UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

APLICACIÓN MÓVIL PARA CONECTAR PASAJEROS Y TAXI COLECTIVOS

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

CRISTÓBAL EDUARDO DOTTE SILVA

PROFESOR GUÍA:
NELSON BALOIAN TATARYAN

MIEMBROS DE LA COMISIÓN:
CLAUDIO GUTIÉRREZ GALLARDO
EDUARDO GODOY VEGA

SANTIAGO DE CHILE
2021

Resumen

Muchos de los sistemas de transporte público en Chile y en el mundo han desarrollado aplicaciones digitales para dar un mejor servicio a sus usuarios. El objetivo principal de estas aplicaciones es dar información oportuna y relevante para que el pasajero pueda tomar mejores decisiones al momento de usarlos. Sin embargo, actualmente no existe una solución de este tipo para los taxis colectivos en Santiago. La falta de información es probablemente uno de los problemas más grandes que impiden que este medio de transporte sea usado más frecuentemente. Por lo tanto, en el presente trabajo de título, se enfocó en desarrollar una solución en el mundo del desarrollo de software para disminuir la desinformación entre conductores y potenciales pasajeros de este medio de transporte.

La primera parte de este trabajo de título estuvo enfocada en entender cuales son exactamente las necesidades de información que pueden ser provistas mediante una solución tecnológica basada en el desarrollo de software.

Con esta información se procedió a evaluar soluciones que beneficiarían tanto a los conductores y pasajeros del ecosistema de los taxi colectivos. En la evaluación se presenta aspectos claves para decidir una solución sobre otra, como por ejemplo, la usabilidad de la solución, el tipo de información a compartir entre usuarios y el tiempo de desarrollo de la solución.

La solución se plasmó en un sistema compuesto por tres aplicaciones, dos móviles y una web. La primera aplicación móvil se enfocó en las principales problemáticas que afectan a los conductores de los taxi colectivos; informar su ubicación a sus posibles pasajeros, informarse de la ubicación de sus pasajeros en espera en el recorrido y ubicación de sus colegas. La segunda aplicación móvil abordó las siguientes problemáticas de los pasajeros; ubicación del siguiente taxi colectivo perteneciente a una cierta línea, encontrar nuevas líneas y ver el en el mapa del recorrido de estas. Con respecto a la aplicación web, se enfocó en la administración de líneas de colectivos, dando la posibilidad de agregar nuevas líneas, eliminar líneas deprecadas y modificar líneas en evolución.

Las aplicaciones fueron puestas en producción en un grupo interno en la Google Play Store, donde se desarrolló una prueba de usabilidad y encuesta de satisfacción con el objetivo de conocer la percepción de los usuarios sobre la efectividad de las aplicaciones para solucionar los problemas detectados en el presente trabajo de título. El resultado de esta encuesta de satisfacción arroja una buena percepción de los usuarios sobre las aplicaciones, atribuyéndose este resultado a la inclusión de estos mismos en el levantamiento de problemas y en la validación del diseño en el desarrollo de las funcionalidades.

A mi familia

Tabla de Contenido

Índice de Tablas **v**

Índice de Ilustraciones **vi**

1. Introducción **1**
1.1. Problema abordado . . . 1
1.2. Objetivos . . . 2
1.3. Metodología . . . 2
1.4. Estructura del documento . . . 3

2. Desarrollo de software **5**
2.1. Ingeniería de software . . . 5
2.2. Tecnologías . . . 7
2.2.1. Stack tecnológico . . . 7
2.2.2. Proceso del desarrollo . . . 8

3. Requerimientos **9**
3.1. Conductores . . . 10
3.1.1. Entrevista con un conductor de colectivo de la línea 2004 . . . 10
3.2. Pasajeros . . . 11
3.3. Conclusión . . . 14

4. Diseño de la solución **16**
4.1. Abordando el problema . . . 16
4.1.1. Frecuencia en el recorrido. . . . 16
4.1.2. Información de los recorridos. . . . 17
4.1.3. Información durante el recorrido. . . . 17
4.1.4. Definición de funcionalidades . . . 18
4.2. Metodología de desarrollo de software . . . 18
4.3. Arquitectura de la solución . . . 20
4.3.1. Vista general . . . 20
4.3.2. Interacción con Firebase . . . 21
4.4. Entrega continua . . . 23
4.5. Stack tecnológico . . . 23
4.5.1. Aplicación móvil . . . 23
4.5.2. Aplicación web . . . 24
4.5.3. Back end e infraestructura . . . 24

4.5.4. Pipeline . . . 25
4.5.5. Otros . . . 25

5. Implementación de la solución **26**
5.1. Aplicación móvil del conductor . . . 26
5.1.1. Login . . . 26
5.1.2. Inicio y fin de actividad en recorrido . . . 28
5.1.3. Manejo de asientos disponibles . . . 31
5.1.4. Ubicación de colegas activos y pasajeros en espera . . . 33
5.1.5. Perfil del conductor . . . 35
5.2. Aplicación móvil del pasajero . . . 38
5.2.1. Autenticación anónima . . . 38
5.2.2. Buscador de recorridos . . . 39
5.2.3. Perfil del recorrido . . . 41
5.2.4. Recorridos favoritos . . . 44
5.3. Aplicación web administrativa . . . 45
5.3.1. Login . . . 45
5.3.2. Nueva línea de taxi colectivos . . . 46
5.3.3. Modificar línea existente . . . 51
5.3.4. Eliminación de una línea . . . 51
5.4. Back end . . . 52
5.4.1. Arquitectura . . . 52
5.4.2. Base de datos . . . 54
5.4.3. Consistencia de datos . . . 57
5.5. Integración continua . . . 57
5.5.1. Despliegue en producción . . . 58

6. Evaluación Preliminar **60**
6.0.1. Conductores . . . 61
6.0.2. Pasajeros . . . 61
6.0.3. Análisis . . . 61

Conclusión **63**

Bibliografía **65**

Índice de Tablas

3.1. Distribución de taxi colectivos urbanos por regiones en Chile [7] 10

6.1. Tabla explicativa de los puntajes de satisfacción relacionados a las respuestas de los usuarios 60
6.2. Puntajes de satisfacción relacionados a las respuestas de la aplicación del conductor 61
6.3. Puntajes de satisfacción relacionados a las respuestas de la aplicación del pasajero 61

Índice de Ilustraciones

4.1. Arquitectura de la solución basada en las tres aplicaciones interactuando con Firebase 21

5.1. Primera versión y versión definitiva del Login de la aplicación del conductor 27
5.2. Primera versión y versión definitiva de la pantalla inicial de la aplicación del conductor 29
5.3. Primera versión y versión definitiva de la pantalla con recorrido activo de la aplicación del conductor 30
5.4. Primera versión y versión definitiva de la pantalla con el manejo de asientos de la aplicación del conductor 32
5.5. Mapa cuando ingresa nuevo conductor activo de la línea 33
5.6. Pasajero en espera en el mapa 35
5.7. Flujo de edición en pantalla del perfil del conductor 36
5.8. Primera versión y la versión final del perfil del conductor 37
5.9. Menu del perfil del conductor 38
5.10. Primera versión y versión final del buscador de recorridos en la aplicación del pasajero 39
5.11. Primera versión y versión final del resultado de recorridos cercano a un punto en la aplicación del pasajero 40
5.12. Perfil del recorrido en la aplicación del pasajero 41
5.13. Zoom en un conductor en perfil del recorrido en la aplicación del pasajero . . 42
5.14. Pantallas de espera en la aplicación del pasajero 43
5.15. Primera versión y versión final de favoritos en la aplicación del pasajero . . . 44
5.16. Agregar y eliminar favoritos en la aplicación del pasajero 45
5.17. Login de la aplicación web administrativa 46
5.18. Pantalla inicial de la aplicación web administrativa 47
5.19. Sección información general en agregar nueva línea en la aplicación web administrativa 48
5.20. Sección correos de Google de conductores de la ruta en agregar nueva línea en la aplicación web administrativa 49
5.21. Sección trazado de la ruta en agregar nueva línea en la aplicación web administrativa 49
5.22. Agregar punto relevante en el trazado de la ruta en la aplicación web administrativa 50
5.23. Ruta trazada terminada mediante la unión con el inicio de la ruta 51
5.24. Trazado resultante al quitar punto relevante 52

5.25. Interacción entre aplicaciones y entradas de la base de datos de Firebase . . 55

Capítulo 1

Introducción

Durante la última década hubo una transformación en todas las áreas de la industria y comercio con respecto a cómo estas integran la tecnología en sus procesos para tener un mejor funcionamiento y crecimiento. El rubro del transporte colectivo también se ha sumado a esta tendencia y hoy existen variadas soluciones basadas en el desarrollo de software de aplicaciones para dispositivos móviles y computadores de escritorio para distintos medios de transportes en Chile.

Dentro de los medios de transportes que se han beneficiado de las tecnologías de la información podemos nombrar el Metro de Santiago, que cuenta con una aplicación web que contiene información relevante para sus usuarios permitiéndoles a estos viajar informados a través de sus servicios. El Transantiago de igual manera cuenta con aplicaciones móviles que brindan la información general de sus servicios y en tiempo real de la ubicación de sus buses [1]. Los taxis y servicios particulares (Uber [2], Cabify [3], etc) también cuenta con aplicaciones móviles para brindar una comunicación clara entre pasajeros y conductores.

Si bien varios medios de transporte público cuentan con soluciones tecnológicas que apoyan su mejor funcionamiento brindando información útil y oportuna a sus usuarios, existe uno que se ha quedado atrás en este contexto: el de los taxis colectivos. Este medio de transporte cuenta con un total de 51.657 vehículos distribuidos en 1.200 recorridos o líneas de taxi colectivos urbanos [7], cuyos vehículos tienen la capacidad de transportar cuatro pasajeros a la vez, dando un indicio del alcance que cuenta este medio de transporte en el país.

1.1. Problema abordado

En base lo mencionado anteriormente nace la siguiente pregunta: **¿Como se podrían beneficiar los taxi colectivos incorporando tecnología a su negocio?** Esta pregunta se puede responder desde dos perspectivas; la del pasajero y la del conductor de taxi colectivo.

[1] https://transapp.cl
[2] https://www.uber.com/cl/es/
[3] https://www.cabify.com

Desde el punto de vista de los pasajeros, ellos no cuentan con ningún tipo de sistema o plataforma para poder acceder a información importante con respecto a una línea de taxi colectivos, como por ejemplo, el recorrido, la tarifa, ubicación del próximo vehículo que pasará por donde está esperando el pasajero, entre otros.

Por otro lado, los conductores no tienen una forma de saber con exactitud donde están sus colegas, o la cantidad y ubicación de pasajeros esperando para ser transportados. Esto muestra que existe una falta de información importante entre pasajeros y conductores de taxi colectivos, que limita el crecimiento y el desarrollo del uso de este medio de transporte en nuestro país.

1.2. Objetivos

Objetivo General

El objetivo general de este trabajo de título es crear una solución basada en desarrollo de software para conectar a los usuarios y conductores de colectivos. El principal objetivo de esta conexión es el intercambio de información relevante entre pasajeros y conductores.

Objetivos Específicos

1. Levantar requerimientos desde el punto de vista de los pasajeros y conductores de los taxi colectivos.
2. Definir, diseñar y desarrollar las funcionalidades de la aplicación móvil para pasajeros.
3. Definir, diseñar y desarrollar las funcionalidades de la aplicación móvil para conductores.
4. Definir y desarrollar las funcionalidades necesarias para administrar el sistema, permitiendo el ingreso, modificación y eliminación de líneas/recorridos y conductores asociados a estas mediante una aplicación web.
5. Definir y desarrollar un back end que haga la conexión y mantenga la consistencia del sistema en donde participan las dos aplicación móviles y la aplicación web administrativa.
6. Llevar las aplicaciones a producción mediante un proceso automatizado y fácil de iterar.
7. Validar la solución mediante una prueba cerrada.

1.3. Metodología

Primero se detectarán las problemáticas de los pasajeros y conductores de taxi colectivos mediante un acercamiento al mundo de taxi colectivos, utilizando encuestas no estructuras y estructuras, lo cual esta descrito en el capítulo 3, para luego proceder a traducir las problemáticas detectadas a historias de usuario que serían abordadas en el desarrollo de una aplicación móvil para conductores y una para pasajeros. Estas historias de usuario pasarían por el siguiente ciclo de vida.

1. Diseño (se utilizará Figma [17] para la creación y documentación del diseño)

2. Validación diseño con un grupo de usuarios
3. Implementación *(front end y back end)*
4. Test unitario y funcional
5. Producción

Estas historias de usuario se trabajarán siguiendo una metodología basada en el enfoque iterativo e incremental; la metodología ágil de software. Las dos aplicaciones móviles seguirán la misma estructura en su proceso de desarrollo, la cual será la siguiente.

1. **Primer Sprint.** *(primer periodo de 4 semanas)* Se definirá el pipeline automatizado para la compilación, pruebas unitarias y puesta en producción , y además se desarrollarán las funcionalidades vitales en forma de historias de usuario.
2. **Segundo Sprint.** *(segundo periodo de 4 semanas)* Se procederá a desarrollar las funcionalidades restantes en forma de historias de usuario y configurará el ambiente de producción para llevar las aplicaciones a grupo cerrado de usuarios.

Por último, se desarrollará en un periodo excepcional de 2 semanas la aplicación administrativa para poder agregar, eliminar y modificar líneas y conductores asociados a estas. Al igual que las aplicaciones móviles está será llevada a un ambiente productivo en el cual puedan interactuar usuarios finales.

1.4. Estructura del documento

En el capítulo 2 se introduce las metodologías de desarrollo de software y las tecnologías que están siendo utilizadas actualmente en la industria del software. Se da el contexto del porqué se están utilizando en base a las necesidades del problema que se está abordando y cómo beneficiaría el uso de estas en este trabajo de título.

En el capítulo 3 se aborda el mundo de los taxi colectivos y las problemáticas que perciben los pasajeros y conductores de este medio de transporte. Durante esta sección se definirán los criterios fundamentales de aceptación que debe tener la solución, entendiendo los dolores de los usuarios que utilizarán el desarrollo resultante de este trabajo de título.

En el capítulo 4 se describe la solución de manera conceptual. Se introduce la arquitectura del sistema constituido por cuatro componentes; la aplicación móvil para conductores, aplicación móvil para pasajeros, aplicación web administrativa y el back-end. Además, se especifica cómo se distribuyen las responsabilidades para alimentar el sistema entre los componentes mencionados.

En el capítulo 5 se presenta la implementación concreta de los componentes de cada aplicación. Se divide en subsecciones en donde se abordará cada aplicación que compone la solución, en cada una de estas subsecciones se detallará los principales desafíos y cómo estos fueron abordados. Por último, se mostrará mediante ilustraciones el resultado de cada implementación.

Finalmente, en la capítulo 6 se presenta la validación de la solución, en donde se discuten los resultados de una prueba cerrada del sistema completo. Un grupo de usuarios de la

aplicación para conductores y pasajeros evaluarán si las aplicaciones presentadas soluciona (y en que grado) los problemas percibidos por los usuarios de taxi colectivos.

Capítulo 2

Desarrollo de software

En el mundo de desarrollo de aplicaciones móviles existe una variedad de formas de afrontar el desafío de crear una aplicación móvil de buena calidad. Este desafío se puede dividir en formar un equipo, guiar al equipo mediante un método de desarrollo de software, elegir la batería de tecnologías a utilizar, medir la robustez de la aplicación con pruebas sobre ésta, entre otros. En esta sección se verá cómo el mundo del desarrollo de aplicaciones móviles ha tomado este desafío y que han hecho las mejores aplicaciones del mundo para tener aplicaciones robustas y de calidad. En esta sección estaremos viendo cómo y con que se afrontará este desafío en el presente trabajo de título basado en lo que esta ocurriendo hoy en día en la industria del software.

2.1. Ingeniería de software

Dentro del mundo del desarrollo de software es de suma importancia establecer la metodología que se utilizará. Esta metodología dará el marco de trabajo para estructurar, planificar y controlar el desarrollo de las aplicaciones. Existe un abanico de enfoques en los cuales se pueden agrupar las metodologías existentes.

Primero, el modelo en cascada del desarrollo de software [13], siendo el primero en originarse, plantea tener una secuencia de etapas que se deben completar de manera rigurosa antes de pasar de una etapa a otra. Este modelo, en general, cuenta con la siguiente secuencia de etapa: Análisis de requisitos, diseño del sistema, diseño del programa, codificación, pruebas, implementación o verificación del programa y mantenimiento. Este modelo no se adecua a la naturaleza del problema, debido a que se tiene un problema en la definición y evaluación, debido a que proviene de un cliente que no tiene claro lo que necesita, pero si sus problemáticas en el mundo de los taxi colectivos. Cabe destacar que este *cliente* siendo a la vez el usuario de la solución. Por lo tanto, el enfoque de necesitar que una etapa tenga que estar completa rigurosamente y definida en una etapa temprana, impide la flexibilidad de iterar en conjunto de la retroalimentación del usuario durante el transcurso del desarrollo.

En base a esta necesidad de iterar con la participación del usuario, aparece el modelo de desarrollo de software iterativo e incremental, siendo el modelo que involucra al usuario en

todas las etapas de un proyecto de desarrollo de software, empezando en el levantamiento de requerimientos o problemáticas, para luego tener una partición activa en el desarrollo y validación de la solución [14]. Esta metodología plantea crear una primera versión, solucionando las problemáticas que el cliente percibe, para iterar sobre esta e ir creando un producto que solucione las necesidades del cliente de manera satisfactoria.

En base a la necesidad y la naturaleza del proyecto la metodología que más se adecua (y la más utilizada actualmente en la industria) es una basada en este último enfoque (iterativo e incremental), la **metodología ágil de software** [12]. Esto se debe a que se necesita ir creando iterativamente en cortos periodos de tiempo un software robusto que satisfaga las problemáticas y necesidades de los potenciales clientes (y usuarios a la vez), que utilizaran ambas aplicaciones móviles y la metodología antes mencionada permite hacer la exploración y validación necesaria para llevar a cabo un software que solucione las problemáticas del mundo de taxi colectivos. Esta metodología es la más utilizada cuando se trata de buscar la solución a la necesidad de los clientes de manera iterativa y flexible, y esto se refleja a que actualmente la mayoría de los **startups** del mundo la están utilizando para desarrollar sus softwares.

Una vez decidida la metodología de software que se utilizará, se debe indagar en cómo se aplica esta metodología en los equipos de desarrollo de aplicaciones móviles. En una gran parte de los proyectos que utilizan metodologías ágiles también utilizan el enfoque de ingeniería de software de entrega continua [16] a tal punto que se cuestiona el uso de metodologías ágiles sin este enfoque [44] debido a que nace a problemas de equipos que han fallado utilizando estas metodologías. Este enfoque cuenta con tres etapas que deben ser implementada como parte del desarrollo de las aplicaciones de este trabajo de título.

1. **Automatización de la compilación e integración continua.** La primera etapa consiste en hacer la compilación del código fuente de las aplicaciones. A medida que el proyecto avance se irán agregando funcionalidades y se tiene que verificar que ninguna rompa la compilación de las aplicaciones.

2. **Automatización de pruebas.** La segunda etapa consiste en correr el conjunto de pruebas para garantizar calidad en las aplicaciones que van a ser desplegadas en producción.

3. **Automatización de llevada a producción.** El último paso es llevar las aplicaciones a un ambiente de producción donde se puedan probar desde el punto de vista del usuario final.

Con estas tres etapas implementadas se tendrá un flujo automatizado que permitirá que cada ciclo/iteración desarrollado sea un software robusto que llegue rápidamente al usuario para ser probado y poder incluir retroalimentación que brindará los usuarios que tendrán acceso temprano a las aplicaciones. Teniendo en mente lo anterior, nace naturalmente la pregunta de cómo se implementará la metodología ágil en el proyecto con este enfoque de entrega continua y esto se puede verificar en la siguiente sección.

2.2. Tecnologías

En esta sección se abordará las tecnologías que darán soporte a las dos aplicaciones desde diferentes puntos de vista. En primer lugar se verá las tecnologías utilizadas en el código de fuente que sostendrá los diferentes componentes de software de las apps, y el segundo lugar se verá las tecnologías enfocadas al proceso detrás del desarrollo de software, es decir, lo que dará vida a lo que se ha pronunciado en la sección anterior.

2.2.1. Stack tecnológico

Esta sub sección hablaremos sobre las tecnologías que utilizaremos en el diseño del software y se dividirán en los siguientes componentes: front end y back end [23].

Por parte del **front end** se tienen que desarrollar dos aplicaciones móviles y para dos plataformas distintas (iOS y Android), la primera necesidad que nace es tener una buena velocidad de desarrollo. Esto se traduce a buscar cómo las grandes empresas desarrollan múltiples aplicaciones en las dos plataformas previamente mencionadas. Después de hacer investigación los dos *frameworks* más utilizados en la industria para desarrollar aplicaciones móviles para iOS y Android sin tener que hacer código separado para cada plataforma son: React Native [36] y Flutter [22]. El primero es desarrollado y mantenido por Facebook y la inmensa comunidad que se ha generado desde su creación hace que sea uno de los frameworks más utilizados en el desarrollo de aplicaciones móviles [32]. Por otro lado, Flutter es creado y mantenido por Google y tiene una comunidad relativamente más nueva. Este último también esta siendo utilizada por aplicaciones de renombre [31]. Estos frameworks son los que han hecho posible una agilidad y calidad en las aplicaciones en que las grandes empresas lanzan a producción periódicamente y es natural la necesidad de utilizar uno de estos frameworks en el desarrollo de este trabajo de título (se indicará en la solución propuesta cual de los dos se eligió).

En cuanto al **back end**, el estado del arte para aplicaciones móviles son los basados en la arquitectura **serverless** [6], estos cuentan con grandes beneficios que ayudan una entrega rápida al cliente. Algunos de estos beneficios son:

1. **Cero administración de la infraestructura.** La infraestructura vive en las instalaciones de las empresas que brindan esta experiencia de tener un backend serverless. Las más utilizadas son: AWS [1], Firebase [21], Azure [5], entre otros. Estos se encargan de mantener sus plataformas funcionando de manera correcta para que los desarrolladores solo tengan que enfocarse en escribir código y nada más.
2. **Auto-escalado.** Cada proveedor cuenta con un servicio de disponibilizar servidores dependiendo de la necesidad del cliente. Esto brinda estabilidad para aplicaciones que eventualmente necesiten atender miles de clientes sin tener caídas por sobre carga y en caso contrario cuando necesiten atender pocos clientes solamente brindar lo justo y necesario sin tener tanto hardware ocioso.
3. **Pago por uso.** Estos proveedores de infraestructura nos brindan un servicio de una muy buena calidad y el pago por estos depende netamente en el tamaño de la aplicación que lo esta consumiendo. En el caso de una aplicación pequeña, en general se les da una versión gratuita de sus servicios y esto nos da la oportunidad de utilizar cualquiera

para este trabajo de título.

4. **Reducción del time-to-market.** Al tener una reducción en el tiempo invertido en la mantención y despliegue de la infraestructura, se puede invertir en crear nuevas funcionalidades de cara al usuario y así tener un prototipo mínimo viable de cualquier app en un tiempo que en otro tipo de backend/infraestructura no se podría tener.

