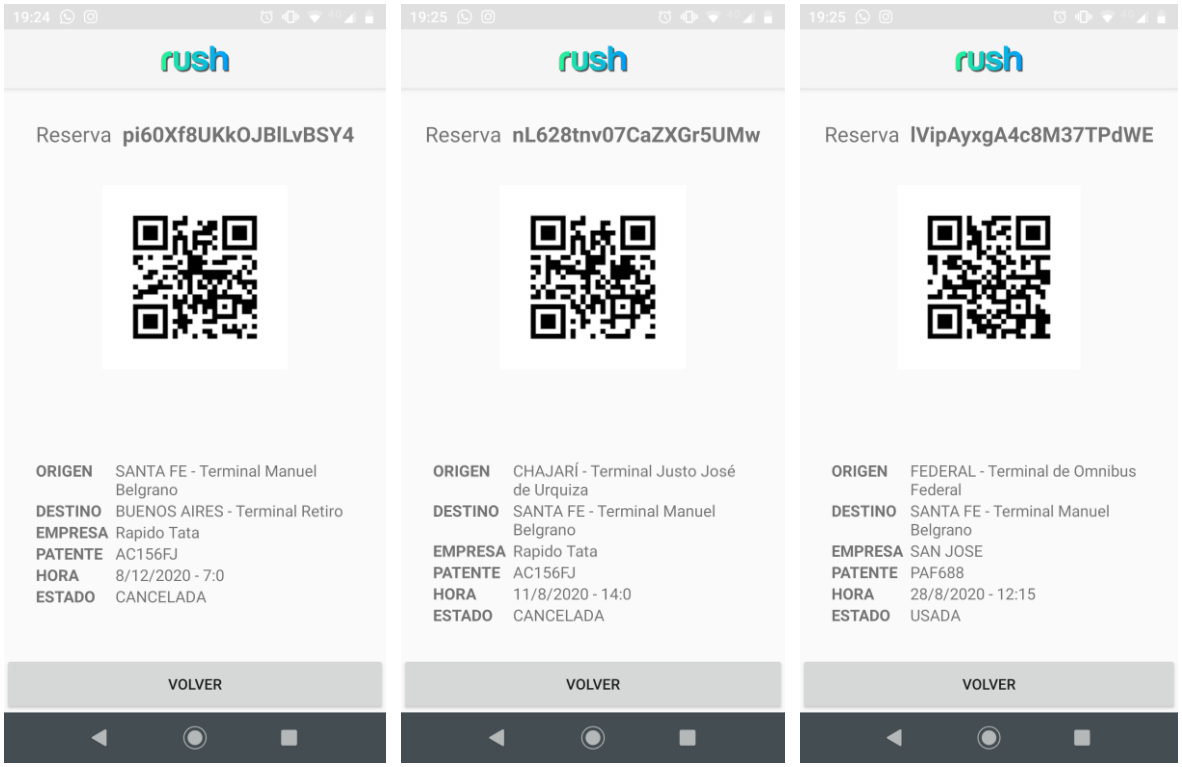
Al abrir las reservas se observa en pantalla los detalles de todas ellas y, dado que todas están en estado CREADA, la opción para cancelarlas se encuentra visible.



f. Historial de reservas

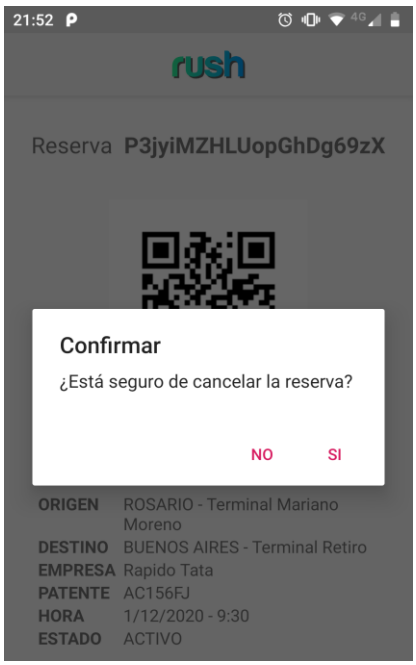
En el historial se muestran las reservas USADAS (en negro) y CANCELADAS (en rojo) presentes en los datos de prueba.