En la sección de solución propuesta se hablará de cuál proveedor de infraestructura que soporte un backend con la arquitectura serverless se utilizará en este trabajo de título.

2.2.2. Proceso del desarrollo

Se tiene la misión de organizar el trabajo que hay que hacer para tener dos aplicaciones móviles en producción y hacer este proceso lo más fácil posible. Para esto utilizaremos las herramientas más usadas en la industria del software para cubrir los puntos hablados en la sección anterior.

1. **Trello.** Existen varias herramientas para reflejar en la práctica los principios de las metodologías ágiles y todo lo que esto conlleva. Esto incluye la creación, planificación y organización de los periodos y las tareas relacionadas en las que se irán iterando. Dado a que el equipo es de un integrante se decidió por una plataforma gratuita y de fácil configuración. Trello [43] cumple con los anteriores requisitos además de ser una de las plataformas más utilizadas para organizar proyectos pequeños en el mundo de las metodologías ágiles.
2. **Bitrise.** La entrega continua cuenta también con varias aplicaciones web que permiten reflejar el concepto de automatizar pasos (pipeline) en el desarrollo de software. Dado que se está trabajando con aplicaciones moviles, Bitrise [8] es una de las herramientas que dan opción para compilar aplicaciones híbridas (iOS y android) en sus servidores de manera gratuita. Esto quiere decir que tendremos la posibilidad de incluir un paso de compilación para iOS (utilizando un servidor con macOS que es el sistema operativo para poder compilar aplicaciones para iOS) y esto no es brindado de manera gratuita por casi ninguna herramienta de pipelines. En Bitrise se reflejará todos los pasos que harán que el proceso de creación de las apps sea automatizado, rápido y fiable.
3. **Codecov.** Como se habló anteriormente, dentro de los pasos automatizados en el pipeline de las aplicaciones, existe un paso importante para asegurar la calidad del software que se está creando y este paso es el de la automatización de las pruebas y visibilidad de estas. Por esto, usaremos la herramienta Codecov [9] que es gratuita y de fácil configuración con cualquier software de versionamiento (Github, bitbucket, etc) para darle visibilidad e incluir las pruebas de las aplicaciones en el pipeline que se generará en Bitrise.

Capítulo 3

Requerimientos

Los taxi colectivos son una alternativa viable a considerar tanto en la capital como en regiones del país en cuanto a transporte de pasajeros. Éstos compiten de manera directa con distintos sistema de transportes como metro, buses, taxis y servicios particulares (Uber, Cabify, etc).

Este medio de transporte está presente en Chile desde aproximadamente la mitad del siglo XX con dos variantes: urbano y rurales. El primero atiende viajes con trazados de ida y vuelta previamente definidos, y dentro de los límites urbanos. En cambio el taxi colectivo rural, exceden el radio urbano de una localidad para transportar pasajeros, con una restricción de no superar los 200 kilómetros de recorrido. En este trabajo de título nos enfocaremos en los taxi colectivos urbanos.

Los taxi colectivos urbanos ofrecen un recorrido fijo similar a los buses urbanos, micro o metro, pero con una capacidad máxima de 4 pasajeros al mismo tiempo en el vehículo. El taxi colectivo urbano se caracteriza por ser de color negro con patente amarilla. Como se dijo anteriormente, estos taxi colectivos están distribuidos por toda el país y esta distribución la podemos ver en la **Tabla 3.1**.

Como podemos ver, a nivel nacional se cuenta con 51.657 taxi colectivos urbanos distribuidos en 1200 líneas de recorrido en las distintas regiones de Chile. En la región metropolitana se concentra el mayor porcentaje de taxi colectivos con un 19,6 %, luego la sigue la región de Valparaíso con un 15,9 %. Esto nos da una pequeña pista de cuántos usuarios podrían verse beneficiados por una solución tecnológica que ayude a la comunicación entre conductores y pasajeros.

Para detectar las problemáticas presentes en el mundo de los taxi colectivos, se requiere un acercamiento desde el punto de vista de los dos principales entes; pasajeros y conductores. La forma de abordar este acercamiento se hizo mediante una entrevista no estructura para los conductores. La entrevista no estructura tiene como principal ventaja la definición de un objetivo, y abordarlo de manera flexible. El objetivo de la entrevista no estructura para los conductores fue definida como la detección de problemas de información entre las líneas de taxi colectivos y sus pasajeros. Además, se optó por este tipo de entrevista por la flexibilidad

Región	N° de líneas	Flota	%
Tarapacá	18	254	0,49
Antofagasta	56	3901	7,55
Atacama	40	2201	4,26
Coquimbo	65	4692	9,08
Valparaíso	200	8232	15,9
O'Higgins	66	3284	6,35
Maule	73	3021	5,84
Bío Bío	102	3659	7,08
La Araucanía	41	2557	4,94
Los Lagos	82	3688	7,13
Aysén	19	442	0,85
Magallanes	30	1247	2,41
Metropolitana	326	10134	19,6
Los Ríos	21	1138	2,20
Arica y Parinacota	25	1996	3,86
Ñuble	36	1211	2,34
Total país	1200	51657	

Tabla 3.1: Distribución de taxi colectivos urbanos por regiones en Chile [7]

y cercanía que genera tener una entrevista no tan formal para conocer más a los conductores, que no suelen confiar en agentes externos a su contexto laboral.

Por otro lado, se procedió a hacer una encuesta estructurada para los pasajeros, que apunta a validar los problemas percibidos por los conductores de taxi colectivos, y así, obtener una intersección de la falta de información entre los dos, pasajeros y conductores. La estructura de la encuesta se hizo mediante los problemas levantados por los conductores, pero a la vez, dejando un cierto grado de libertad para que los pasajeros puedan expresar problemas que no perciben los conductores.

3.1. Conductores

En esta sección se hablará de cómo los conductores perciben el estado actual de sus servicios y así poder conocer un poco más sobre las problemáticas existentes por parte de los conductores.

3.1.1. Entrevista con un conductor de colectivo de la línea 2004

Con el propósito de conocer más de cerca el funcionamiento y las problemáticas existentes en el mundo de taxi colectivos por parte de los conductores, se procedió a entrevistar a Rene Dotte, conductor y co-propetario de la línea 2004 con su trayecto ubicado en la comuna de Ñuñoa. Además, cabe recalcar que Rene ha trabajado en el mundo de los taxi colectivos desde hace aproximadamente 20 años en distintas líneas de colectivos. En esta instancia se logró recopilar información clave para entender el mundo de los taxi colectivos desde la perspectiva

de un conductor y sus necesidades de información insatisfechas.

1. **Frecuencia para salir al recorrido.** Para un conductor de taxi colectivo es muy importante saber con que frecuencia salir al recorrido en búsqueda de sus pasajeros. Si el tiempo que transcurre desde que salio un colega es poco, se puede traducir en que todos los pasajeros que están esperando en el recorrido sean tomados por el colega que va adelante y el conductor en cuestión se quede sin pasajeros. Por otro lado, si el tiempo es mucho, se podría acumular pasajeros esperando y estos decidan elegir otra opción como metro, micro, taxi, etc. Con esta información podemos deducir que para un conductor es importante saber en que momento salir, en base a la información de localización de sus colegas y número de pasajeros en la ruta.
2. **Informar al pasajero.** Para un conductor la preocupación de que un pasajero opte por otro medio de transporte es algo de todos los días, por esto mismo los conductores suelen dejarles número de contactos a ciertos pasajeros para que puedan contactar cuando están en ruta y así poder informarles donde vienen y que lo puedan esperar. Aun así, un porcentaje de los pasajeros que lo esperan terminan optando por otro medio de transporte ante la incertidumbre. De este punto se puede desprender que hay una falencia importante entre la comunicación de pasajeros y conductores de taxi colectivos.
3. **Los pasajeros casi siempre son los mismos.** Durante su servicio en distintas líneas de colectivo el entrevistado informó que pudo encontrar un patrón sobre que los pasajeros que frecuentan en su ruta casi siempre eran los mismos, indicando que no existe suficiente información o difusión de su recorrido para que un nuevo cliente se suba a su colectivo. Este punto tiene sentido al no haber mucha información de los recorridos en línea, en donde la gente siempre busca cómo llegar de un lugar a otro.

Durante la entrevista se pudieron evidenciar múltiples oportunidades para que un desarrollo tecnológico apoye a la comunidad de conductores de taxi colectivos. Sin embargo, para que una posible solución sea consistente entre pasajeros y conductores, se debería confirmar con los pasajeros y hacer una intersección de dolencias entre los pasajeros y conductores. Por lo tanto, se procedió a analizar de cómo los pasajeros perciben las problemáticas existentes en el mundo de taxi colectivos.

3.2. Pasajeros

Esta sección se orientará a la visión de los pasajeros sobre cuáles son las principales problemáticas de los taxi colectivos en la Región Metropolitana. Para esto se hizo una encuesta mediante Google Forms y publicitándola en diferentes plataformas (grupos de Facebook, foros universitarios y grupos de WhatsApp vecinales), logrando obtener más de 100 respuestas divididas en segmentos de frecuencia de uso del servicio; pasajeros frecuentes, no tan frecuentes y que nunca utilizan el servicio. La encuesta fue estructura de la siguiente forma.

La principal pregunta para identificar los problemas en los taxi colectivos desde el punto de vista desde sus **usuarios mas frecuentes** fue: **¿Qué problemas identificas en el día a día en los colectivos?** en donde las respuestas predeterminadas y su porcentaje de respuesta se mostrarán a continuación.

1. Siempre vienen llenos **75 %**

2. Solo reciben efectivo **75 %**
3. Se demoran mucho en llegar **25 %**
4. No entiendo la tarifa variable que cobran **25 %**
5. No entiendo por donde pasan **0 %**
6. El recorrido no me deja donde necesito **0 %**

Este menu de respuestas fueron inferidas desde la entrevista con el conductor, tratando de cuadrar las dolencias de las dos partes; conductor y pasajero. Al final de esta pregunta también se dio la posibilidad de que los usuarios pueden agregar texto libre describiendo problemas que ellos perciben pero que no estuvieran en la lista mostrada anteriormente. Un extracto de las respuestas más frecuentes recibidas se muestra a continuación.

1. "Es más caro y solo me conviene en ciertos días o a ciertas horas"
2. "Pocos colectivos de algunas líneas. "
3. "muchos de ellos se ven en mal estado, no hay fiscalización"

De los problemas mencionados se puede desprender que hay otro tipos de problemas que van más allá de lo tecnológico, pero a la vez también refuerza algunos problemas que si se podría solucionar con tecnología, cómo por ejemplo, el problema de *pocos colectivos en una línea* puede ser un reflejo de una descoordinación en el horario de salida al recorrido por parte de los conductores.

Para los **pasajeros no tan frecuentes** (i.e. utilizan un par de veces a la semana o al mes el servicio de taxi colectivos) se formularon las siguientes preguntas.

1. **¿Por qué no usas más frecuentemente el colectivo cómo medio de transporte?** En esta pregunta las respuestas predeterminadas y el porcentaje que obtuvo cada una se muestran a continuación.

   (a) Siempre vienen llenos **41.5 %**
   (b) Se demoran mucho en llegar. **31.7 %**
   (c) No entiendo por donde pasan **31.7 %**
   (d) Solo reciben efectivo **31.7 %**
   (e) El recorrido no me deja donde necesito **26.8 %**
   (f) No se si hay recorridos que me sirvan **26.8 %**
   (g) No entiendo la tarifa variable que cobran **24.4 %**

   La lista anterior está ordenada de mayor a menor frecuencia.

   El punto (a) podría estar relacionado con una falta de dotación de colectivos de la línea o como se mencionó anteriormente, una mala distribución de los taxi colectivos en la ruta. Esto último también podría estar relacionado con el punto (b). Finalmente, los demás puntos están relacionados con la falta de información que perciben los pasajeros sobre la línea o ruta de taxi colectivos.

2. **¿Existe otro problema de los colectivos que te gustaría comentar?** A continuación, al igual que en el segmento de usuarios más frecuentes, se mostrará un extracto de las respuestas obtenidas y se indicará una percepción general de estas respuestas.

   (a) "Filas para tomarlos en terminal, no sabría dónde esperar uno en mitad del recorrido."
   (b) "En general solo entiendo los recorridos cerca de mi casa, seria bueno tener una pagina donde se encuentren todos sus recorridos y explicando las tarifas, no si es posible, pero quizás si estos se mostrarán en google maps también seria más fácil de conocer los demás recorridos. "
   (c) "En la pregunta anterior no hay alguna respuesta que me represente: mi razón es que existen medios alternativos que cumplen el mismo recorrido (generalmente micro)."

Entre los problemas mencionados anteriormente, se refuerza el punto sobre la falta de información de los usuarios sobre los taxi colectivos. Finalmente para los **usuarios que no utilizan** taxi colectivos se les formuló las siguientes preguntas.

1. **¿Por qué no usas servicio de colectivos para transportarte?** Las preguntas y porcentajes se exponen a continuación.

   (a) No se si hay recorridos que me sirvan **67.3 %**
   (b) No entiendo por donde pasan **56.4 %**
   (c) No entiendo la tarifa variable que cobran. **56.4 %**
   (d) Solo reciben efectivo. **47.3 %**
   (e) El recorrido no me deja donde necesito **27.3 %**
   (f) Se demoran mucho en llegar. **25.5 %**
   (g) Siempre vienen llenos **25.5 %**

   De igual manera como se hizo para los usuarios que utilizaban colectivos se dejo un campo para que el usuario especificara si percibe otro problema existente para que este decida no utilizar el servicio de taxi colectivos. A continuación mostraremos un extracto de las respuestas y que se puede desprender de estas.

   (a) "El no conocer quién es el conductor, en general si debo tomar un taxi o algo así prefiero apps que me den los datos del conductor para así poder mandárselos a mis cercanos por si pasa algo, los colectivos no me dan esa confianza, incluso más cuando toca subirse en la parte de adelante, me siento muy insegura."
   (b) "(Contexto: ahora ya no ocupo colectivo pero cuando iba al colegio a veces sí). Siento que no hay información en ningún lado acerca de dónde pasan (nunca he buscado pero tampoco me ha aparecido). Me da miedo que al estar sentada tan cerca de la gente haya algún hombre acosador (o el mismo conductor si es que voy sola). Me da lata que la gente (por estar sentada cerca?) meta conversa."
   (c) "Falta de aseo de conductores y vehículos"

En base a las respuestas predeterminadas y las agregadas por los usuarios, podemos ver que para los usuarios que no utilizan los taxi colectivos, la poca información que existe en internet no los incentiva a empezar a utilizar el servicio. En base a los problemas predeterminados más votados, podemos deducir que la poca información de las rutas existentes y cómo se distribuyen estas en el mapa es el problema más importante para que los usuarios que no tienen mucha información sobre los colectivos empiecen a utilizar el servicio. Para confirmar esto y tener una referencia de cuántos usuarios podrían empezar a utilizar el servicio, se formuló la siguiente pregunta.

2. **¿Utilizaría los colectivos como medio de transporte si se solucionaran los problemas que mencionó anteriormente?** De esta pregunta, un 30 % respondió que **SI**, otro 50 % respondió que **TAL VEZ** y finalmente un 20 % respondió que **NO**.

   Lo anterior se traduce a que potencialmente un **80 %** de los usuarios que no utilizan taxi colectivos, podrían empezar a utilizar el servicio si los problemas que mencionaron anteriormente se solucionaran. Sin embargo, para entender a los usuarios que indicaron que no los utilizarían incluso solucionando los problemas que mencionaron previamente, se les formuló una última pregunta.

3. **¿Por qué no utilizarías los colectivos como medio de transporte si se solucionan los problemas?** Donde un extracto de las respuestas agregadas fueron las siguientes.

   (a) "Prefiero la bici o la micro"

   (b) "Personalmente prefiero un sistema de transporte parecido al de las app (uber, didi, etc), es un medio de transporte que utilizo solo en ocasiones de emergencia, la poca infraestructura y conciencia vial, me genera incomodidad"

   (c) "Te obliga a estar cerca de otra gente, no fomenta el uso de buses que, pudiendo llegar a ser eléctricos, contaminan menos por persona. Aparte los buses y el metro aportan dinero al estado con el cual se puede mejorar el transporte."

   (d) "El metro es igual o más útil y confiable"

   En base a las respuestas agregadas por este segmento de usuarios, se puede desprender que la mayoría prefiere otro medio de transporte debido a la confianza, utilidad o por la capacidad económica del usuario. Por otro lado, un cierto porcentaje también indico sobre la contaminación de este medio de transporte. En el trabajo de título de Jorge Villa (cita) se presenta la factibilidad económica de renovar un 30 % de taxi colectivos urbanos por autos eléctricos en Chile. Dando paso a que usuarios que se preocupan por la contaminación del medio de transporte en un futuro también tengan un incentivo de empezar a utilizar taxi colectivos.

3.3. Conclusión

En base a lo mostrado y discutido durante la sección, podemos verificar que existen problemáticas que se repiten en los distintos segmentos de usuarios. Las cuales se pueden ordenar en

base a la alta concurrencia en la perspectiva de los problemas en el servicio de taxi colectivos por parte del usuario y conductor.

1. **Información sobre líneas existentes.** Basado en las opciones predeterminadas *No se si hay recorridos que me sirvan*, *No entiendo por donde pasan* y las agregadas por los usuarios. Se define cómo criterio de aceptación de la solución que debe tener una forma de poder buscar líneas de taxi colectivos.
2. **Información sobre la ruta.** Basado en las opciones mencionadas en el punto anterior y las agregadas por los usuarios indicando que no saben donde esperar a un taxi colectivo, se define cómo criterio de aceptación de la solución una forma de mostrar la información de la ruta, esto agrupa información general y georeferencial de esta.
3. **Posición de pasajeros y conductores.** Basado en lo conversado durante la sección sobre la problemática de cómo distribuir a la flota en el recorrido teniendo en consideración la posición de los pasajeros y colegas conductores, se define como criterio de aceptación de la solución brindar la posición de los pasajeros en espera en la ruta y de los colegas conductores que están haciendo el recorrido en la ruta.

Finalmente, la construcción de una buena solución tecnológica para los conductores y usuarios de taxi colectivos, podría tener un gran impacto en la cantidad de usuarios nuevos y activos de los taxi colectivos. Cómo se ha visto durante la sección, existe mucha desinformación por parte de los usuarios sobre el mundo de los colectivos y esta incertidumbre hace que finalmente opten por otro medio de transporte (Metro de Santiago, TranSantiago, taxis o Uber). Al mismo tiempo, los conductores también se verían beneficiados con mayor información de sus clientes y colegas, así dando paso a poder optimizar sus recorridos y utilizar la tecnología a su favor para brindar un mejor servicio a sus clientes.

Capítulo 4

Diseño de la solución

4.1. Abordando el problema

Teniendo en cuenta las problemáticas tratadas en la sección 3, se procedió a diseñar una solución basada en las necesidades y carencias más importantes percibidas por los usuarios de taxi colectivos. Esto se hizo basándonos en las soluciones que existen actualmente en otros medios de transporte, cómo las aplicaciones para saber donde vienen los buses de un recorrido de Transantiago, aplicaciones de Taxi (EasyTaxi) y otros. Se resumen las problemáticas y la forma de abordarlas de la siguiente manera.

4.1.1. Frecuencia en el recorrido.

La frecuencia en el recorrido es un problema que puede ser abordado con información de los pasajeros en espera y los colegas activos en el recorrido. Esto se debe a que los conductores no tienen suficiente información de cuando salir al recorrido por no saber si existen pasajeros esperando o si sus colegas ya recogieron a todos los pasajeros que estaban en espera. En consecuencia, se procedió a evaluar una forma de comunicar la ubicación en tiempo real de los pasajeros y colegas. En base a lo dicho al principio de esta sección y considerando la forma más utilizada basada en la geolocalización se tratará de desarrollar una aplicación móvil que comunique la ubicación mediante el hardware del teléfono. Como se tiene dos clientes distintos en el mundo de los taxis colectivos (pasajeros y conductores), se identifica la necesidad de desarrollar una aplicación móvil para cada uno y que estos tengan sus funcionalidades en base a las problemáticas que afectan a cada uno y en su conjunto.

En cuanto a la frecuencia, en el recorrido para los conductores se traduce en incorporar las siguientes funcionalidades en su aplicación.

- Compartir la ubicación cuando el taxi está activo en el recorrido
- Cambiar el estado de la actividad del taxi durante el recorrido entre activo e inactivo
- Poder ver la ubicación de sus colegas en el recorrido y si estos están activos o no.
- Poder ver la ubicación de los pasajeros en espera para el recorrido.

Por otro lado, para que el conductor de un taxi pueda ver la ubicación de los pasajeros en espera para su recorrido, se proveen las siguientes funcionalidades en la aplicación de los pasajeros.

- Compartir la ubicación cuando está esperando en el recorrido
- Cambiar el estado de espera en el recorrido (esperando o no)

4.1.2. Información de los recorridos.

En la sección 3 se vio que los usuarios muchas veces no utilizaban taxi colectivos como medio de transporte debido a la poca información que existe de los recorridos disponibles y si estos le sirven en su trayecto o no, entre otras razones. Por lo tanto, para dar una mejor visibilidad y entendimiento a los usuarios (y posibles nuevos usuarios) de taxi colectivos, se procedió a desarrollar funcionalidades para ayudar a solucionar el problema.

En la aplicación de los pasajeros, se incluirán dos funcionalidades que abordan el problema. La primera es un **Buscador de líneas de taxi colectivos en base a una ubicación**, siendo de utilidad para la exploración de nuevas líneas y aclaración para los usuarios que ya las utilizan. En segundo lugar, y principalmente como complemento a la anterior, se agregará un visualizador de los resultados arrojados por el buscador.

Por otro lado, se agregará una funcionalidad en la aplicación web administrativa para poder agregar el **trazado en el mapa** de líneas de taxi colectivos y su información relacionada.

4.1.3. Información durante el recorrido.

Una vez localizados los recorridos existentes y viendo cómo éstos se distribuyen en un mapa, quedan aún algunas preguntas abiertas para informar de manera más precisa al usuario respecto a un recorrido o línea en especial. Estas son: ¿Dónde viene el próximo taxi colectivo?, ¿Cuanto me cobrará aproximadamente?, ¿Estará en horario de funcionamiento?. Todas estas preguntas están relacionadas a la información relativa a un recorrido, por lo tanto se procederá a traducirlas en funcionalidades.

En la aplicación de los pasajeros, se incluirá un **perfil del recorrido** en donde el usuario será capaz de visualizar información relevante, como por ejemplo, tarifa del recorrido, horario de funcionamiento, posición del próximo taxi colectivo activo en la ruta y capacidad disponible de asientos en el taxi colectivo.

En la aplicación de los conductores, se incluirá una función de manejo de la capacidad del taxi colectivo para tener información en tiempo real de cuántas personas viajan y así el usuario en espera podrá tomar una decisión con esta información sobre esperar o no al próximo taxi colectivo activo en ruta.

En la aplicación administrativa se agregará una forma de añadir la información requerida mencionada anteriormente: horario de funcionamiento, tarifa del recorrido, conductores asociados a la ruta y nombre del recorrido. Esto sumado a la información del trazado del recorrido completa el modelo de recorrido en el sistema.

4.1.4. Definición de funcionalidades

Una vez definidas todas las funcionalidades asociadas a conductores y pasajeros de taxi colectivos, y la administración de la conexión entre ambos, surge la necesidad de definir en detalle como debería comportarse y cuales serían los criterios claves para determinar si la funcionalidad cumple con su objetivo o no. En base a esta necesidad se define que cada funcionalidad seguirá una estructura basada en la metodología de desarrollo de software elegida para este trabajo de título. La próxima sección se definirá la metodología y cómo las funcionalidades descritas en esta sección tomarán una forma definitiva.

4.2. Metodología de desarrollo de software

Para este trabajo de título se decidió utilizar una metodología de software que fuese acorde a la naturaleza del problema. En nuestro escenario,los clientes son los conductores de taxi colectivos y los pasajeros actuales y potenciales que usarían la aplicación resultante de este trabajo. Por esto se decidió utilizar una metodología de software que tuviese una capacidad flexible de adaptarse a los requisitos dinámicos que vayan saliendo en el camino. Este es un paradigma muy común para las startups de todo el mundo. La metodología de software que se adapta a estos requisitos, como vimos en el capitulo 2, es la metodología ágil basada en un concepto iterativo e incremental.

Las metodologías ágiles proponen estructurar las funcionalidades requeridas mediante la activa participación del usuario final. Esto quiere decir que es el usuario el que relata cómo debería ser la funcionalidad y en base a esto se generan los criterios de aceptación mínimos para la certificación de la funcionalidad. El resultado de esto serán historias de usuarios [28] que encapsularan la funcionalidad desde un punto de vista del usuario final. La forma que tendrán es la siguiente.

1. **Descripción.** Es la descripción de la funcionalidad desde el punto de vista del usuario final.
2. **Criterios de aceptación.** Son los requisitos mínimos funcionales desde el punto de vista del usuario final para que la historia cumpla con lo descrito.
3. **Criterios de aceptación técnicos.** Son requisitos que debe tener la implementación de la historia de usuario desde el punto de vista del desarrollador.
4. **Diseño.** Se adjunta un archivo con el diseño de la historia de usuario. Este diseño es una maqueta de como debería verse la historia de usuario en un dispositivo real.

Una vez definida la estructura de la historia de usuario, se define el ciclo de vida por el que cual estas historias de usuarios deben pasar para llegar al usuario final.

1. **Diseño.** En esta etapa se crea el diseño de la historia de usuario. Para la documentación y desarrollo del diseño se decidió por utilizar Figma [17] por dos razones. La primera, es multiplataforma, esto quiere decir que se puede utilizar en Windows, Mac, en el navegador, etc. La segunda, y la más importante, es que la curva de aprendizaje debe ser baja, y esto es un punto a favor para personas que no han trabajado en diseño anteriormente.
2. **Validación diseño con un grupo de usuarios** La base del diseño es la interacción

con los usuarios, y esta no se puede lograr sin tener una opinión diversa sobre el diseño que se está construyendo, por lo tanto, antes de pasar a desarrollar las historias de usuarios se hizo una verificación con usuarios de la línea 2004. Esta verificación fue realizada de manera informal con conductores y pasajeros frecuentes de la línea 2004, se le presentó la interfaz propuesta y se tomo notas de las sugerencias frecuentes que iban surgiendo. Esta verificación resultó fructífera debido a que se logró incorporar una retroalimentación temprana por parte de los usuarios en el diseño final.

3. **Implementación.** En la implementación se tomó en consideración todos los criterios de aceptación y el diseño de la historia de usuario y se llevó a código.
4. **Test unitario y funcional** El concepto de metodologías ágiles viene acompañado de generar constantemente un producto de cara al usuario e ir moviéndose *ágilmente*. El problema común que enfrentan muchos de los proyectos basados en estas metodologías es que las entregas sacrifican calidad por rapidez. La solución a este problema pasa por desarrollar un código de calidad y robusto. Para esto se define que el proyecto debe contar con un porcentaje mínimo de cobertura de su código. Este porcentaje nos brindará la seguridad de que estamos creando un software robusto y de calidad. El porcentaje de cobertura para el proyecto se definirá en un 85 % que es el porcentaje utilizado por múltiples proyectos de renombre, y este porcentaje marcará la aprobación para pasar la historia de usuario a la siguiente etapa del ciclo de vida.
5. **Producción** La historia de usuario esta implementada en un programa que será llevado al usuario final.

Una vez definida la estructura de historia de usuario y su ciclo de vida, la pregunta siguiente sería *¿Cómo éstas se distribuirán en el tiempo contemplado para desarrollar el trabajo de título?*. Estas historias de usuario irán agrupadas en un periodo de tiempo para su desarrollo llamado Sprint [41], cada Sprint tendrá una duración de 4 semanas. La idea es que de cada Sprint salga una versión mínima funcional, probada y robusta de las aplicaciones.

La distribución en Sprints de las historias de usuario relacionadas a cada aplicación será la siguiente.

Aplicación de los pasajeros

1. **Sprint 1**
   - Inicialización del proyecto
   - Buscador de recorridos
   - Resultado del buscador de recorridos
2. **Sprint 2**
   - Perfil del recorrido
   - Informar espera en el recorrido
   - Recorridos favoritos

Aplicación de los conductores

1. **Sprint 1**

- Inicialización del proyecto
- Login
- Informar actividad en el recorrido (activo/inactivo)

2. **Sprint 2**
   - Manejo de asientos disponibles
   - Perfil del conductor
   - Ver colegas activos y pasajeros en espera en el recorrido

Aplicación administrativa

1. **Sprint 1**
   - Inicialización del proyecto
   - Login
   - Agregar un nuevo recorrido
2. **Sprint 2**
   - Modificar un recorrido
   - Eliminar un recorrido

4.3. Arquitectura de la solución

En este sección se verá la arquitectura de la solución desde una perspectiva general y de cómo los principales componentes interactúan con Firebase y sus servicios.

4.3.1. Vista general

La arquitectura de solución cuenta con **cuatro** componentes claves: aplicación móvil para los conductores, aplicación móvil para los pasajeros, aplicación web para administrar las líneas de colectivos y los conductores asociados, y un back end para gestionar los datos entre los componentes mencionados anteriormente.

La aplicación destinada para usuarios conductores brindará la información en tiempo real del estado del conductor en la ruta de la línea de taxi colectivo. Además, los conductores serán capaces de ver la ubicación de los pasajeros en espera en el recorrido.

De la misma forma, la aplicación destinada para usuarios pasajeros brindará la información en tiempo real del estado del pasajero en espera en una ruta de una línea de taxi colectivo. También, los pasajeros serán capaces de buscar lineas de taxi colectivos mediante un buscador geográfico, y ver el estado de los conductores en una linea de taxi colectivo de interés.

En paralelo, la aplicación web administrativa está destinada para agregar, modificar y eliminar líneas de taxi colectivos, y sus conductores asociados. La principal función de esta aplicación web es mantener un control sobre las líneas y conductores asociados que entran al sistema.

Finalmente, el back end se encargará de manejar los datos (información de los conductores, pasajeros, lineas, trazado del recorrido, etc) entre las tres aplicaciones mencionadas anteriormente. Las principales funciones son: comunicación de los datos entre las aplicaciones, mantener la consistencia de los datos y definir reglas de seguridad de escritura y lectura en la base de datos.

4.3.2. Interacción con Firebase

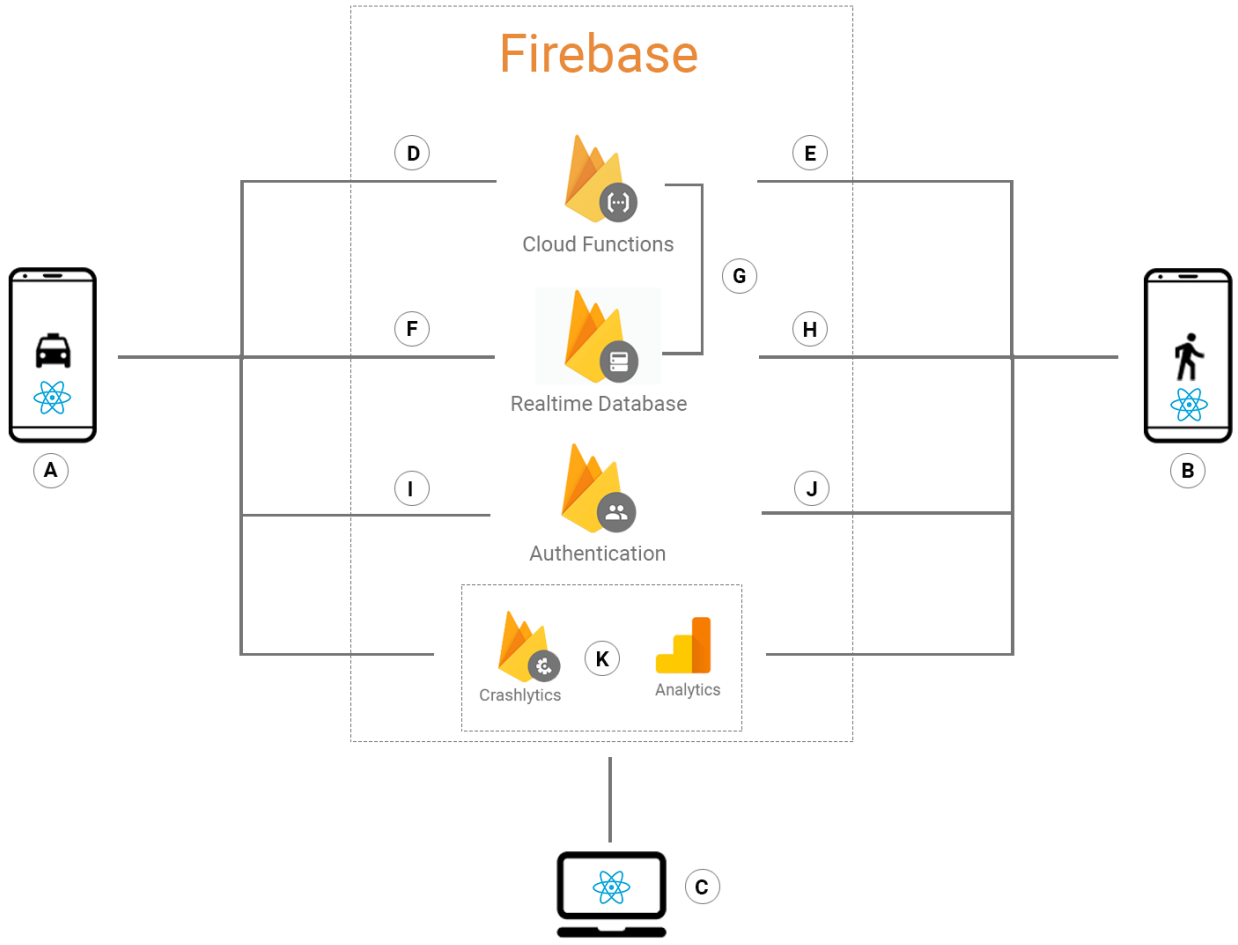

Figura 4.1: Arquitectura de la solución basada en las tres aplicaciones interactuando con Firebase

Con las problemáticas asociadas a historias de usuario y teniendo un panorama general de cómo será la solución basada en tres aplicaciones, se procedió a estructurar este sistema bajo un diagrama que explica la interacción entre los principales componentes de la solución con Firebase, como se puede apreciar en la Figura 4.1. En esta sección explicaremos, basándonos en el diagrama anterior, cómo las aplicaciones serán capaz de cumplir con las historias de usuario definidas anteriormente.

Los puntos **A**, **B** y **C** son la aplicación del conductor, del pasajero y administrativa, respectivamente. Estas serán hechas bajo un framework multiplataforma: React Native [36] en donde su principal ventaja como se hablo en la sección 2 es que se generará una aplicación

para iOS y Android con un mismo código fuente. La decisión de utilizar este framework se discutirá en la sección de tecnologías con mas profundidad.

Para implementar el **login** de las aplicaciones se utilizará un servicio de Firebase [21] llamado **Firebase Authentication**, el cual brinda una librería para hacer el proceso de autenticación en pocos pasos. En cuanto a la aplicación del conductor contará con una autenticación con cuenta de Google y que está representada por el punto **I**. La aplicación de los pasajeros contará con un una autenticación anónima (se explicará más en detalle en la sección 5) representado por el punto **J**. Por último, la aplicación administrativa también contará con una autenticación con cuenta de Google y solamente permitirá acceso a cuentas de Google autorizadas (también se explicará en la sección 5). Cabe mencionar que Firebase Authentication da soporte a la autenticación anónima y con cuenta de Google, por lo tanto satisface los requerimientos de autenticación de las aplicaciones.

Para compartir los datos en tiempo real se utilizará un servicio de Firebase llamado **Firebase Realtime Database**. La ubicación de los pasajeros y de los conductores en el recorrido, los asientos disponibles y los datos en general se guardaran en esta base de datos. En especifico, el punto **F** da referencia a como la aplicación del conductores compartirá su ubicación, asientos disponibles y información del conductor al sistema. Por otro lado, el punto **H** da referencia a como la aplicación de los pasajeros compartirá su ubicación y estado de espera en el recorrido al sistema. Finalmente, la aplicación administrativa agregará datos relacionados al recorrido (y sus conductores) al sistema.

Para el procesamiento de datos y tratamientos de datos que requieran una capacidad de hardware mayor a la de un teléfono celular, se utilizará otro servicio de Firebase llamado **Cloud Functions** el cual brinda un back end *serverless* (el cual se definió sus ventajas para el proyecto en la sección 2). Este back end tendrá la función de procesar llamadas desde las aplicaciones del estilo *"Obtiene todos los recorridos cercanos a este punto geográfico"* (representado por los puntos **D** y **E**) y podrá hacerlo de manera eficiente gracias a la escalabilidad del servicio. Además, este tendrá la capacidad de actualizar la base de datos para hacer limpieza de datos u otro tipo de procesamiento de mantención que esta requiera (representado por el punto **G**. Las características de este back end se verán en profundidad en la sección 5.

La componente marcada con el punto **K** usa dos servicios de Firebase: **Crashlytics** y **Analytics**. Crashlytics brinda información sobre las fallas de la aplicación y genera un reporte general. Analytics aporta información de cómo el usuario navega en las aplicaciones, lo que permite determinar el comportamiento de estos. En resumen, en su conjunto, el componente marcado como punto **K** ayudará a monitorear la experiencia de usuario en las aplicaciones, desde el punto de vista de fallas y navegación dentro de estas y así poder actuar en conforme a datos concretos en caso de cualquier evento percibido gracias a estas herramientas.

Finalmente, como se pudo ver durante esta sección, se establece un conjunto de servicios de Firebase para poder dar soporte a todas las funcionalidades de las aplicaciones. En la sección de tecnologías se discutirá la elección de Firebase como proveedor de servicios para aplicaciones web y móviles.

4.4. Entrega continua

El proceso de llevar las aplicaciones a los usuarios finales durante el desarrollo de esta es un pilar necesario para toda aplicación. Para esto es necesario definir un proceso que garantice la entrega del código fuente para la instalación de la aplicación en el teléfono celular de un usuario final. En el contexto de este trabajo de título y la metodología de software que se utilizó, se procedió a introducir el concepto de **entrega continua**, el cual apunta a la construcción, prueba, y liberación del software de forma más rápida y frecuente de manera automatizada. Esto es consistente con la agilidad que deberían tener las aplicaciones para ir iterando y agregando funcionalidades detectadas.

El concepto de Pipeline se refiere a la automatización de un proceso en el desarrollo de software. Al estar trabajando con metodologías ágiles para el de desarrollo de aplicaciones móviles, estaremos en constante crecimiento de la versión que irá finalmente a producción (en el caso de este trabajo de título: App Store y Play Store), por lo tanto, gracias a este proceso automatizado, se tendrá una versión robusta, confiable y que sea fácil de desplegar por intermedio de estas tiendas virtuales en todo momento. Esto nos lleva a la necesidad de automatizar el proceso del despliegue desde el código fuente a las tiendas. En la sección 5 se verá la implementación de este proceso automatizado en detalle.

4.5. Stack tecnológico

Durante esta sección se han mencionado someramente algunas tecnologías que se utilizaran para del desarrollo de cada componente de la solución. Sin embargo, no se ha presentado que tecnologías se utilizarán para cada componente de la solución. A continuación se presentarán formalmente las tecnologías utilizadas y la justificación de su elección.

4.5.1. Aplicación móvil

En la sección 2 se revisaron las tecnologías disponibles para desarrollo de aplicaciones móviles mas utilizadas en la industria que podían generar una aplicación para iOS y Android con un mismo código fuente. Estas son React Native y Flutter.

React Native es un framework hecho en JavaScript que permite desarrollar aplicaciones en este mismo lenguaje. Es mantenido por Facebook por más de 5 años y cuanta con una de las comunidades de usuarios más grande. La inmensa variedad de librerías para agilizar el desarrollo de funcionalidades dentro de aplicaciones móviles ha hecho que sea una opción de prioridad en múltiples proyectos a gran escala.

Por otro lado, Flutter es un framework hecho en Dart y para desarrollar en este mismo lenguaje. Es mantenido por Google desde hace 3 años y su comunidad esta en crecimiento. Aun así, este framework para desarrollar aplicaciones híbridas está siendo utilizado por grande empresas a nivel mundial [31].

Para el desarrollo de las aplicaciones móviles de este trabajo de título, se decidió usar React Native, principalmente por la gran cantidad de librerías disponibles y su inmensa comunidad dándole soporte a estas, lo que da la seguridad de tener un abanico disponible para elegir

librerías que agilicen el desarrollo de, por ejemplo, un mapa en la aplicación y saber que está en una fase madura y estable.

4.5.2. Aplicación web

En la sección 4.5.1 se introdujo la tecnología a utilizar para desarrollar las aplicaciones móviles. React Native está hecho basado en React.js [33] que es un framework de JavaScript para construir interfaces de usuario. React también está presente en los frameworks más conocidos y mantenidos para el desarrollo web. Por lo tanto, para optimizar el tiempo destinado al aprendizaje de la tecnología y mantener la uniformidad en cuanto a lenguajes de programación entre plataformas (todo utiliza JavaScript), se procedió a utilizar React para desarrollar la aplicación web administrativa.

4.5.3. Back end e infraestructura

En la sección 2 se discutió los beneficios de una de las arquitecturas más utilizadas actualmente debido a los servicios en la nube que están disponibles en la industria del software. El back end serverless satisface las principales condiciones para ser la infraestructura de la conexión entre las aplicaciones desarrolladas.

En el ámbito de ofertas de servicios en la nube, existen tres grandes proveedores: Amazon Web Services, Google Cloud Platform y Microsoft Azure, los que en general ofrecen servicios muy similares. Para el objetivo de este trabajo de título se necesitaba principalmente tres características: Rapidez de configuración, adaptabilidad con aplicaciones móviles y web, y de bajo precio (preferentemente gratuito).

En la búsqueda de un servicio que satisficiera las características anteriores, se encontró Firebase. Este servicio de Google es una adaptación de los servicios de Google Cloud Platform para agilizar proyectos de software y brindar toda la infraestructura necesaria para brindar mantención y escalabilidad de manera fácil.

Se procedió a elegir Firebase por sobre servicios en Amazon Web Services y Microsoft Azure por las siguientes razones.

1. **Cumple con las tres características buscadas.** Se puede configurar toda la arquitectura mencionada en la Figura 4.1 en un par de horas, todos los servicios de Firebase fueron creados pensando en ser utilizados por aplicaciones móviles y web, por lo tanto, vienen con documentación para integrarse de manera rápida en cualquier plataforma. Además cuenta con un plan gratuito (y luego se paga dependiendo del uso).
2. **Back end serverless en JavaScript.** Firebase cuenta con su servicio de Cloud Functions, el cual es un framework que permite correr código de back end con una arquitectura serverless [6] en JavaScript. Lo cual es consecuente con el lenguaje de programación utilizado en el resto de las tecnologías utilizadas en las aplicaciones. No hay necesidad de aprender un lenguaje de programación nuevo.
3. **Experiencia.** El alumno de este trabajo de título ha tenido experiencias previa trabajando con Firebase y Google Cloud Platform, por lo tanto, dado la naturaleza del proyecto y la cantidad de componentes a crear para brindar la solución propuesta,

ayuda bastante a completar la solución en el tiempo contemplado.

4.5.4. Pipeline

En la sección 4.4 se discutió sobre la importancia de tener un proceso automatizado para llevar a producción las aplicaciones. Para este trabajo de título se necesitaba un servicio o plataforma para poder desarrollar este Pipeline. Durante la búsqueda se encontró con diferentes plataformas que cumplían con este objetivo, entre ellos, Bitrise, CircleCI, Jenkins, Fastlane, etc.

Se optó por utilizar Bitrise dada las siguientes razones.

1. **Orientado para aplicaciones móviles.** Bitrise autodefine su labor como *"Automatizamos tus desarrollo de aplicaciones móviles desde la compilación y pruebas hasta el despliegue en producción"*. Esto se acomoda a las necesidades de este trabajo de título, donde se busca desarrollar un pipeline para aplicaciones móviles.
2. **Plataforma automantenida.** Bitrise brinda su servicio mediante una plataforma web y esta disponible en cualquier dispositivo que cuente con un navegador. Además, es mantenida por un equipo de profesionales increíble dando como resultado un 100 % de uptime.
3. **Compilación iOS.** Bitrise cuenta con un plan gratuito que brinda un número de ejecuciones de un Pipeline definido por el usuario al mes. Lo importante es que esta ejecución tiene un soporte para compilación de aplicaciones en iOS, lo cual es pagado en la mayoría de las plataformas.

Para finalizar, Bitrise como la mayoría de las plataformas mencionadas para desarrollar un Pipeline que enmarque la automatización de los procesos de desarrollo, cuenta con soporte para proyectos alojados en Github que es la plataforma de versionamiento que utilizará el proyecto para sus aplicaciones. En la sección 5 se presentará la implementación del Pipeline, donde la fuente de alimentación de este es el código de los proyectos en Github.

4.5.5. Otros

En esta sección se mencionará las tecnologías utilizadas en procesos que no requirieron una gran discusión sobre cual plataforma sobre otra se iba a ocupar.

Para el **versionamiento del código** de las distintas aplicaciones (móviles, web y back end) se utilizó Github [25]. Siendo esta la plataforma abierta de excelencia para alojar proyectos privados y gratuitos.

En cuanto a la plataforma donde se alojó el Kanban con las historias de usuarios, se decidió ir por Trello [43], siendo gratuita, de fácil adaptación y el alumno de este trabajo de título trabajó con esta durante toda su estadía universitaria.

Capítulo 5

Implementación de la solución

En esta sección se presentará la implementación de las funcionalidades de cada aplicación. Además, se verán otras implementaciones relacionadas con la infraestructura, pipeline y despliegue en producción.