Al abrir las reservas se observan los mismos datos que en el listado de mis reservas, pero en ningún caso está visible la opción para cancelarlas.

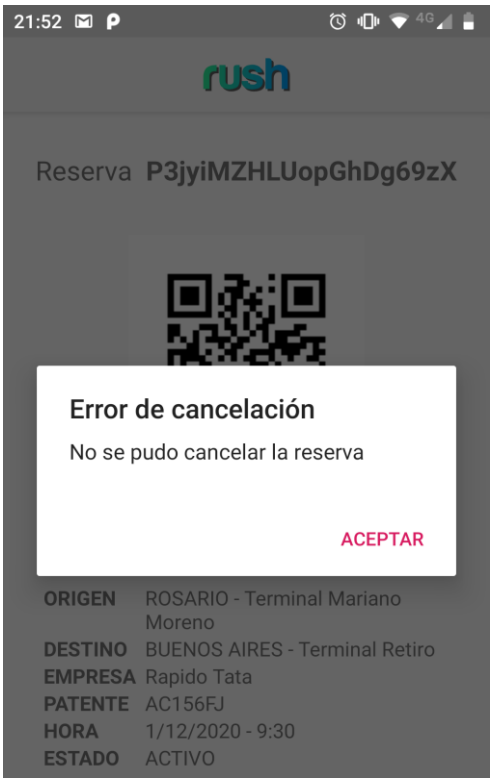


g. Cancelación de reservas

Al cancelar la reserva se pide una confirmación. Si la misma se produce con éxito, se muestra que la misma ahora figura en el historial en color rojo.



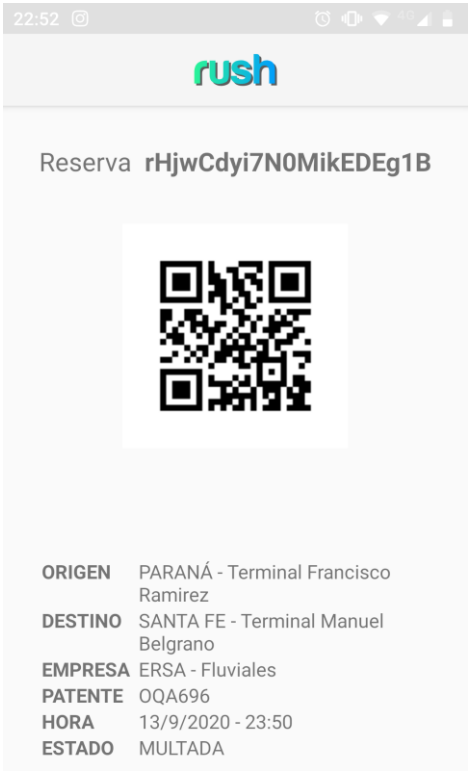
Si sucede algún error (como por ejemplo la falta de conectividad al momento de realizar la cancelación) se imprime un mensaje en pantalla indicando la falla.



h. Multado de reservas

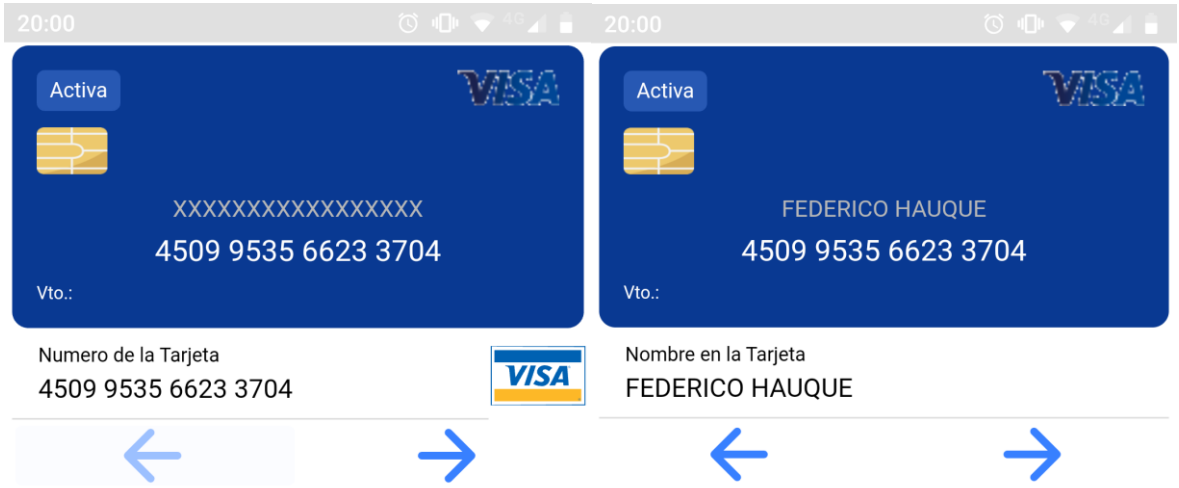
La reserva realizada para el día 13/09 fue dejada sin usar hasta el día 15/09, fecha en la que se ejecutó el cronjob que automáticamente multó la reserva. A partir de este momento la misma figura como multada (en rojo) en el historial de reservas.