5.1. Aplicación móvil del conductor

En la implementación de la aplicación del conductor, se procedió a crear un proyecto en **React Native** siguiendo su documentación oficial [36] en conjunto con **Redux** para manejar un estado local en la aplicación [37], y así poder inicializar un mapa utilizando la librería **react-native-maps** [34]. Con lo anterior configurado, se tiene lo vital para empezar a desarrollar las funcionalidades como se verá a continuación.

5.1.1. Login

Para esta implementación se utilizó **Firebase Authentication** [19], este servicio de Firebase brinda una librería de autenticación en JavaScript. Sin embargo, se prefirió utilizar una librería que adapta la implementación de servicios de Firebase para poder ser utilizada sin problemas en conjunto a React Native [39].

Primero, en Firebase Authetication se habilitó el proveedor **Google** para admitir nuevos usuarios mediante este servicio de autenticación. Luego, se siguió las instrucciones en la página oficial de **RNFirebase** para integrarlo con el proyecto y se procedió a integrar la autenticación con la implementación del diseño mostrado en la parte izquierda de la Figura **5.1**.

Como parte del ciclo de vida de la respectiva historia de usuario, en la validación del diseño, se procedió a evaluar el diseño con un grupo cerrado de usuarios. La retroalimentación principal por parte de los usuarios fue la poca identidad de los colores para ser una *aplicación de transporte* y además de tener un logotipo muy genérico. En base a lo anterior, se procedió a buscar colores y un logotipo más adecuado para el Login de la aplicación del conductor, y este resultado se refleja en la parte derecha de la Figura **5.1**.

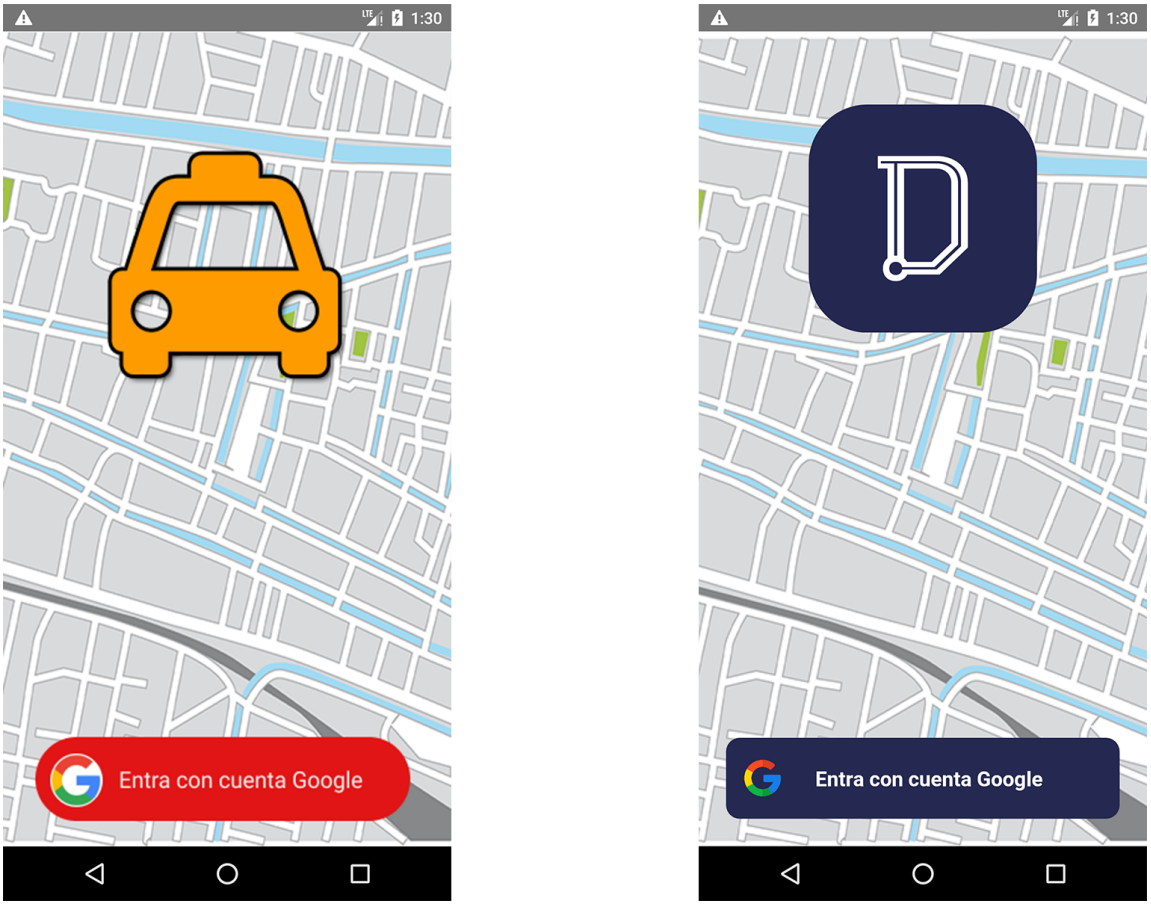

Figura 5.1: Primera versión y versión definitiva del Login de la aplicación del conductor

Para los errores definidos en los criterios de aceptación de la historia de usuario, los relacionados al proveedor de Google fueron directamente implementados gracias a la librería de autenticación de Firebase en JavaScript.

Un caso importante a abordar es cuando un usuario intenta autenticarse en la aplicación y no pertenece a ninguna línea de taxi colectivos. Para este caso, se crea un registro en la base de datos con todos los conductores asociados a una línea de taxi colectivo. Estos registros se encuentran bajo la siguiente entrada de la base de datos.

```
/route_drivers/{route_id}/[{email_driver_1}, ... , {email_driver_N}]

route_id: Identificador de la línea de taxi colectivos.
email_driver_X: Correo electrónico relacionado a la cuenta de Google.
```

Se decidió que el correo electrónico relacionado a la cuenta de Google sea el identificador debido a la fácil integración con la autenticación de este mismo proveedor. Esta entrada de la base de datos es alimentada por la aplicación web administrativa, cuando se ingresa una nueva línea de taxi colectivos y se adjunta los correos de Google de cada conductor. Esto se verá en más profundidad en la sección 5.3.

Finalmente, en el caso exitoso de la autenticación, se crea una entrada en la base de datos no relacional con los datos obtenidos en la autenticación con Google del conductor. Esta entrada esta definida de la siguiente manera.

```
/driver_information/{driver_id}/{driver_obj}

driver_id: Identificador del conductor.
driver_obj: Es un objeto con la información del conductor.
```

La información relacionada al conductor esta definida por: **Nombre, teléfono, URL de la foto de perfil, patente del vehículo e identificador de la línea de taxi colectivos a la cual pertenece**. Donde el teléfono y la patente del vehículo son ingresados por el mismo conductor en la sección de Pérfil dentro de la aplicación y la información restante es proveída por Google. Un ejemplo de este objeto sería de la siguiente forma.

```
{
    name: Rene Dotte,
    phone: 56957541474,
    carLicensePlate: BJRP18,
    photoUrl: https://lh3.googleusercontent.com/a-/AOh14GgakE,
    routeId: 104
}
```

Una vez completado el proceso de autenticación, se guarda localmente en el dispositivo móvil la información del conductor y que está autenticado, esto con el fin de persistir la autenticación y no pedirla cada vez que el usuario cierre la aplicación. Finalmente, la aplicación lleva al conductor a la pantalla inicial donde podrá acceder a las funcionalidades de la aplicación.

5.1.2. Inicio y fin de actividad en recorrido

La funcionalidad de dar inicio y fin a la actividad por parte del conductor nace de la necesidad de brindar a los los pasajeros información en tiempo real sobre los conductores que están activos en la ruta. El principal objetivo de esta funcionalidad es compartir la ubicación del conductor solamente si está activo en el recorrido. Dado esto, se procedió a crear una entrada en la base de datos para compartir la ubicación del conductor solamente si este se encuentra *activo*. La entrada en la base de datos es la siguiente.

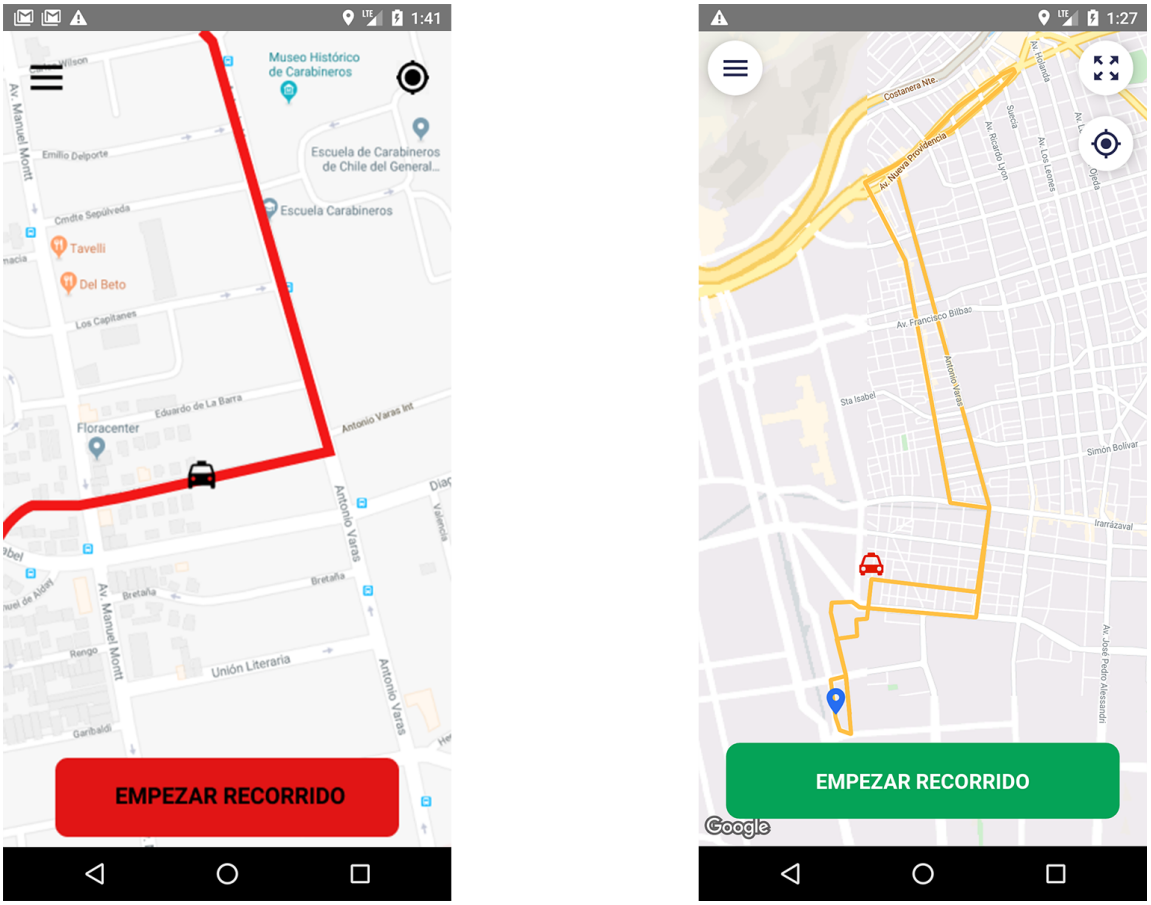

Figura 5.2: Primera versión y versión definitiva de la pantalla inicial de la aplicación del conductor

```
/drivers_location/{driver_id}/{driver_loc}

driver_id: Identificador del conductor
driver_loc: Es un objeto que contiene coordenadas de latitud
y longitud del conductor
```

Esta entrada es alimentada mediante la librería GeoFire [24], la cual brinda herramientas para hacer consultas geográficas que veremos en la sección 5.1.4. GeoFire se encarga de escribir en la base de datos la ubicación del conductor, sin embargo, la ubicación es obtenida mediante el hardware del teléfono utilizando la librería **react-native-geolocation-service** [35].

En la validación del diseño, la retroalimentación de los usuarios fue con respecto a los colores y la consistencia del botón utilizado para finalizar el recorrido, por lo que se procedió a cambiar el color del botón para empezar el recorrido con un color más relacionado a una acción de *"comenzar algo"*. Además, los usuarios tuvieron problemas al momento de activar el **zoom** para ver la ruta completa de la línea, por lo que se procedió a agregar un botón para poder dejar la ruta en primer plano. Estos cambios se pueden apreciar en la Figura 5.2

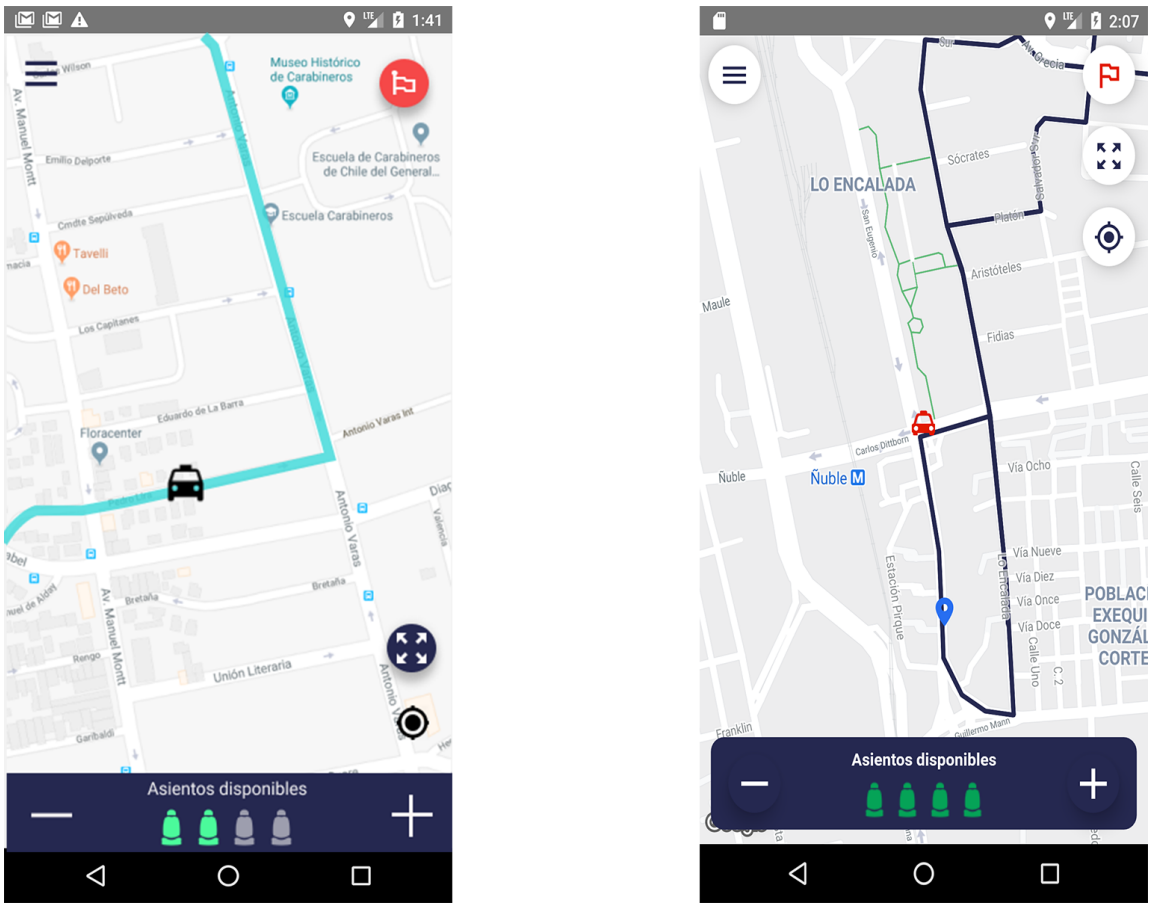

Figura 5.3: Primera versión y versión definitiva de la pantalla con recorrido activo de la aplicación del conductor

Por otro lado, en el diseño para dar fin a la actividad del recorrido, se procedió a dejar con la misma estructura todos los botones y solamente cambiarle el color del icono de la bandera, además de posicionarlos como conjunto en una misma posición como se muestra en la Figura 5.3.

Al momento que el conductor presiona el botón *EMPEZAR RECORRIDO* mostrado en la Figura 5.2, la ubicación es enviada a la base de datos y almacenada localmente en el dispositivo móvil. Además, se procede a guardar un estado de **activo** en la aplicación de manera local, para indicar que la pantalla deba redireccionar a la pantalla con el recorrido activo. De igual manera, cuando el usuario mira recorrido **activo** y presiona el botón con el icono de la bandera en la parte superior derecha de la versión final de la Figura 5.3, se registra el estado **inactivo** y se guarda un objeto con latitud y longitud en **0** para indicar inactividad y el conductor no aparezca en ninguna búsqueda futura, esto se profundizará más en la sección 5.1.4.

A continuación se muestra un ejemplo del objeto guardado en la base de datos no relacional con la información de la ubicación del conductor estando activo e inactivo.

```
Conductor activo

{
    g: 66jc9hbq1c, //geo id
    l: {
        0: -33.4592, //latitud
        1: -70.6194 //longitud
    }
}

Conductor inactivo

{
    g: 66jc9hbq1c, //geo id
    l: {
        0: 0, //latitud
        1: 0 //longitud
    }
}
```

Por último, la actividad del conductor persiste en el caso de que el conductor cierre o minimice la aplicación para que el conductor vea la aplicación en el mismo estado en el que la dejó. Esto se logró consultando si la **latitud** y **longitud** en la base de datos eran distintas de **0** al momento de ingresar a la aplicación. Existen casos cuando el conductor cierra la aplicación y no la vuelve a abrir en un tiempo considerable, este conductor debería desaparecer para los pasajeros. Este tipo de problemas de consistencia de datos, se presentará la solución en la sección 5.4.3.

5.1.3. Manejo de asientos disponibles

Esta funcionalidad fue pensada para informar la cantidad de asientos disponibles a los pasajeros que están en espera en el recorrido. El conductor podrá manejar la capacidad de los asientos solamente cuando el recorrido está activo. El manejo de los asientos se abordó también con una entrada en la base de datos no relacional de la siguiente forma.

```
/seats_occupancy/{driver_id}/{seats_taken}

driver_id: Identificador del conductor
seats_taken: Numero de asientos ocupados
```

Esta entrada es alimentada por el componente mostrado la Figura 5.4, empezando con un valor inicial de 0 asientos ocupados. El conductor podrá indicar cuando un usuario ingresa a su vehículo con el botón etiquetado + y con el botón **-** cuando uno se baja, como se muestra

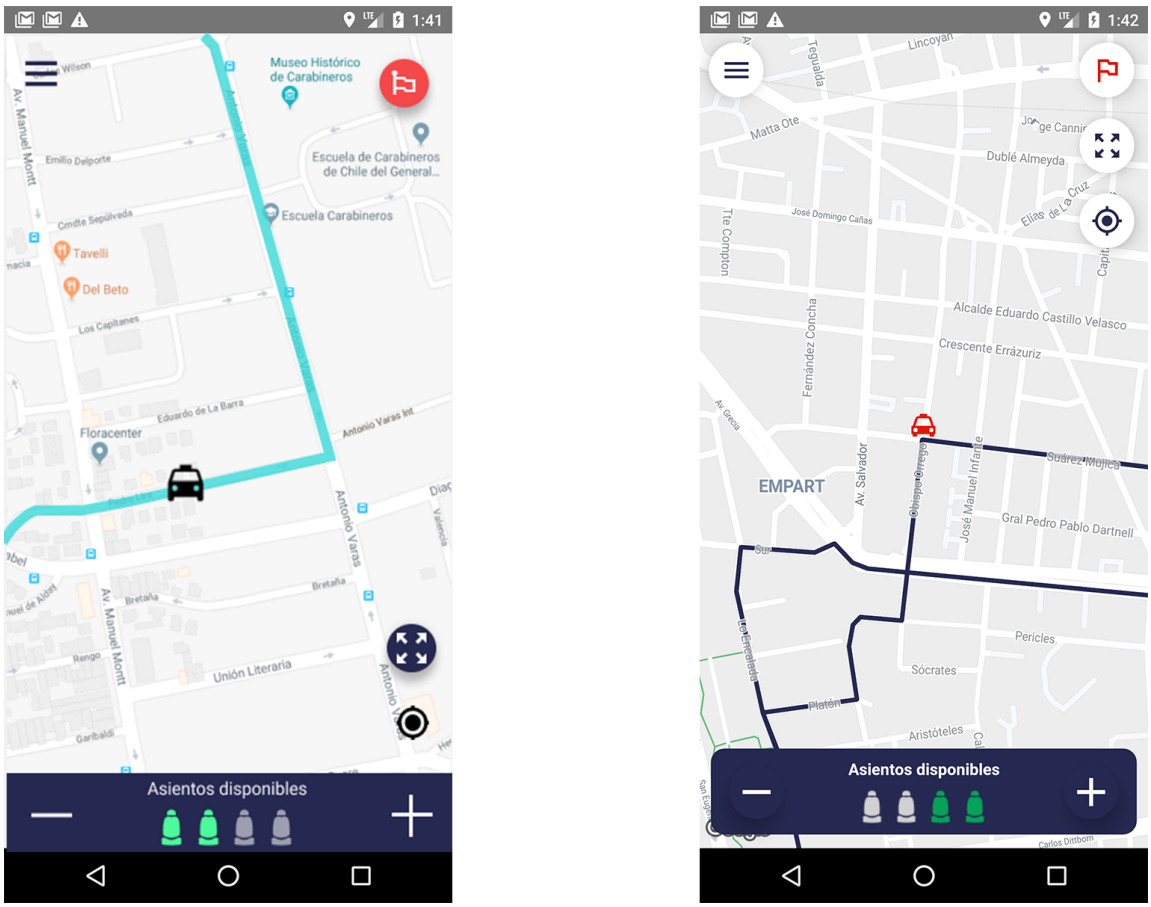

Figura 5.4: Primera versión y versión definitiva de la pantalla con el manejo de asientos de la aplicación del conductor

en la Figura 5.4.

En cuanto a la validación del diseño, los usuarios indicaron que la coherencia de los colores utilizados para indicar que un asiento está ocupado o disponible era confusa, por lo que se estableció que los asientos disponibles fueran marcados por el color verde y un color gris para los ocupados como se muestra en la parte derecha de la Figura 5.4. Además, se cambió el orden de como se marcaban los asientos, en un principio se marcaban los asientos ocupados de derecha a izquierda, pero resultaba más intuitivo para los usuarios que estos se marcaran de izquierda a derecha. Finalmente, los usuarios indicaron que la forma del componente que enmarca el manejo de los asientos podría seguir la consistencia de los bordes redondeados que presenta la mayoría los componentes en la aplicación. El resultado final se puede apreciar en la Figura 5.4.

Finalmente, cuando un conductor pasa de estar **activo** a **inactivo** en la ruta, los asientos ocupados se reinician a **0**, este cambio es reflejado de manera local y en la entrada de la base de datos, sin embargo, si el usuario activo minimiza la aplicación y la vuelve a abrir, los asientos ocupados persisten.

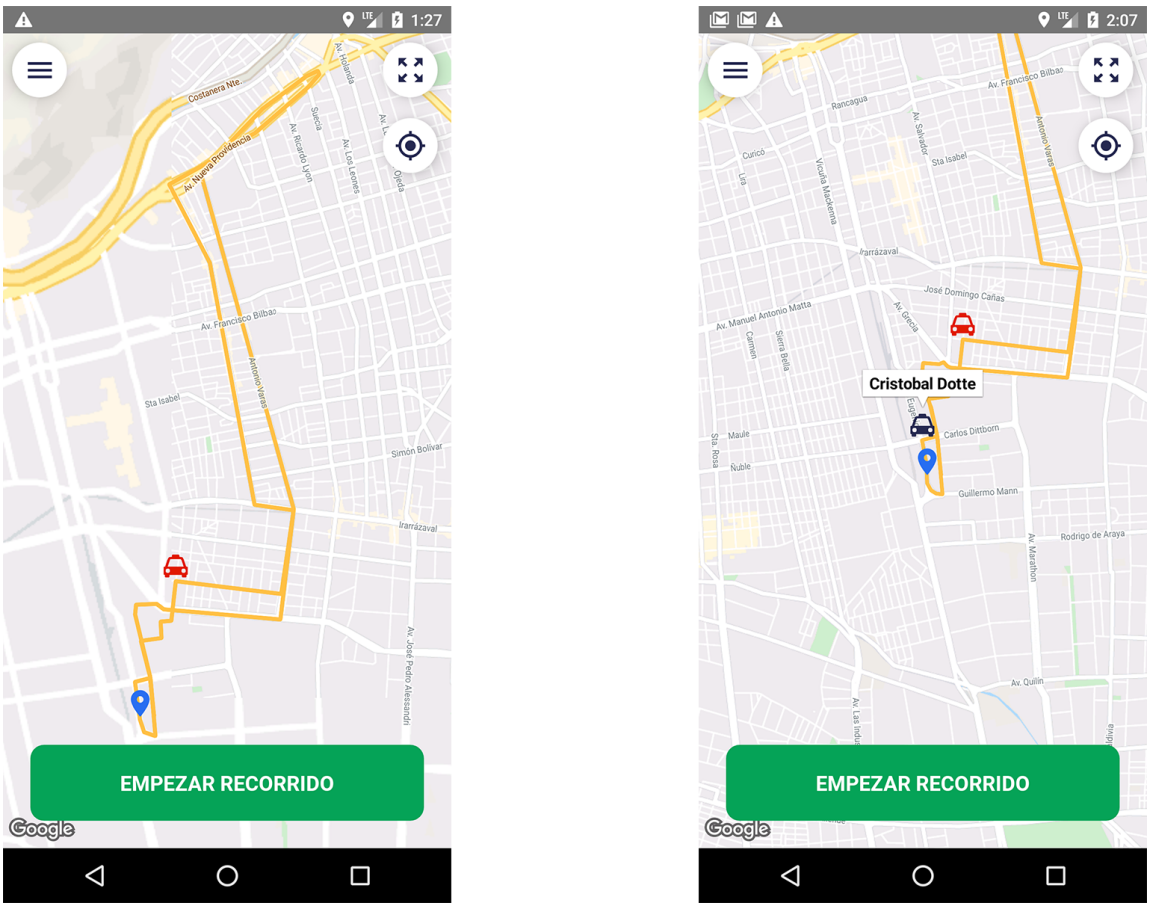

Figura 5.5: Mapa cuando ingresa nuevo conductor activo de la línea

5.1.4. Ubicación de colegas activos y pasajeros en espera

Esta funcionalidad muestra en tiempo real la ubicación de los **colegas activos** y **pasajeros en espera** relacionados al recorrido del conductor. La principal tarea es lograr una consulta geográfica que obtenga a los conductores activos y pasajeros en espera relacionados a la misma línea de taxi colectivos y mostrarlos en el mapa de la aplicación. Para esta tarea se utilizó consultas de GeoFire [15]. En resumen, dada una consulta basada en una entrada en la base de datos no relacional, un radio y un centro, GeoFire brinda tres eventos relacionados a esta consulta geográfica. El primero. cuando se agrega un nuevo componente que cumple con los criterios geográficos (esta dentro del área definida por el centro y radio), el segundo, cuando un componente cambia de ubicación (latitud o longitud) dentro del área y el tercero, cuando un componente ya no cumple los criterios geográficos (sale del área).

Colegas activos

Para la consulta de los colegas activos en ruta, se utilizó la entrada `drivers/location` de la base de datos. Esta entrada define a qué elementos de la base de datos se aplicará la consulta.

El segundo componente de la consulta es el **centro** (latitud y longitud). Este centro se utiliza en conjunto con el **radio** para filtrar las ubicaciones obtenidas en la entrada de la base

de datos. El centro se calcula como un promedio de las coordenadas de la ruta del recorrido al cual pertenece el conductor. Este recorrido está definido como una lista de puntos geográficos (latitud y longitud) que definen el trazo del recorrido en un mapa. Por otro lado, el radio se calcula como la máxima distancia del centro a un punto del recorrido, garantizando que todo el recorrido este dentro de la consulta y que los conductores que se salgan de esta área definida no se obtengan con esta consulta. Esto explica por qué en la sección 5.1.2 se deja los valores de latitud y longitud en **0** cuando el conductor se pone como inactivo en la ruta (no aparece ni para los pasajeros esperando ni para los colegas activos en ruta).

El tercer componente son los eventos que se definen en la consulta. Estos eventos son definidos una vez y quedan activos hasta que la aplicación se cierre. El primer evento registra por primera vez los conductores que se encuentran bajo la entrada de la base de datos que cumplen con los criterios geográficos (están dentro del área definida por el centro y radio). Al entrar un nuevo conductor, se obtiene el identificador de este mediante la entrada nueva definida por `drivers/location/{driver_id}` y la ubicación almacenada en esta entrada (latitud y longitud). Luego, se procede a hacer un filtro que verifica si este conductor pertenece al mismo recorrido del usuario conductor, utilizando el identificador y la entrada de la base de datos `driver_information/driver_id/{route_id}` el cual da el identificador del recorrido al cual pertenece el conductor encontrado. Este se compara con el identificador de la ruta, y finalmente, si este pertenece al mismo recorrido se procede a agregarse a una lista de colegas activos a mostrarse en el mapa, esto se puede apreciar en la Figura 5.5.

El segundo evento es cuando se registra un cambio de la latitud o longitud de un conductor que ya estaba en los criterios geográficos de la consulta. En este caso, se procede a moverlo a la nueva ubicación en el mapa.

El último evento, es cuando un conductor se sale del área de la consulta. En este caso, se procede a eliminarlo de la lista de conductores a mostrar en el mapa. Esto puede ocurrir cuando un conductor finaliza su ruta (inactivo) y deja su latitud y longitud en **0**, o simplemente si se equivocó y se alejó de su recorrido.

Pasajeros en espera

Esta funcionalidad fue pensada para que los conductores vean a los pasajeros en espera en su recorrido y puedan tomar decisiones en base a la posición de estos (salir a la ruta en un determinado tiempo). Para implementarlo se utilizó la entrada `passengers_location` de la base de datos. En esta se almacenan las ubicaciones de los pasajeros en espera (se puede ver más en profundidad en la sección 5.2.2). Sobre esta entrada se aplicará la consulta con el mismo radio y centro definidos anteriormente en los **colegas activos**, debido a que abarca lo que es el recorrido del conductor en un área circular, lo que nos sirve para la consulta de los pasajeros también.

Para ver si un nuevo pasajero que cumple con los criterios geográficos, se procede a verificarse si este está esperando a la línea relacionada al conductor. Para esto, se creó una entrada en la base de datos que guarda la información de los pasajeros esperando para una ruta específica. Esto se define de la siguiente manera en la base de datos.

```
/passengers_waiting/{route_id}/[{passenger_id_1}, ... , {passenger_id_N}]

route_id: Identificador de la ruta
passenger_id_X: Identificar del pasajero esperando en la ruta
```

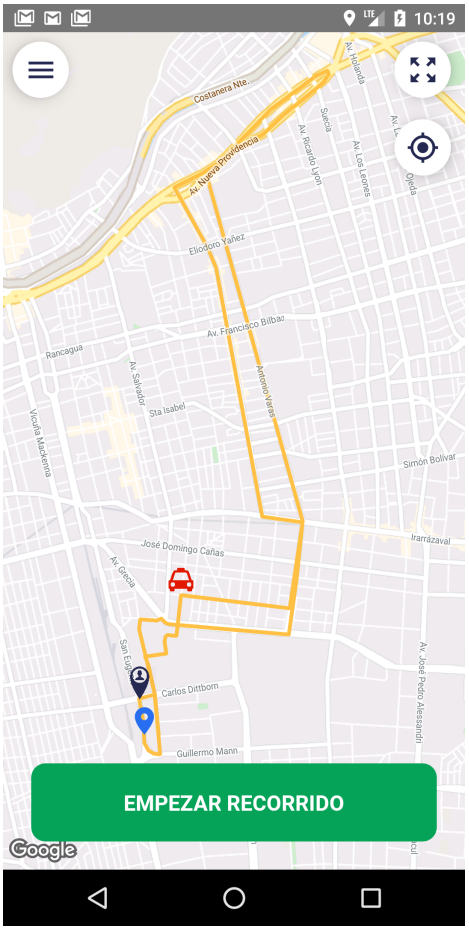

Figura 5.6: Pasajero en espera en el mapa

Si este nuevo pasajero se encuentra registrado con esta entrada de la base de datos, se procede a agregarse a una lista de pasajeros para mostrarse en el mapa. Esto se puede apreciar en la Figura 5.6.

Por último, con los dos eventos restantes se procede de igual manera que con los colegas activos. Cuando el pasajero se mueve dentro del área de la consulta, se procede a actualizar la ubicación de este dentro del mapa. Cuando el pasajero sale del área de la consulta, se elimina de la lista de pasajeros en espera a mostrar en el mapa.

5.1.5. Perfil del conductor

La funcionalidad del perfil del conductor fue pensada para compartir información del conductor a los pasajeros y colegas en el recorrido. Para esto se agregó una sección en el **Menú** de la aplicación mostrada en la Figura 5.9 en donde se puede acceder al **Perfil**.

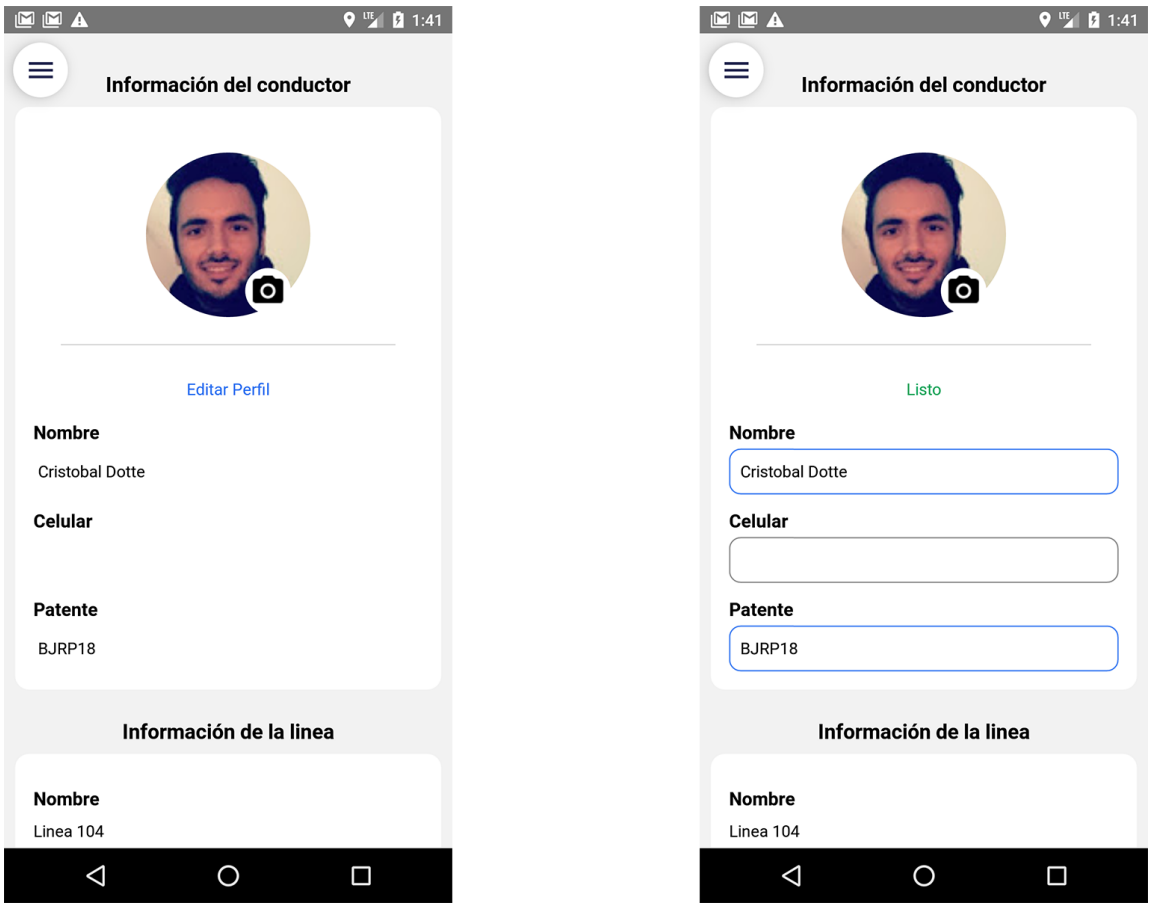

Figura 5.7: Flujo de edición en pantalla del perfil del conductor

Durante la validación del diseño, los usuarios indicaron que se podría seguir con la misma línea de colores que se han estado utilizando. Por lo tanto, se procedió a cambiar los colores utilizados en el botón de edición y de modificación en la pantalla del perfil del conductor como se muestra en la Figura 5.8.

Para modificar la información del conductor, el usuario conductor procederá a presionar el botón **Editar**, dando la posibilidad de modificar su nombre, teléfono, patente del vehículo como se muestra en la Figura 5.7. Esta información del perfil del conductor se almacena bajo la entrada `drivers_information/{driver_id}` siendo el `driver_id` el identificador del usuario conductor que esta modificando sus datos. Esta entrada es creada en el proceso de autenticación y es modificada bajo este flujo.

En cuanto a la modificación de la foto del conductor, el usuario conductor procederá a presionar el botón con el icono de la cámara, llevándolo al flujo de obtención de la foto. Para lograr esto, se utilizó la librería **react-native-photopicker**, la cual maneja la lógica de pedir permisos al usuario para tomar una foto o elegir una de la galería para luego mostrar el flujo nativo del dispositivo para obtener la foto. Una vez con la foto obtenida, se procede a subir este archivo a **Firebase Storage** [20]. El resultante de este proceso es la modificación de la siguiente entrada en la base de datos.

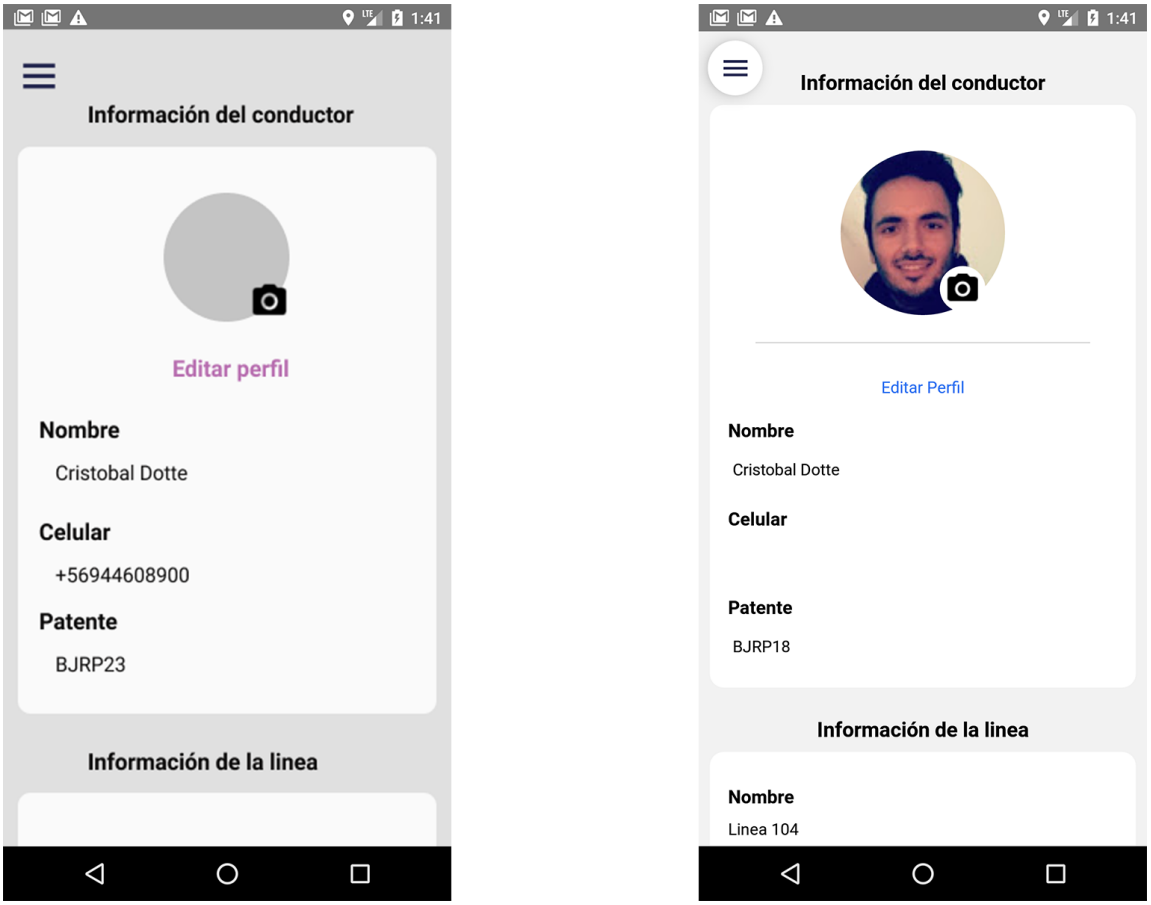

Figura 5.8: Primera versión y la versión final del perfil del conductor

```
/drivers_information/{driver_id}/{photo_url}

driver_id: Identificador del conductor
photo_url: URL del archivo almacenado en Firebase Storage
```

En la parte inferior de la sección del perfil del conductor se muestra también información relacionada a la línea de taxi colectivos a la cual pertenece, esta información esta dada por nombre de línea, horario de funcionamiento y tarifa. La información relacionada a la línea el conductor no tiene acceso a editarla, de hecho, la única forma de editar esta información es mediante la aplicación web administrativa. Todo lo relacionado a la información de la línea de taxi colectivo se vera en profundidad en la sección 5.3.

Finalmente, la información relacionada al conductor y la línea de taxi colectivos a la cual pertenece se almacenada de manera local en el dispositivo, para no tener que consultar esta información a la base de datos cada vez que el usuario la necesite. Esto es una buena practica para datos que no cambian mucho en el tiempo, como lo es la información del conductor y su línea (casi nunca cambia).

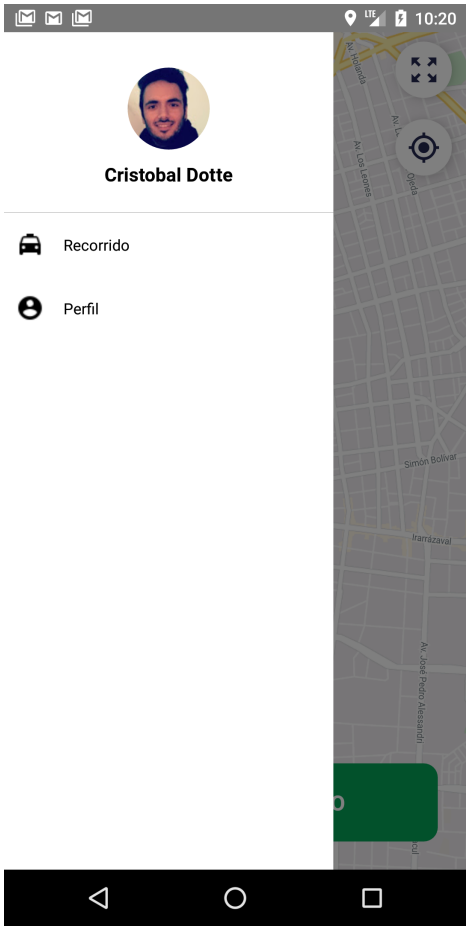

Figura 5.9: Menu del perfil del conductor

5.2. Aplicación móvil del pasajero

En la implementación de la aplicación del pasajero, al igual que la del conductor, se procedió a crear un proyecto en **React Native** siguiendo su documentación oficial [36] en conjunto de **Redux** para manejar un estado local en la aplicación [37], y así poder inicializar un mapa utilizando la librería **react-native-maps** [34]. Con lo anterior configurado, se tiene lo necesario para empezar a desarrollar las funcionalidades como se verá a continuación.

5.2.1. Autenticación anónima

En la aplicación del pasajero no existe un **Login** para identificar al usuario pasajero desde el punto de vista de los usuarios (conductor y pasajeros), sin embargo, se hizo una implementación para poder identificarlos dentro del sistema, asociándole un `passenger_id` que es usado en distintas implementaciones relacionadas a la base de datos como veremos en durante el desarrollo de la aplicación móvil del pasajero.

Esta implementación se hizo bajo **Firebase Anonymous Authentication** [18], resultando una herramienta muy útil para poder tener una autenticación *rápida* de los usuarios que utilizan la plataforma. Esta versión de autenticación tiene como principal cualidad que puede transformarse en una autenticación de otro tipo cuando se estime conveniente, como

por ejemplo, una con cuenta **Google**, manteniendo el `passenger_id` y el historial de este usuario desde que fue **anónimo** hasta que pasa a ser un usuario con autenticación de **Google** ganando todo lo que conlleva tener una cuenta **Google** asociada (nombre, foto de perfil, correo de Google, entre otros).