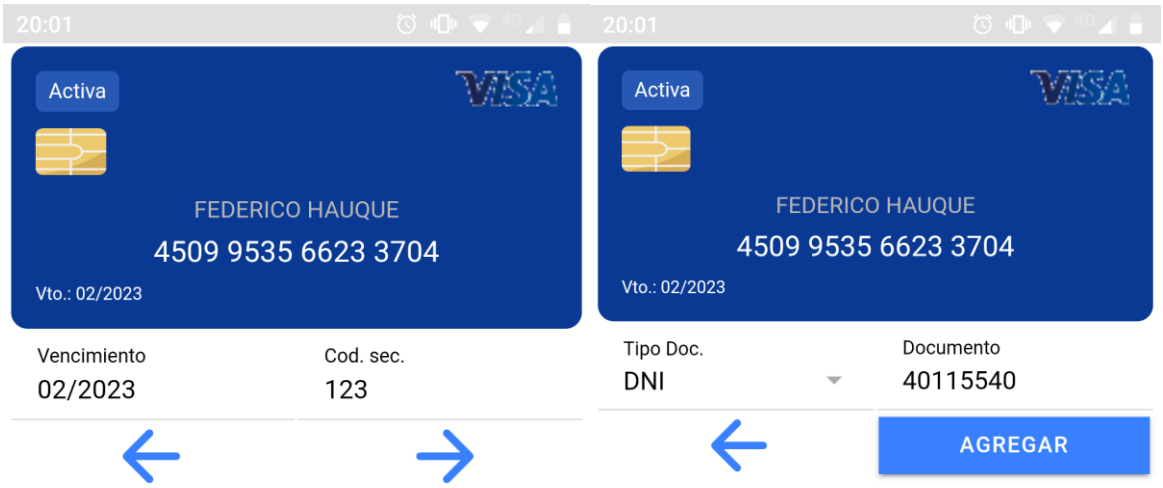
En los detalles de la reserva podemos observar su cambio de estado a “MULTADA”.



i. Carga de tarjetas de crédito

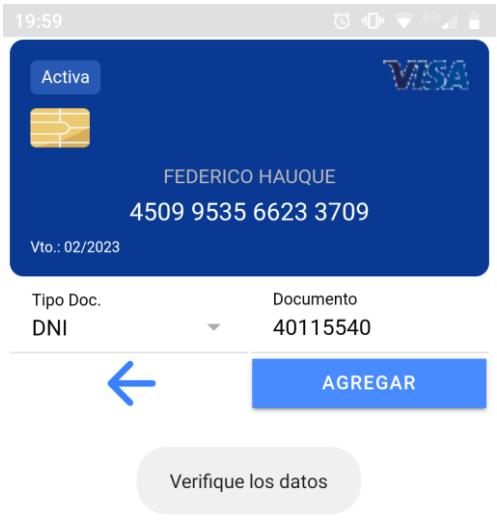
Para la verificación del funcionamiento de la tarjeta en entorno de testing debemos emplear medios de pago ficticios provistos por la API. Ingresaremos los datos de la tarjeta VISA de testing indicada en la documentación de MercadoPago en https://bit.ly/2FmiQAl.





Una vez que se completan los datos correctamente, la tarjeta queda agregada a la billetera en el listado. Como es la única disponible, se la marca como activa.

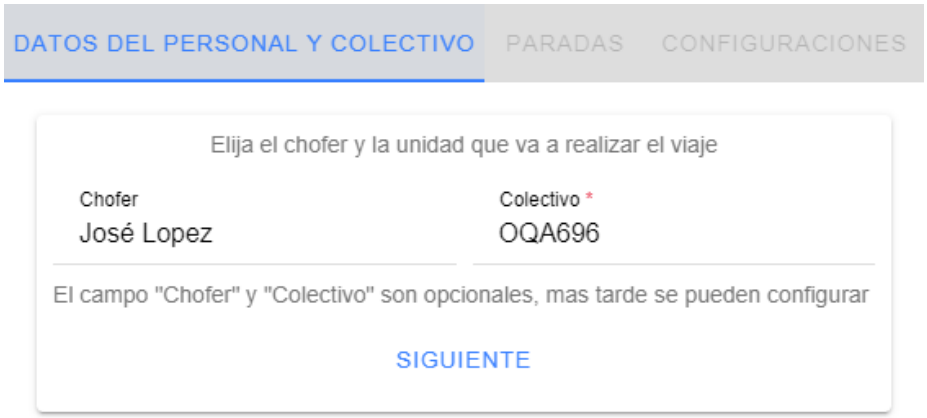
Si alguno de los datos indicados en la carga es erróneo, la tarjeta no resulta aceptada y se solicita al usuario que verifique los datos ingresados.



IV. Casos de prueba en la interfaz de administración

a. Creación de un viaje

Para crear un viaje se deben ingresar los datos del mismo. Los datos de chofer, colectivo y todas las paradas se eligen de listas desplegables.

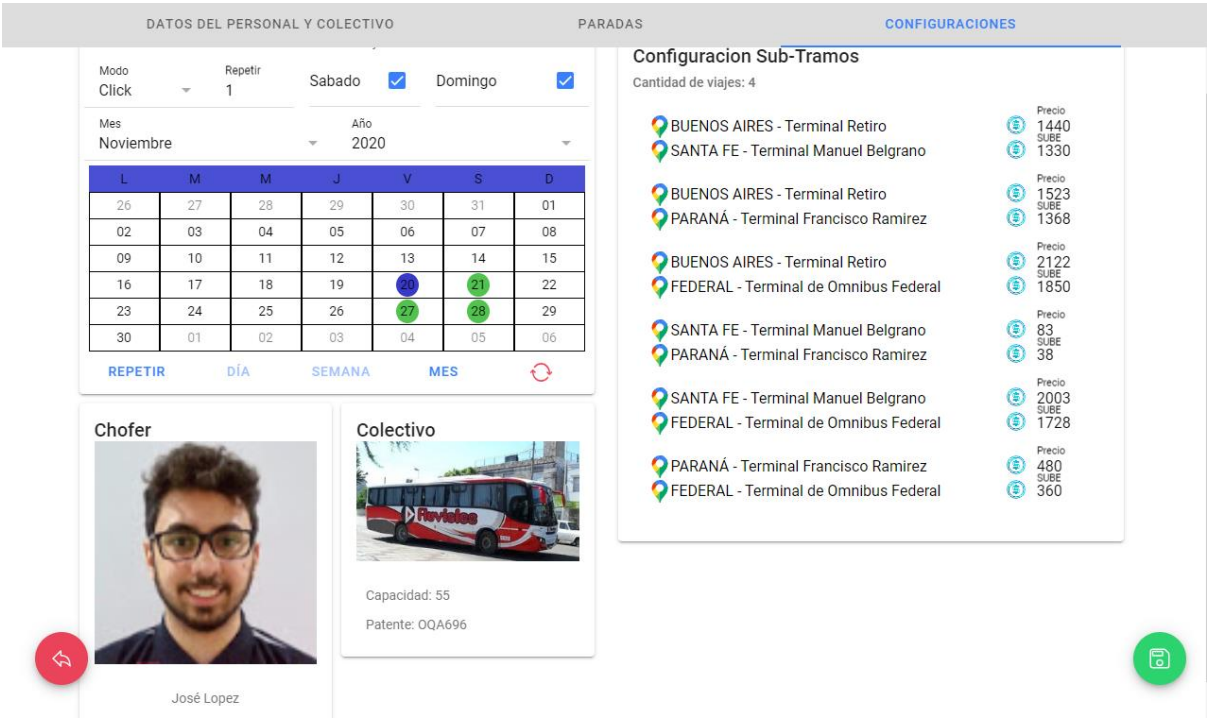


Luego se ingresan las fechas y horas en las que se arribará a cada parada, así como los precios de cada tramo. Ninguna fecha ni horario puede ser anterior al momento en que se está creando el viaje.