5.2.2. Buscador de recorridos

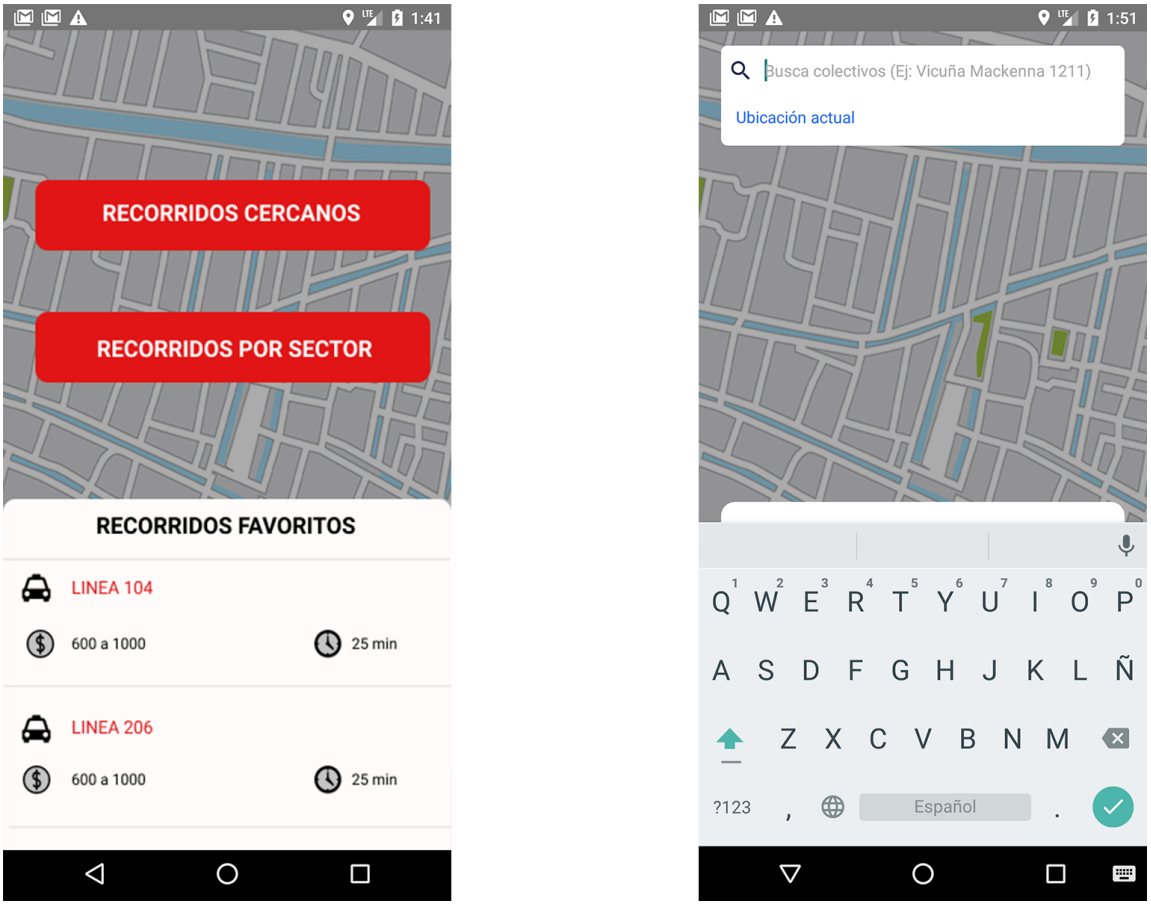

Figura 5.10: Primera versión y versión final del buscador de recorridos en la aplicación del pasajero

Esta funcionalidad nació en base a la poca información existente de las líneas de taxi colectivos y sus recorridos como se muestra en la sección 3.2. Para implementar esta funcionalidad se buscó dar respuesta a las preguntas: ¿Existirán recorridos cercanos a mi ubicación que me puedan llevar a mi destino? y ¿En que lugares hay recorridos de taxi colectivos?.

El primer acercamiento a esta funcionalidad se puede ver en la parte izquierda de la Figura 5.10 en donde se puede apreciar dos opciones que derivaban a una búsqueda en base a distintos parámetros. Las dos opciones eran **Recorridos cercanos** y **Recorridos por sector** para buscar recorridos en base a la ubicación actual del pasajero y para buscar por un sector que no estuviese cerca el pasajero respectivamente. En base a la retroalimentación de los usuarios se procedió a cambiar la forma de abordar este problema a una manera similar a como lo hacen las aplicaciones modernas (Uber, Beat, Cabify, etc) mediante un buscador por lugares, dando como primera opción para buscar, la ubicación actual del pasajero como

se aprecia en la parte derecha de la Figura 5.10.

Para llevar a acabo esta funcionalidad se utilizó la **API Places Search** [3] de Google Maps para buscar lugares mediante el ingreso de una dirección en un campo de búsqueda. El resultado que se obtiene de la búsqueda de un lugar es un punto `{latitud, longitud}` que se procede a utilizar para buscar los recorridos cercanos a este. Esta búsqueda se hace utilizando una función implementada en el back end llamada `findNearRoutes` la cual recibe como parámetro un punto `{latitud y longitud}` y retorna una lista de recorridos cercanos almacenados en la entrada de la base de datos `route_directions`, esta función se explica en profundidad en la sección 5.4. Cabe destacar que para buscar los recorridos cercanos, se utiliza un valor por defecto llamado **Ubicación actual** que el usuario puede seleccionarlo con lo que se envía el punto `{latitud, longitud}` obtenido mediante el hardware del celular (posición actual).

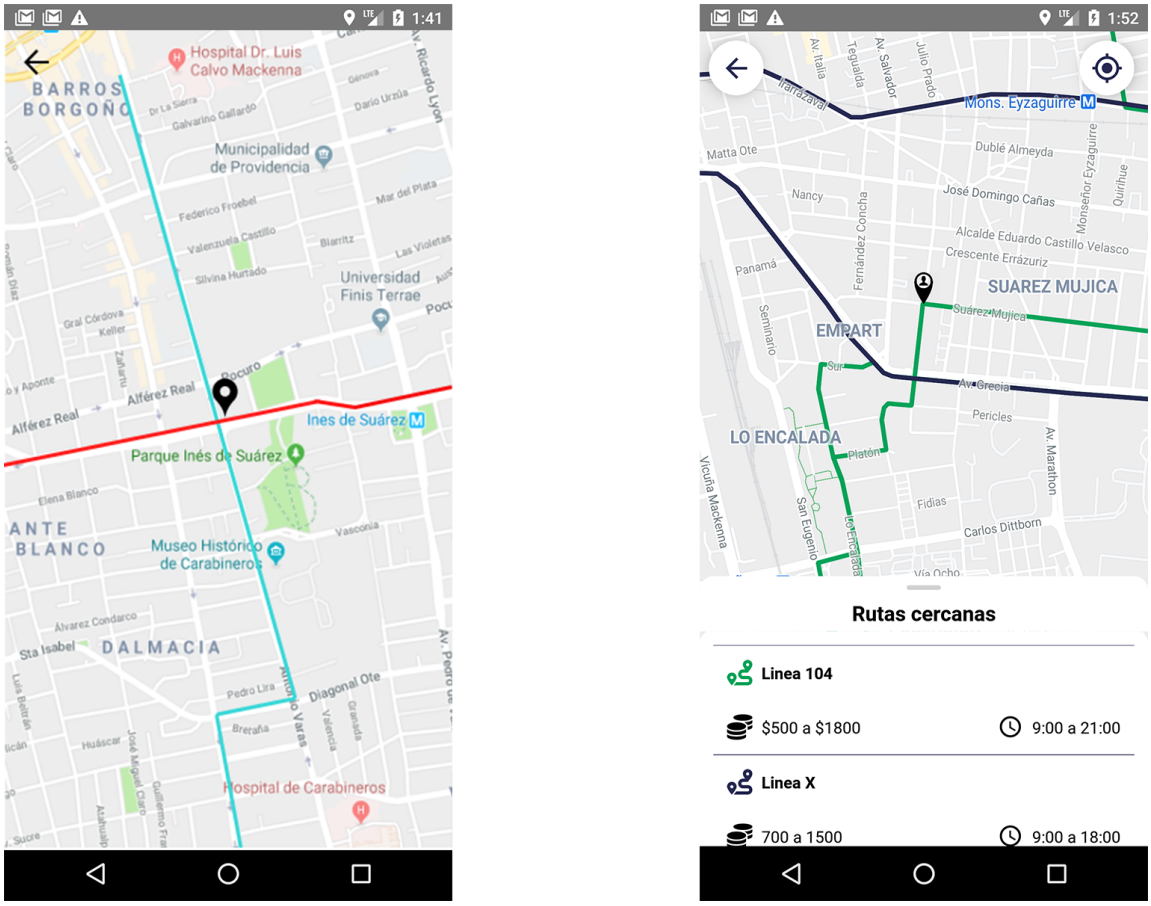

Figura 5.11: Primera versión y versión final del resultado de recorridos cercano a un punto en la aplicación del pasajero

Una vez obtenidos los recorridos se procedió a crear una forma de visualizarlos en un mapa. La primera versión para mostrar los recorridos resultantes es mostrada en la parte izquierda de la Figura 5.11. El principal problema que se encontró en la validación con los usuarios fue la poca información de qué hacer en la pantalla más allá de moverse en el mapa. Por se implementó una funcionalidad para que el usuario pueda ingresar al recorrido de una línea en especifico y ver su información de mejor manera, agregando una lista con un extracto

de la información de la línea, dando como indicio la posibilidad de que los elementos de esta lista puedan ser seleccionados para saber más del recorrido en cuestión como se muestra en la parte derecha de la Figura 5.11.

Finalmente, cuando un usuario pasajero selecciona un recorrido de la lista, la aplicación lo redirige a la sección del Perfil del recorrido, presentado en la siguiente sección.

5.2.3. Perfil del recorrido

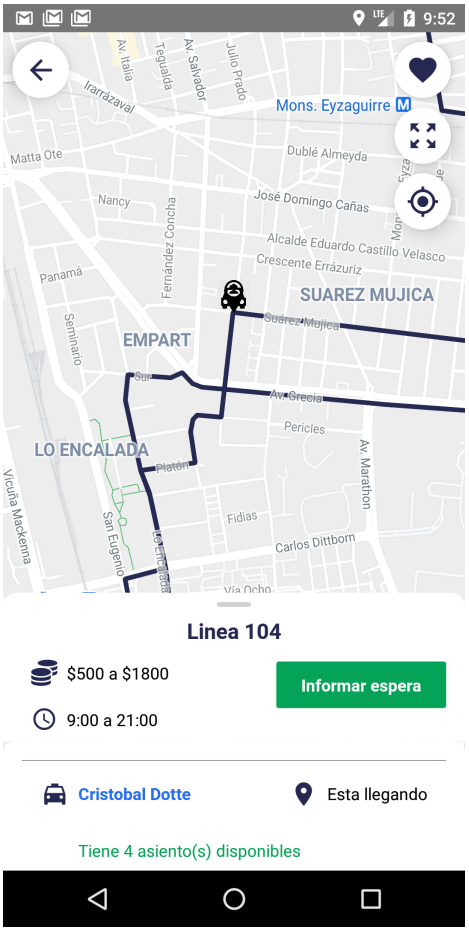

Figura 5.12: Perfil del recorrido en la aplicación del pasajero

Esta funcionalidad se creó para que el usuario pasajero pueda interactuar con un recorrido de una línea de taxi colectivo. Las principales responsabilidades de esta funcionalidad es informar dónde vienen los conductores activos en la ruta, agregar y eliminar de favoritos (sección 5.2.3), informar espera en el recorrido y ver información general del recorrido.

En la validación del diseño, los usuarios se encontraron bastante satisfechos con la primera versión de la funcionalidad, por lo tanto, se procedió a implementarla como se muestra en la Figura 5.12. Los componentes principales de esta funcionalidad son una lista en la parte inferior de la pantalla con la información de los conductores activos en ruta, ordenados por la distancia que se encuentran del usuario pasajero. Esta lista también sirve para poder ubicar al conductor en el recorrido, esto se logra presionando un elemento de la lista que hace

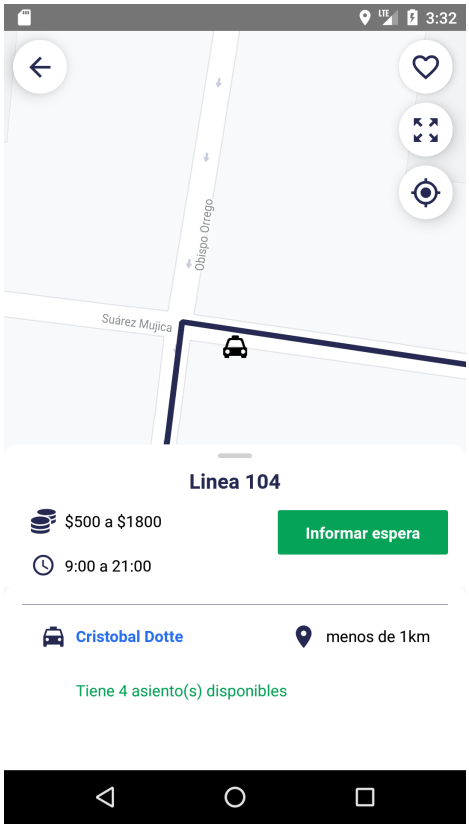

Figura 5.13: Zoom en un conductor en perfil del recorrido en la aplicación del pasajero

referencia a un conductor y el mapa hace un enfoque en el punto geográfico que se encuentra este conductor como se aprecia en la Figura 5.13.

Esta lista de conductores se alimenta de la misma forma que se implementó la funcionalidad de ubicación de colegas activos y pasajeros en espera presentado en la sección 5.1.4, esto quiere decir que existe una consulta con un radio y un centro que se subscribe a eventos en la entrada de la base de datos `drivers_location`. Si entra un nuevo conductor al área de la consulta, este es agregado a la lista, si sale de la consulta, se elimina de la lista, y si se mueve dentro del área de la consulta, se actualiza la ubicación y la distancia entre el conductor y el usuario pasajero. La implementación de la ubicación de los conductores activos en el mapa utiliza el mismo componente que se utiliza en la aplicación del conductor para agregar, eliminar y modificar la posición de los conductores.

En esta funcionalidad se le agregó dos botones para manipular el mapa como se muestra en la parte superior derecha de la Figura 5.12. Estos botones tienen como función enfocar al usuario pasajero y enfocar el recorrido completo. Esto se hizo para que los usuarios puedan ver todo el recorrido y sus conductores activos con tan solo presionar un botón, para luego volver a su ubicación presionando el otro botón y esperar que llegue el conductor a la ubicación donde se encuentra. Existe otro botón en la parte superior derecha de la Figura 5.12 con un icono de corazón para agregar y eliminar de los favoritos el recorrido, lo cual se verá en profundidad en la sección 5.2.3.

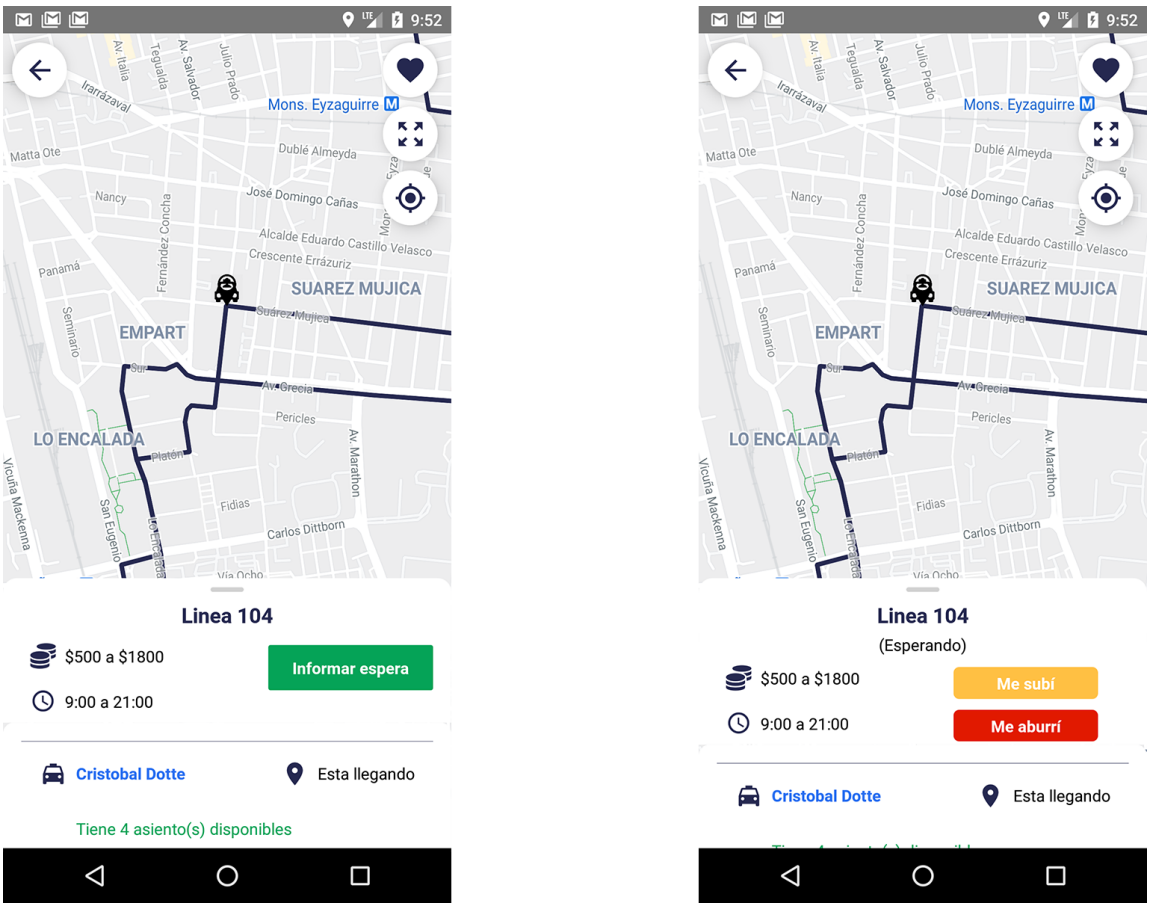

Figura 5.14: Pantallas de espera en la aplicación del pasajero

Informar espera en el recorrido

Para satisfacer la necesidad de informar a los conductores que existen pasajeros esperándolos en el recorrido, se procedió a crear una funcionalidad dentro del perfil del recorrido.

Para informar su espera, el usuario pasajero deberá presionar el botón **Informar espera** y pasará a un estado de espera dentro del recorrido como se aprecia en la Figura 5.14. Esta acción procede a guardar un estado **activo** locamente en el dispositivo, y se registra con la siguiente entrada a la base de datos.

```
/passenger_waiting/{route_id}/{passenger_id}/{timestamp}

route_id: Identificador de la línea
passenger_id: Identificador del pasajero
timestamp: unidad de tiempo en que ocurre la espera
```

Esta entrada registra los pasajeros que están esperando en el recorrido `route_id` y la hora ala que empezaron a esperar. La hora es importante para tener una referencia al momento de encontrar pasajeros que nunca dejaron de esperar o llevan un tiempo muy largo de espera y se deben limpiar de la base de datos, esto se puede ver en profundidad en la sección 5.4.3.

Para que el usuario deje de esperar tiene dos opciones, la primera es cancelarlo con los botones indicados en la parte derecha de la Figura 5.14 o apretando el botón para regresar atrás en la misma pantalla, en la cual le saldrá una advertencia de que si desea continuar con la acción a pesar de que se terminará el estado de espera. Al momento de terminar la espera, se elimina la entrada `/passenger_waiting/{route_id}/{passenger_id}` y se marca como inactivo en el estado local del dispositivo.

5.2.4. Recorridos favoritos

Esta funcionalidad se pensó para los usuarios frecuentes de taxi colectivos dado que brinda un fácil acceso en el inicio de la aplicación a los recorridos que frecuentemente utilizan los pasajeros.

En la validación del diseño, solamente se hizo un ajuste en la altura y ancho del componente que contiene la lista de los favoritos como se muestra en la Figura 5.15. Además, se puede apreciar, que se agregó el horario de funcionamiento en la descripción en los recorridos mostrados en la lista de favoritos, para dar una información más completa de fácil acceso.

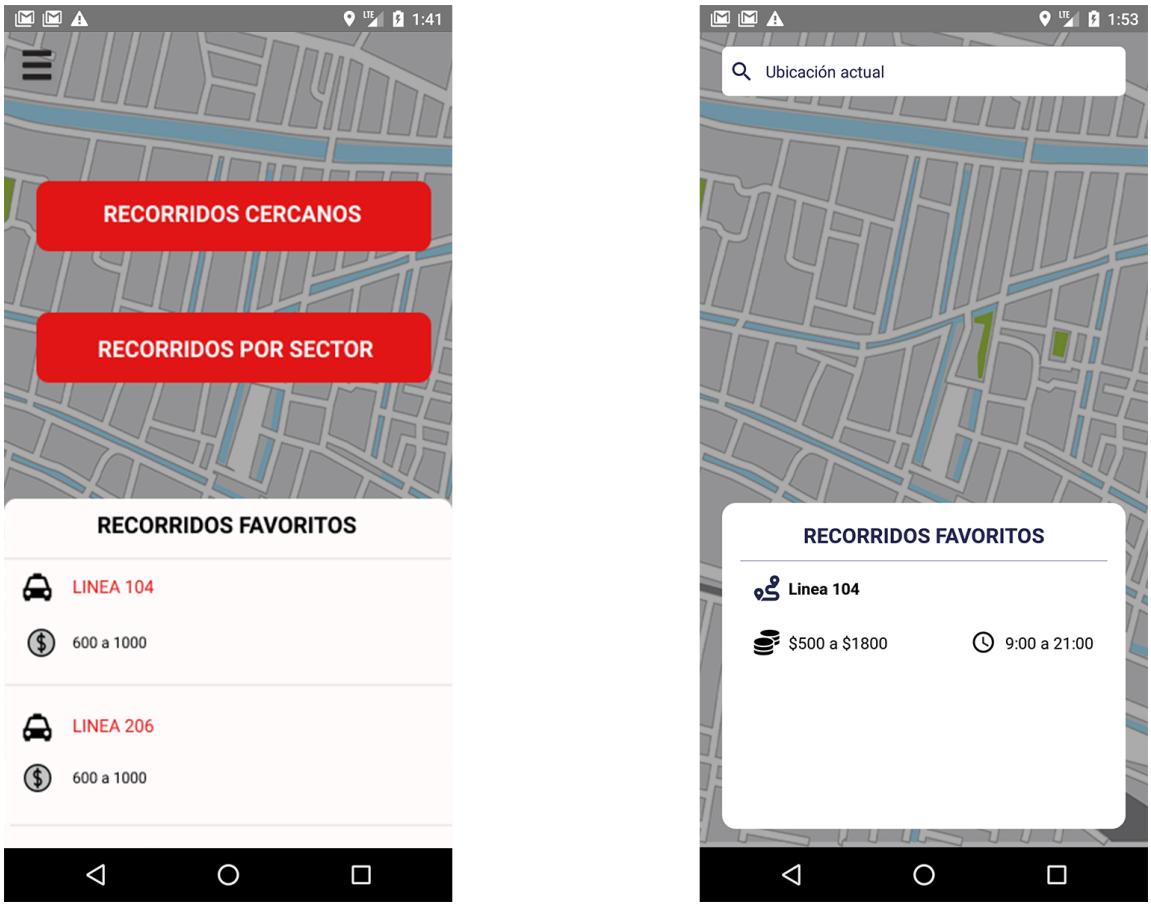

Figura 5.15: Primera versión y versión final de favoritos en la aplicación del pasajero

Esta lista de favoritos está guardada de manera local en el dispositivo, por lo tanto no existe ninguna entrada en la base de datos que almacene los favoritos relacionados a los pasajeros. El único problema con guardarlo de manera local, es que si un pasajero borra la

aplicación y la vuelve a instalar, perderá la lista de favoritos guardada previamente. Se decidió hacerlo así debido a que a la única aplicación que le sirve mantener registro de los favoritos de un pasajero, es la aplicación de los pasajeros, por lo tanto no existe otro componente tratando de obtener esta información mediante una consulta a la base de datos.