En la pantalla final se pueden ingresar todas las fechas en las que se deberían replicar viajes con los mismos parámetros. Esto permite cargar masivamente viajes con los mismos horarios y precios en varias fechas simultáneamente.

Del lado derecho de esta última pantalla se hace presente una lista con todos los trayectos posibles dentro del viaje y sus precios. En este caso, cada trayecto entre dos paradas sucesivas tiene el precio que fue cargado en la pantalla anterior. En el caso de los trayectos entre dos terminales no consecutivas, su precio se calcula como la suma de todos los sub tramos que conectan el origen del trayecto con su destino. En otras palabras, si previamente se habían cargado los precios de los tramos Buenos Aires-Santa Fe y Santa Fe-Paraná, el precio del trayecto Buenos Aires Paraná se calcula como la suma de los últimos 2. De cualquier manera,

si el usuario desea cambiar estos precios, puede hacerlo, pues todos los valores de la lista de la derecha son modificables.

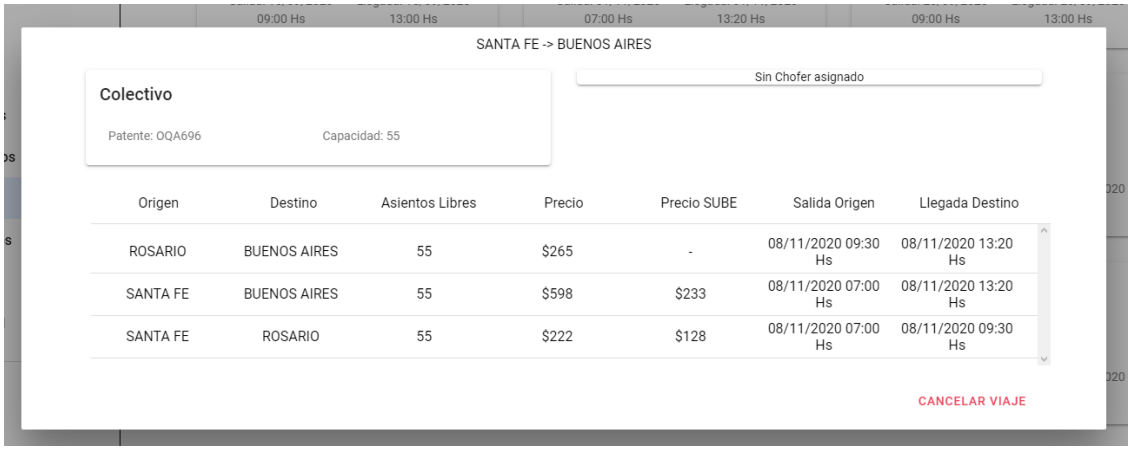


Una vez que se creó el viaje, ya puede observarse el mismo en la lista de viajes de la empresa.