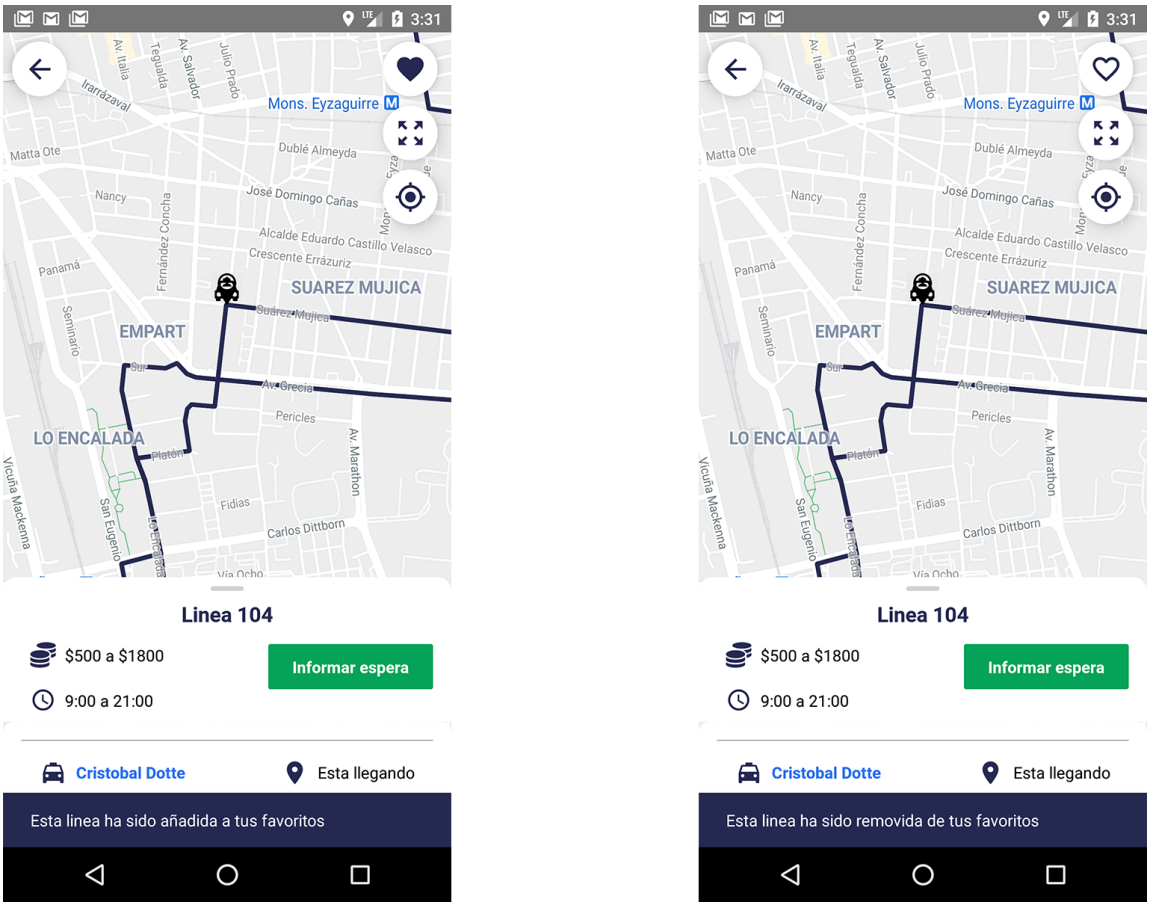

Figura 5.16: Agregar y eliminar favoritos en la aplicación del pasajero

Para agregar o eliminar un recorrido a la lista de favoritos, se agregó un botón con icono de corazón en el **Perfil del recorrido** como se puede apreciar en la Figura 5.16.

5.3. Aplicación web administrativa

Esta aplicación nació en base a la necesidad de mantener un control administrativo sobre las líneas de taxi colectivos. Las principales tareas de esta aplicación es poder agregar, modificar y eliminar líneas de taxi colectivos.

5.3.1. Login

La implementación del Login en la aplicación web administrativa nació de la necesidad de que solamente ciertos usuarios administradores puedan ingresar a la plataforma. En la implementación, siendo consistente con las implementaciones de autenticación en las aplicaciones móviles, se utilizó **Firebase Authentication** en conjunto con el proveedor de Google.

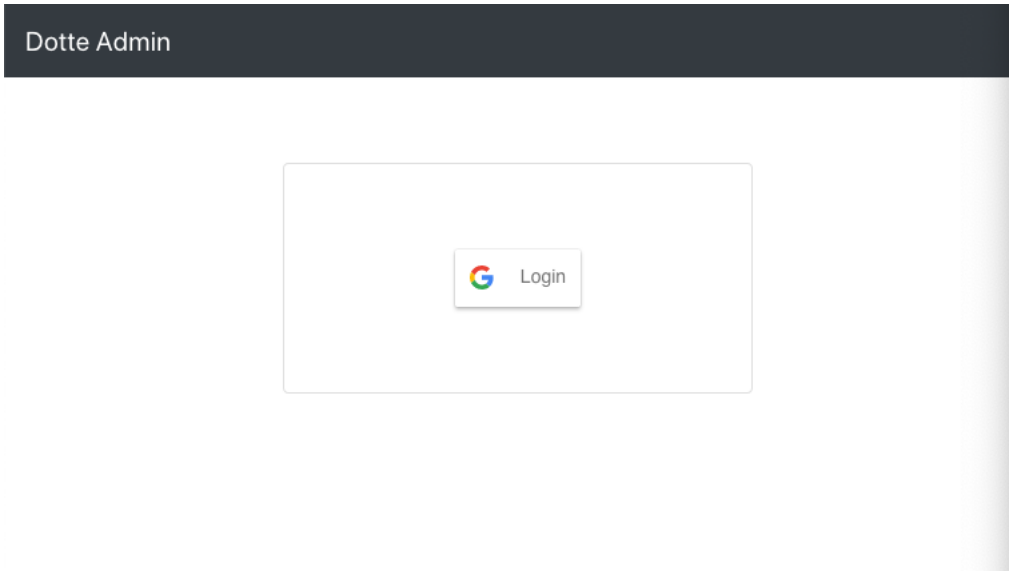

Figura 5.17: Login de la aplicación web administrativa

Firebase brinda una librería en JavaScript para aplicaciones web [19], la que se procedió a utilizar.

El diseño de esta funcionalidad se hizo bajo criterios minimalistas, como se muestra en la Figura 5.17, debido a que el único usuario administrador vendría siendo el alumno de este trabajo de título.

El acceso a esta aplicación está restringido por una lista de correos de Google que identifica a los usuarios administradores, la que actualmente consta solo de del correo de Google del alumno desarrollador. Al autenticarse correctamente un usuario, se guarda un estado en la aplicación web que permite acceder a las funcionalidades de esta.

Luego de autenticarse exitosamente el usuario, se le redirige a la pantalla inicial de la aplicación presentada en la Figura 5.18, donde se muestra una lista de líneas de taxi colectivos ingresadas al sistema.

5.3.2. Nueva línea de taxi colectivos

En la pantalla inicial hay un botón para agregar una nueva línea como se muestra en la Figura 5.18 al final de la lista de líneas de taxi colectivos. Este botón redirige al usuario administrador a una vista donde deberá rellenar un formulario que contiene tres secciones: **información general de la línea**, **conductores asociados a la línea**. **trazado de ruta de la línea**.

La primera sección es sobre la información general de la línea, que consta del nombre, horario de funcionamiento y tarifa de la línea como se presenta en la Figura 5.19. El resultado de enviar este formulario es alimentar la siguiente entrada en la base de datos no relacional.

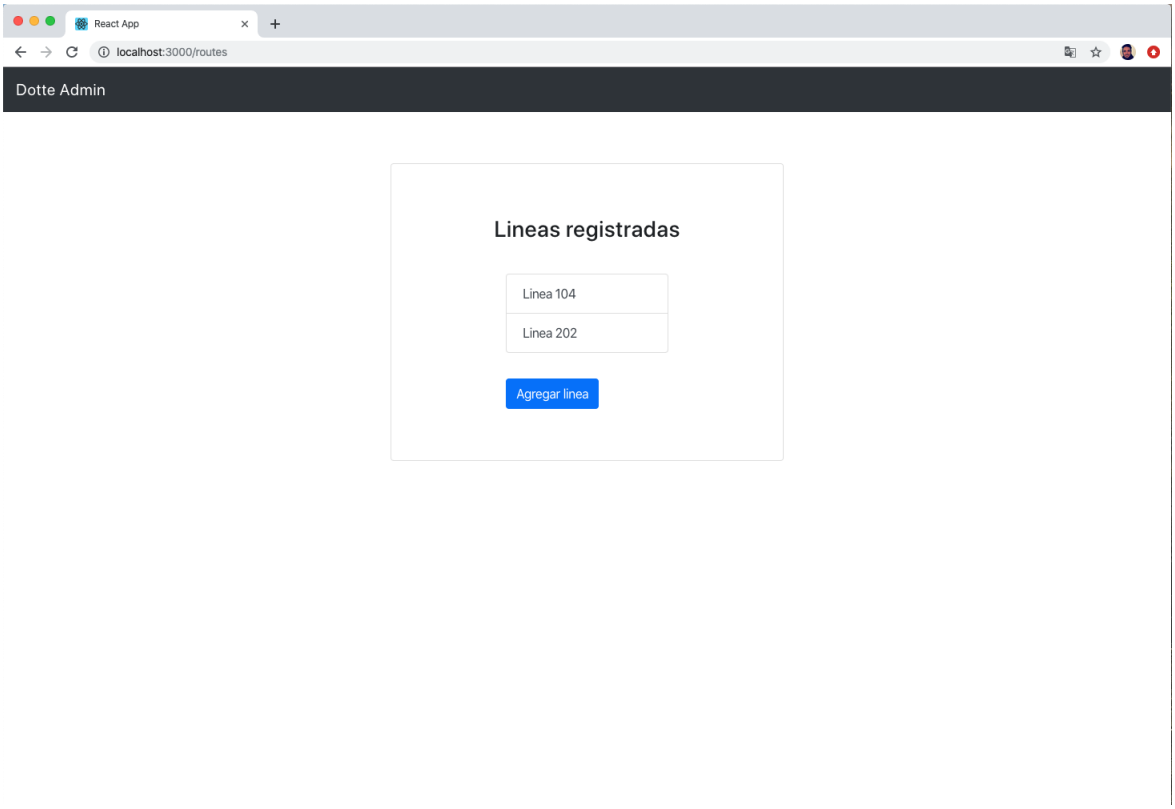

Figura 5.18: Pantalla inicial de la aplicación web administrativa

```
/route_information/{route_id}/{route_obj}

route_id: Identificador de la línea
route_obj: Obtejo con información general de la línea
```

Un ejemplo del objeto con información general de la línea es el siguiente.

```
{
    name: línea 104,
    workHours: {
        from: 10:00,
        to: 20:00
    },
    priceInformation: {
        from: 600,
        to: 1200
    }
}
```

En la segunda sección se asocian usuarios conductores a la línea que se esta ingresando

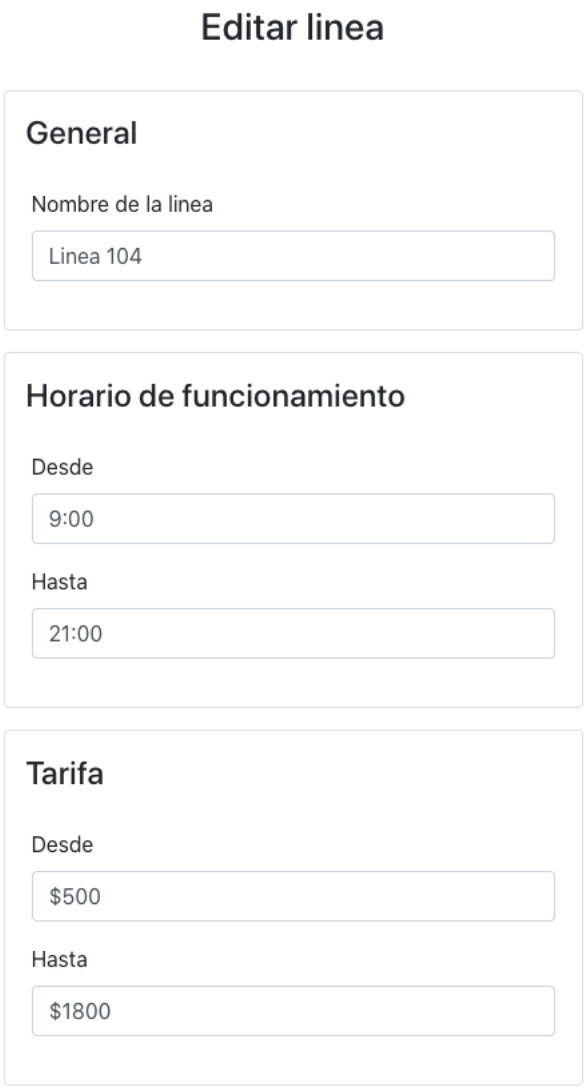

Figura 5.19: Sección información general en agregar nueva línea en la aplicación web administrativa

mediante una cuenta de correo de **Google**, esto se decidió en base a la fácil integración para reconocer a un conductor en el proceso de autenticación con el proveedor de Google mostrado en la sección 5.1.1. La interfaz utilizada para agregar nuevos correos de conductores asociados a la línea de taxi colectivos es una lista que se va expandiendo mediante el texto **Agregar** de color celeste mostrado en la Figura 5.20. La entrada de la base de datos alimentada por esta sección del formulario es la siguiente.

```
/route_drivers/{route_id}/[{email_driver_1}, ... , {email_driver_N}]

route_id: Identificador de la línea de taxi colectivos.
email_driver_X: Correo electrónico relacionado a la cuenta de Google.
```

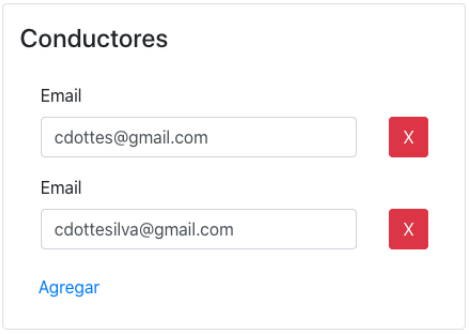

Figura 5.20: Sección correos de Google de conductores de la ruta en agregar nueva línea en la aplicación web administrativa

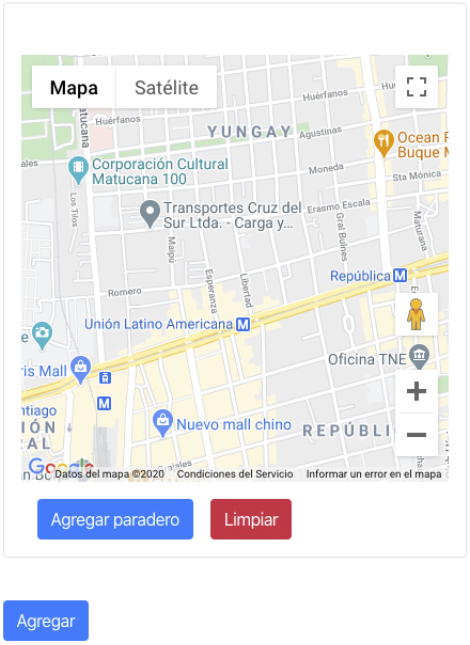

Figura 5.21: Sección trazado de la ruta en agregar nueva línea en la aplicación web administrativa

En la tercera sección se agrega el trazado de la ruta en el mapa del recorrido de la línea de taxi colectivo. La interfaz utilizada para agregar el trazado es un mapa al final de la sección de agregar nueva línea de taxi colectivos como se muestra en la Figura 5.21. Para la lógica de armado del trazado se utilizó un enfoque de ingresar **puntos relevantes**, de tal manera que estos puntos tienen una latitud y longitud por donde el conductor debe pasar para cumplir con la ruta de su línea. La principal función de estos puntos es crear la ruta mediante la API Directions [2] disponible en la librería en JavaScript de Google Maps. La creación del trazado de la ruta cuenta con las siguientes funcionalidades.

1. **Agregar punto relevante.** Para agregar un punto relevante, el usuario administrador debe presionar con el botón secundario del mouse. Al agregar dos puntos relevantes o más, se procede a utilizar la API Directions para generar el trazado entre los puntos relevantes agregados, creándose una ruta en el mapa para la unión de los puntos

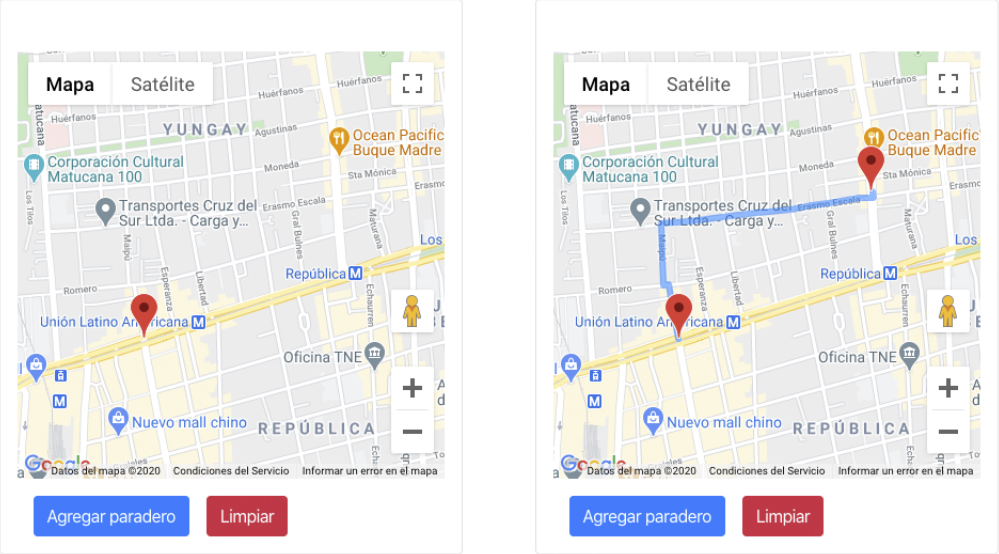

Figura 5.22: Agregar punto relevante en el trazado de la ruta en la aplicación web administrativa

relevantes como se muestra en la Figura 5.22.

2. **Eliminar punto relevante.** Para eliminar un punto relevante, el usuario administrador debe presionar con el botón primario del mouse un punto relevante en el mapa. Al eliminar un punto relevante, se procede a generar una nueva ruta con la API Directions con los puntos relevantes restantes como se muestra en la Figura 5.24.

3. **Cerrar ruta.** Estas rutas son circulares, es decir, siempre se volverá a pasar por el punto relevante inicial. Por lo tanto, para cerrar la ruta, el usuario administrador debe presionar el punto relevante con el botón primario del mouse como se muestra en la Figura 5.23.

4. **Limpiar ruta.** Se agregó un botón para limpiar todos los puntos relevantes y la ruta trazada por la API Directions en el mapa, es decir, lo deja en su estado inicial.

Finalmente, con la ruta trazada por API Directions, se procede a guardar el arreglo de puntos que en su conjunto representan el trazado de la ruta en la siguiente entrada de la base de datos.

```
/route_directions/{route_id}/[{route_coord_1}, ... , {route_coord_N}]

route_id: Identificador de la línea de taxi colectivos.
route_coord_X: Objeto con la latitud y longitud de
un punto geográfico de la ruta.
```

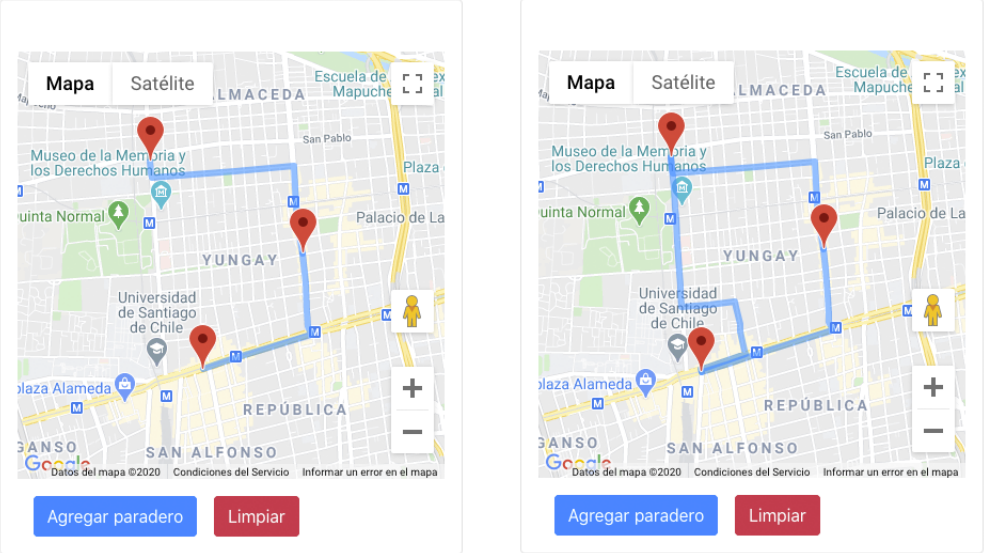

Figura 5.23: Ruta trazada terminada mediante la unión con el inicio de la ruta

5.3.3. Modificar línea existente

Para modificar una línea existente, esta se debe acceder desde la pantalla inicial de la aplicación web y presionar la línea que se quiere editar. El formulario es idéntico al de agregar una nueva línea, pero con la diferencia que los campos de las primeras dos secciones del formulario vienen con los valores existente en la base de datos. Lo más relevante de esta funcionalidad es la edición del trazado de la ruta debido a que si se quiere editar, se deberá partir desde cero. Esta decisión se tomo en base a que este trazado no debería ser cambiado en la existencia de la línea de taxi colectivos debido a que es para una línea pueda cambiar su trazado, deberá hacerlo mediante una petición extensa al ministerio de transporte. Por lo tanto, la única posibilidad de cambiar el trazado, bajo esta plataforma web, es por un error del usuario administrador que ingreso de manera incorrecta el trazado de la ruta.

Para guardar los cambios hechos y enviarlos a las entradas respectivas de la base de datos, se procede a presionar el botón de Guardar al final de este formulario. Sin embargo, si el usuario administrador hizo algún cambio en el trazado de la ruta, existe una alerta de confirmación para proceder con esta acción, debido a que es una acción que no se debería hacer de manera frecuente.

5.3.4. Eliminación de una línea

Para eliminar una línea existente, se debe proceder de la misma manera que para editar una línea existente. Al final del formulario existe un botón **Eliminar**. Al presionar este botón se muestra una advertencia para confirmar la acción, si el usuario administrador procede a confirmar la acción, se elimina la línea y la información guardada de esta en las entradas de la base de datos correspondientes.