b. Cancelación de un viaje

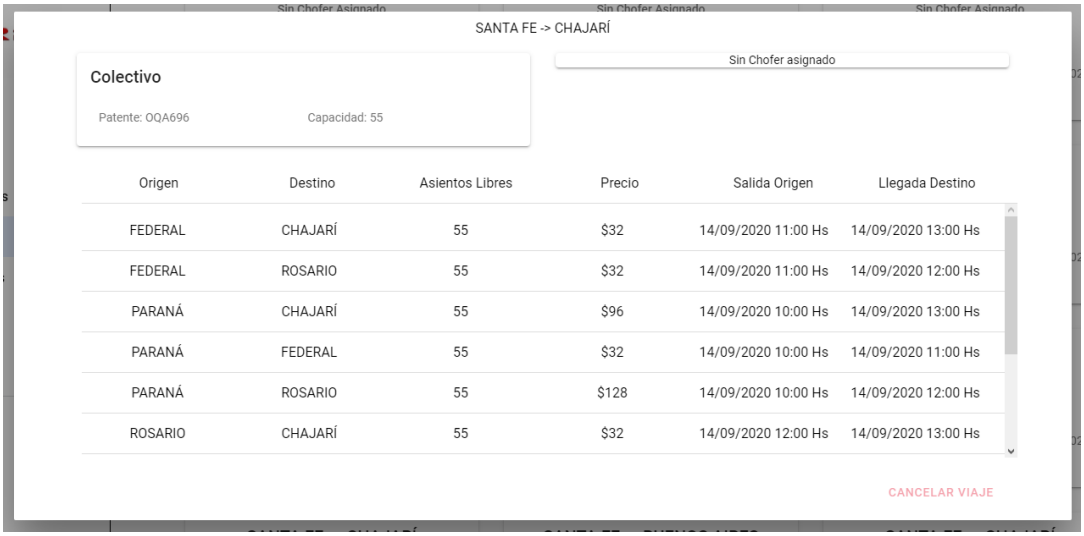
Se elige un viaje de la lista para ver sus detalles.

Junto con la lista de detalles, está presente el botón de cancelar viaje.



Una vez cancelado el viaje, dejará de figurar en el listado de viajes de la empresa.

Si por el contrario se quisiera cancelar un viaje ya pasado, el botón de cancelar no está habilitado y no será posible presionarlo.



V. Riesgos identificados

Al momento de planificar el proyecto se propuso la siguiente tabla que permite cuantificar (subjetivamente) el impacto y la probabilidad de cada riesgo.

VALORACIÓN DE RIESGOS			
PROBABILIDAD		IMPACTO	
Porcentaje estimado	Valoración	Valoración	Cuantificación
0,10	Muy improbable	Muy bajo	1
0,25	Poco probable	Bajo	2
0,5	Moderadamente probable	Moderado	3
0,75	Altamente probable	Alto	4
0,9	Extremadamente probable	Catastrófico	5

A continuación, se presenta una tabla con todos los riesgos identificados, acompañados de su valoración que nos daría la señal de evaluar la necesidad de implementar algún plan de contingencia.

ID	Nombre	Categoría	Prob.	Impacto	ER
R1	Fallas y retrasos excesivos en la implementación de la pasarela de pagos	Proyecto	0,5	3	1,5
R2	Desconocimiento de las tecnologías a emplear	Proyecto	0,25	4	1
R3	Desinterés de las empresas de transporte en implantar el sistema	Empresarial	0,5	4	2
R4	Alteración de los requerimientos del proyecto en medio de la ejecución del mismo	Proyecto	0,75	4	3
R5	Imposibilidad de cumplir con las horas de trabajo estipuladas en el plan	Proyecto	0,25	3	0,75

R6	Aumento desproporcionado de los gastos de servicios en la nube	Proyecto	0,25	3	0,75
R7	Implementación de una UI poco intuitiva	Técnico	0,10	2	0,2
R8	Deserción de un integrante del equipo	Proyecto	0,10	4	0,4
R9	Subestimación de la duración de las actividades	Proyecto	0,5	3	1,5
R10	Enfermedad que impida la participación plena de alguno de los integrantes	Proyecto	0,25	3	0,75
R12	Aparición del cliente con nuevos requerimientos no planeados	Proyecto	0,5	4	1
R13	Actualización de APIs empleadas en el proyecto sin compatibilidad hacia atrás	Técnico	0,25	5	1,25
R14	Implementación de nuevas regulaciones de la CNRT (o cualquier otro organismo de contralor) que imposibiliten la implantación del sistema	Empresarial	0,1	5	0,5
R15	Accidente de algún miembro del equipo que impida su participación plena en el proyecto.	Proyecto	0.1	5	0,5
R16	Fallo en la seguridad de la cuenta de GitHub y accesos malintencionados	Técnico	0.1	5	0,5
R17	Pobre recepción de los usuarios. Elevada cantidad de fallas en la validación con usuarios.	Empresarial	0,25	5	1,25
R18	Falla de un equipo de hardware que pensaba destinarse al desarrollo.	Proyecto	0.1	3	0,3