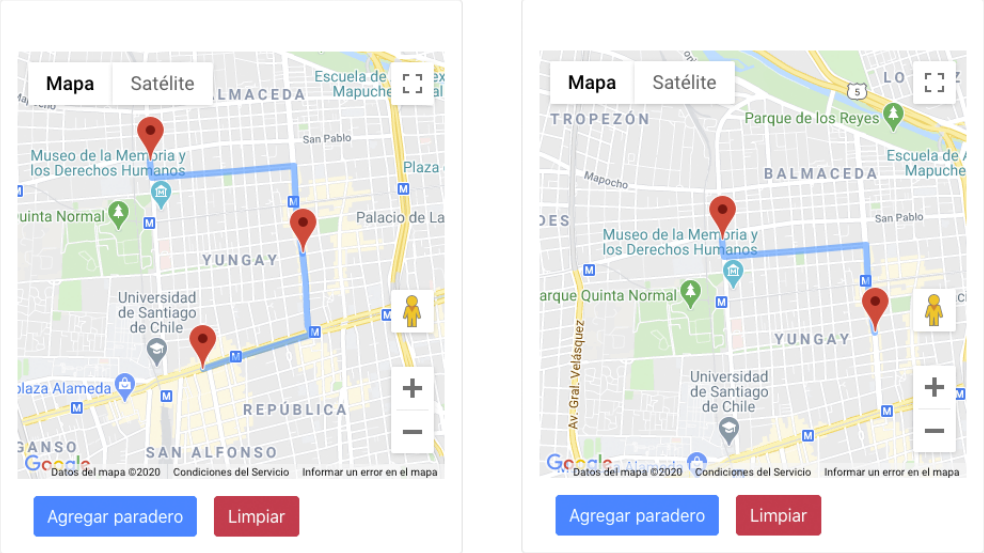

Figura 5.24: Trazado resultante al quitar punto relevante

5.4. Back end

En esta sección se mostrará la forma de abordar el manejo de datos y cómo este se transforma en el motor de la lógica detrás de las aplicaciones. Primero se describe la arquitectura e implementación, luego la estructura de la base de datos, y finalmente, la integridad de los datos para que el sistema funcione consistentemente.

5.4.1. Arquitectura

La arquitectura del back end utilizada en la solución es del tipo **serverless**, orientado a funciones que ejecutan código en la nube cumpliendo labores especificas. La implementación de este back end se puede dividir en dos, la primera está implementada en **Firebase Cloud Functions** y la segunda en cada aplicación gracias a la librería **RNFirebase**.

En **Firebase Cloud Functions** se implementó todo lo relacionado a consultas que necesitaban un procesamiento computacional mayor a la que puede ofrecer un teléfono celular en un periodo razonable, evitando así malas experiencias de usuario, como por ejemplo, que se demore mucho en encontrar las rutas cercanas en la aplicación del pasajero. A continuación se muestra un ejemplo de estas funciones que se ejecutan en la nube.

```
1
2 exports.passengerCanWait = functions.https.onRequest(async (req, res) => {
3     const passengerLocation = req.body.data.location;
4     const routeId = req.body.data.routeId
5     admin.database().ref(routeDirections)
6         .child(routeId)
7         .once('value')
8         .then((snapshot) => {
9             res.send({ data: {passengerCanWait: utils.checkIfRouteIsNear(
   snapshot.val().route, passengerLocation, 120)} });
```

```
            return null
        })
        .catch(error => {
            console.log(error);
            res.status(500).send(error);
        })
})
```

Esta función expone mediante HTTPS, recibe un ubicación del pasajero `passengerLocation` y un identificador de una línea de taxi colectivo `routeId` y retorna un objeto indicando si el pasajero puede esperar o no en la ruta bajo el criterio de que este cerca del recorrido, como lo expresa la función `checkIfRouteIsNear`.

Como la función anterior que se expone mediante HTTPS existen varias, como por ejemplo, para calcular las rutas cercanas a un pasajero o encontrar la línea de taxi colectivo a la cual pertenece un conductor.

Por otro lado, también se implementó funciones programadas. Estas funciones se ejecutan periódicamente con una labor especifica. A continuación se muestra un ejemplo de este tipo de funciones.

```

exports.clearInactiveDrivers = functions.pubsub.schedule('every 20 minutes
    ').onRun(async (context) => {
     let promises = []
     await admin.database()
     .ref("/driver_in_route_last_activity")
     .once('value')
     .then((snapshot) => {
         return snapshot.forEach(function(childSnapshot) {
             if(childSnapshot.val()) {
                 let minutesDifference = Math.floor((Date.now() -
    childSnapshot.val().timestamp)/1000/60)
                 if(minutesDifference > 13) {
                     promises.push(utils.clearDriverGeoFire(childSnapshot.
    key))
                     promises.push(utils.removeTimestampActiveDriver(
    childSnapshot.key))
                 }
             }
         })
     })
     return Promise.all(promises);
});
```

Esta función *programada* se ejecuta cada **20 minutos** para verificar si existe un conductor en la entrada `driver_in_route_last_activity/{driver_id}` que no haya actualizado su última actividad por más de **15 minutos** (considerándolo inactivo) y procede eliminar la

entrada `driver_in_route_last_activity/{driver_id}` y actualiza la latitud y longitud en 0 en la entrada `driver_location/{driver_id}` para marcarlo como inactivo.

Paralelamente, en las aplicaciones también existe una implementación del back end *serverless* gracias a la librería RNFirebase. Esta librería permite escribir código para comunicarse con la base de datos de Firebase de una manera sencilla. En consecuencia, todas las consultas a la base de datos que no necesiten un procesamiento especial pasan por una comunicación directa entre las aplicaciones y la base de datos.

En cada aplicación existe un modulo **back end** que permite estas interacciones básicas con la base de datos. Un ejemplo de una función básica implementada en este módulo se puede ver a continuación.

```
1
2 export const seatsTakenUpdate = async (driverId, seatsTaken) => {
3     return References.seatsOccupancy
4         .child(driverId)
5         .update({
6             seatsTaken: seatsTaken
7         })
8 }
```

Esta función actualiza la entrada `seats_occupancy/{driver_id}/{seats_taken}` con el número de asientos tomados por los pasajeros del conductor en el recorrido. Esta actualización se considera básica y por lo tanto se implementa directamente en la aplicación del conductor. Existen múltiples de funciones en las distintas aplicaciones que cumplen con labores con baja complejidad como la mostrada en el ejemplo anterior.

Finalmente, el back end *serverless* resultante tiene su implementación descentralizada en las aplicaciones, y además en **Firebase Cloud Functions** como se pudo apreciar en esta sección.

5.4.2. Base de datos

La base de datos que se utilizó es la **Firebase Realtime Database** como se mostró en el diseño de la solución. Esta es una base de datos NoSQL la cual es conformada por una estructura en formato **json** de llave y valor. Las llaves de más alto nivel se denominan entradas y en la siguiente sección veremos como las aplicaciones y el back end interactúa con estas entradas.

Entradas

En la Figura 5.25 existen cuatro principales componentes, la aplicación del conductor (**A**), la aplicación del pasajero (**B**), la aplicación web administrativa (**C**) y el back end implementado en Firebase Cloud Functions (**D**). A continuación veremos cómo estos componentes interactúan con las entradas.

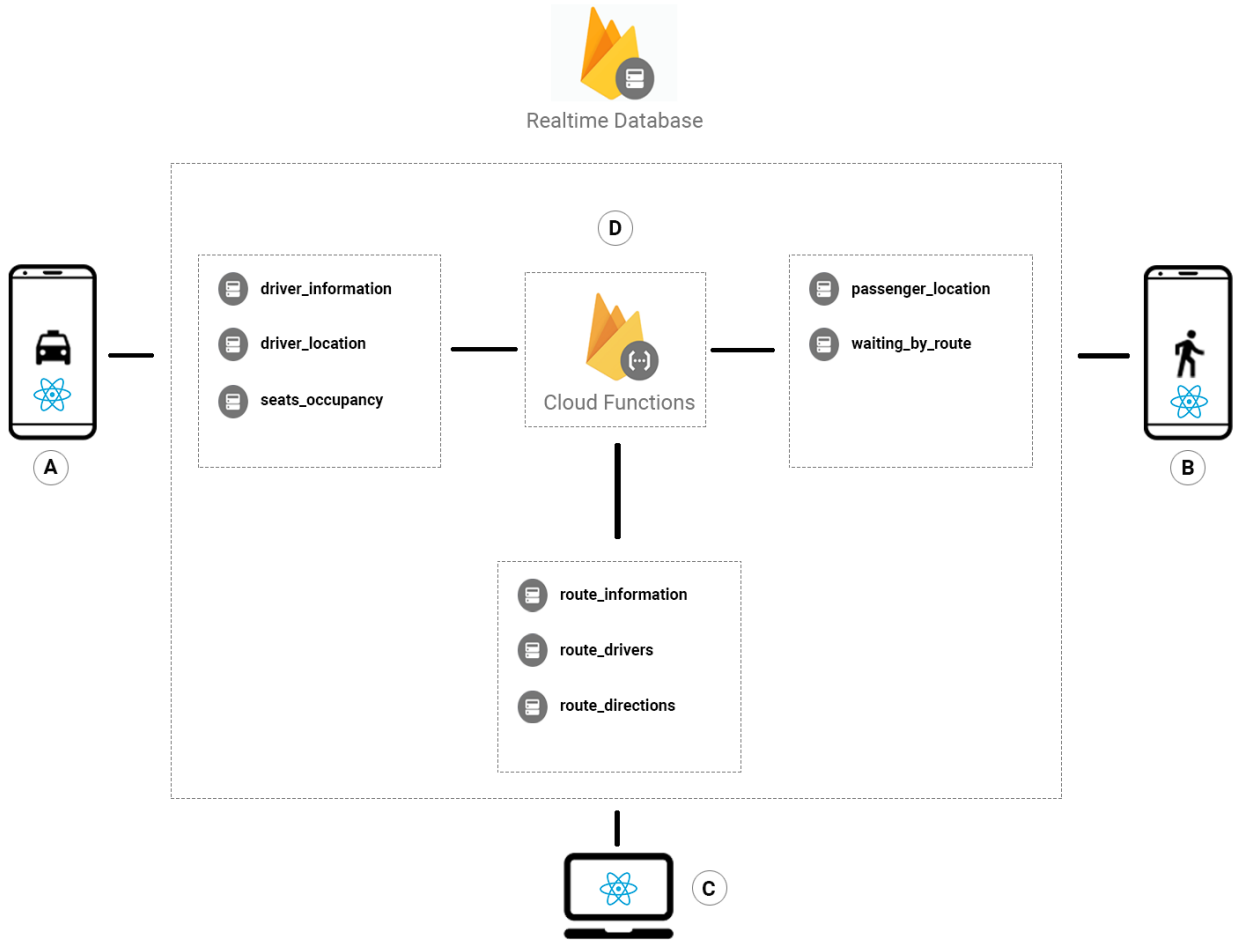

Figura 5.25: Interacción entre aplicaciones y entradas de la base de datos de Firebase

La aplicación de conductor alimenta directamente a las entradas relacionadas al conductor, estas entradas como muestra la Figura 5.25 son: `driver_information` que contiene la información del conductor (nombre, patente del vehículo, etc), `driver_location` que contiene la posición del conductor en la ruta y `seats_occupancy` que guarda la cantidad de asientos utilizados por pasajeros en el vehículo del conductor.

La aplicación del pasajero alimenta directamente a las entradas relacionadas al pasajero, estas entradas como muestra la Figura 5.25 son: `passenger_location` que contiene su posición en la ruta y `waiting_by_route` que contiene a los pasajeros esperando por cada ruta de línea de taxi colectivos.

La aplicación web administrativa alimenta directamente a las entradas relacionadas con las líneas de taxi colectivos, estas entradas como muestra la Figura 5.25 son: `route_information` que contiene la información general de la línea de taxi colectivos (nombre, horario de funcionamiento, tarifa, etc), `route_drivers` que contiene a los conductores asociados a las líneas de taxi colectivos y `route_directions` que contiene los trazados de las líneas de taxi colectivos en el mapa.

Por otro lado, también está el back end montado en Firebase Cloud Functions, donde

tiene permiso e interactúa con todas las entradas antes mencionadas. Esta interacción se da por dos razones como se hablo en la sección anterior, la primera es por la ejecución de tareas que requieren un procesamiento de más alta complejidad y las aplicaciones le derivan la tarea a este back end, y la segunda es por las tareas de consistencia de datos que se verá en la sección 5.4.3.

Cabe destacar que todos los componentes antes mencionados (A, B, C y D) pueden leer de la base de datos, como por ejemplo, en la aplicación del conductor, se lee la entrada `passenger_location` y `waiting_by_route` para exponer a los pasajeros que están esperando en la ruta de la línea de taxi colectivo respectiva. Todos estos detalles sobre permisos de lectura y escritura se verán en detalle en la siguiente sección.

Seguridad

La base de datos en tiempo real de Firebase cuenta con reglas para brindar seguridad a los datos almacenados [38]. Estas reglas se implementan con un objeto un **json** que definen quien puede leer o escribir en una cierta entrada en la base de datos. Las reglas definidas para la base de datos de este trabajo se pueden agrupar en dos. El primer grupo son las entradas de la base de datos que pueden ser **leídas** por cualquier usuario autenticado en el sistema y pueden ser **escritas** solamente si la entrada coincide con su `id`. Por ejemplo, la ruta `drivers_location/{driver_id}` solamente puede ser escrita si el usuario que hace la petición esta autenticado y además su `id` coincide con `{driver_id}`, asegurando que solamente el conductor pueda cambiar su ubicación. Las entradas de la base de datos en este grupo son las siguientes:

1. `drivers_location/{driver_id}`. Como ya se explicó en el ejemplo, solamente el conductor que tiene como identificador `driver_id` puede cambiar la ubicación.
2. `passengers_location/{passenger_id}`. Al igual que el conductor, solamente el pasajero que tiene como identificador `passenger_id` puede cambiar la ubicación.
3. `seats_occupancy/{driver_id}`. Solamente el conductor que tiene como identificador `driver_id` podrá cambiar la capacidad de asientos libres en el trayecto.
4. `driver_information/{driver_id}`. Solamente el conductor que tiene como identificador `driver_id` podrá cambiar su información.
5. `passengers_waiting/{route_id}/{passenger_id}`. Solamente el pasajero podrá agregar `passenger_id` a la lista de pasajeros en espera de una línea `route_id`. Si intenta agregar otro `id`, será rechazado por la base de datos por permiso denegado.

El segundo grupo son las entradas de la base de datos que pueden ser **leídas** por cualquier usuario autenticado en el sistema y pueden ser **escritas** solamente por un administrador desde la aplicación web administrativa. Las entradas de la base de datos en este grupo son las siguientes:

1. `route_information`
2. `route_directions`
3. `route_drivers`

Esto se resume en que cualquier información relacionada a las **líneas de taxi colectivos**

debe ser ingresada y modificada desde la aplicación web administrativa.

5.4.3. Consistencia de datos

Durante el desarrollo de las aplicaciones móviles se presentaron escenarios donde estas podían dejara la base de datos en un estado inconsistente generando una mala experiencia de usuario a los conductores y pasajeros que las utilizan.

El primer escenario es cuando un conductor que está activo en ruta cierra la aplicación de manera abrupta. Esto se puede traducir en que el conductor quedará en una posición en la ruta sin poder actualizar su ubicación ni tampoco sus asientos disponibles. Este escenario puede ocurrir porque el conductor se quedó sin batería en el celular, cerro la aplicación sin querer, etc. Para este caso se procedió a crear una función programada en **Firebase Cloud Functions** que se ejecutará cada **20 minutos** para verificar si los conductores **activos** han actualizado su ubicación en los últimos **15 minutos**, y en el caso de que no, se procede a dejarlo como inactivo (ubicación latitud y longitud en 0). Esta función permite limpiar a los conductores *fantasmas* y deja solamente a los usuarios que están participan en el sistema de manera correcta.

El segundo escenario es con respectos pasajeros en espera que cierran la aplicación de manera abrupta. Esto se traduce a que existirá un pasajero *fantasma* esperando en un recorrido de una línea de taxi colectivos. Este escenario puede ocurrir cuando un pasajero se aburre de esperar y cierra la aplicación sin indicar que ya no esperará más. Para este caso también se procedió a crear una función programada en **Firebase Cloud Functions** que se ejecuta cada **20 minutos** para verificar si el pasajero ha actualizado su ubicación en los últimos **15 minutos**, y en el caso de que no, se procede a sacarlo de su estado de espera en la entrada de la base de datos `passengers_waiting/{route_id}/{passenger_id}`.

En ambos escenarios es necesario saber cuando fue la última vez que se actualizó su ubicación en la base de datos. Para el primer escenario se procedió a crear una entrada en la base de datos que guardará el `timestamp` [42] de la última vez que el conductor actualizará la entrada de la base de datos `drivers_location/{driver_id}`. La entrada con el `timestamp` de la última actividad de conductor es `driver_in_route_last_activity/{route_id}/{timestamp}`. Por lo tanto, la función programada consulta esta entrada para verificar que no haya pasado más de **15 minutos** sin que el conductor no haya actualizado su posición.

En el segundo escenario, se procedió a guardar la última actividad del pasajero en espera bajo la misma entrada de la base de datos en donde este informa que esta esperando `passengers_waiting/{route_id}/{passenger_id}/{timestamp}` y la función programada consulta esta entrada para saber si el pasajero ha pasado más de **15 minutos** inactivo.

5.5. Integración continua

El proceso de llevar las aplicaciones a los usuarios finales se automatizó mediante la plataforma Bitrise como se vio en la sección 4.5.4. En esta sección veremos la implementación del flujo que sigue un cambio en el código hasta que llega a los usuarios finales.

En primer lugar se procedió a inicializar un proyecto en Bitrise con dos aplicaciones móviles, la del conductor y pasajero. Este proyecto da la posibilidad de crear **Workflows** asociados a estas aplicaciones. Estos Workflows son una serie de pasos que una aplicación requiera para un cierto objetivo. Para una aplicación móvil los pasos más importante vistos en la sección 2.1 son la **automatización de compilación, pruebas y llevada a producción**, y su objetivo es llevar una aplicación robusta a los usuarios finales.

Para los pasos relacionados a la compilación y pruebas se procedió a crear un Workflow llamado **Primary**. Este tiene como objetivo compilar la aplicación sin errores y además de verificar que las pruebas unitarias superen el **85 %**. Además, se le agregó la configuración de ejecutarse cuando haya un cambio en el repositorio de las aplicaciones móviles en Github. Para este Workflow se procedió a agregar los siguientes pasos.

1. **Obtener certificado.** Este paso se encarga de verificar que el Workflow tiene permiso para poder obtener el código desde el repositorio de Github.
2. **Obtener código fuente.** Teniendo la autorización para que Bitrise pueda obtener el código de Github, en este paso se obtiene dicho código fuente.
3. **Instalar.** Una vez obtenido el código fuente, se agregó un paso para ejecutar un comando de consola que instala todas las dependencias para que la aplicación se pueda compilar. Este paso genera un versión compilada con sus dependencias.
4. **Pruebas unitarias.** Con la compilación y sus dependencias obtenidas, se agregó un paso para ejecutar un comando de consulta que corre todas las pruebas unitarias asociadas al proyecto. Este paso genera un informe general de la pruebas unitarias gracias a la librería **jest** [29].
5. **Reporte.** Finalmente, se agregó un paso para subir el informe generado en el paso anterior **Codecov** [9]. Esta herramienta ayuda a los desarrolladores a tener una visualización de la calidad de pruebas unitarias en sus proyectos.

5.5.1. Despliegue en producción

Una vez creado el Workflow que se encarga de los objetivos de compilación y pruebas unitarias de las aplicaciones, solamente queda crear uno que se encargue de llevarlas a producción (usuarios finales). Se optó por restringir llevar solamente la aplicación en **Android** a producción debido a que la tarifa para llevarlo a las tiendas de iOS (App Store) es de **99 dolares al año** [10], en cambio la tienda de Android (Google Play Store) cobra solamente **24 dolares y una sola vez**. Teniendo en cuenta lo anterior, se procedió a armar un Workflow llamado **Android** que se encarga de llevar la aplicación a la Google Play Store. A continuación se mostrarán los pasos para cumplir con este objetivo.

1. **Versión de compilación.** El primer paso es la generación de la versión de compilación para la aplicación. Esta versión será utilizada en los pasos siguientes para identificar de manera única la versión de la aplicación que se esta llevando a producción.
2. **Variables de ambiente.** La aplicación necesita credenciales para poder utilizar Firebase. Este paso brinda una llave para que la aplicación tenga permisos de acceder a los servicios de Firebase al momento de compilarse.
3. **Descargar archivo Keystore** Para poder firmar la aplicación y llevarla a producción se necesita de un archivo con credenciales de usuario. Esta credencial se obtiene

mediante la guía que Google tiene en su página web [30]. En este paso se descarga un archivo alojado en Bitrise con extensión **.keystore** con las credenciales asociadas al usuario creado por el alumno de este trabajo de título.

4. **Descargar Google Services.** Durante la configuración de Firebase también se necesita especificar otras configuraciones, como por ejemplo, el nombre del proyecto en Firebase. Estas configuraciones vienen en un archivo llamado Google Services [27].
5. **Compilación Android.** Este paso se encarga en crear el archivo .apk [4] el cual encapsula la aplicación. Este archivo es el que finalmente se instala en el celular del usuario final.
6. **Firmar aplicación.** Una vez con el archivo .apk, este necesita ser creado por un desarrollador certificado en Google. Este proceso de certificación es llevado a cabo mediante la firma del archivo .apk. Durante este paso se utiliza el archivo descargado anteriormente con extensión .keystore y se procede a firmar el archivo .apk para poder ser subido a la Google Play Store.
7. **Despliegue en Google Play Store.** Para este paso se procedió a crear una cuenta en la consola de desarrollador de Google [26] y seguir los pasos indicados por Bitrise para poder hacer un despligue del .apk generado a la Google Play Store [11]. Al finalizar este paso, el archivo .apk firmado anteriormente queda disponible en la consola de desarrollador de Google para poder ser liberado a los usuarios finales.

Después de una ejecución del Workflow, existe una nueva versión de las aplicaciones en la consola de desarrollador de Google. Esta versión debe ser revisada por el proceso de calidad de Google, indicando si es apta o no para ir a la Google Play Store. Luego de este proceso, que demora aproximadamente 2 horas, se configuró para que la nueva versión de las aplicaciones queden en un grupo de prueba de usuarios internos. Este grupo de usuarios son 2 conductores y 6 pasajeros de taxi colectivos. Estos usuarios participaron en la validación de la solución que se presentará en la sección 6.

Capítulo 6

Evaluación Preliminar

Con las aplicaciones registradas en Google Play Store, se procedió a realizar una evaluación preliminar de la solución con un grupo de usuarios para ambas aplicaciones. El grupo de de usuarios estuvo compuesto por 6 conductores y 10 pasajeros de la línea 104. Al grupo de usuarios pasajeros se le envió una invitación para descargar la aplicación de Google Play Store, en cambio al grupo de usuarios de conductores se procedió a acompañarlos para utilizar la aplicación, debido a que son usuarios que necesitan un apoyo extra para empezar a utilizar la aplicación (usuarios con poca experiencia en aplicaciones móviles). Este proceso duró aproximadamente dos semanas y luego se procedió a aplicar una encuesta de satisfacción del cliente (CSAT) [40].

En los cuestionarios para los conductores y pasajeros se adjuntó una explicación sobre cómo proceder con las respuestas, las que consistían en un número del **1** al **5**, en donde el **1** significa que la funcionalidad no soluciona en nada la problemática planteada en la pregunta y **5** significa que la funcionalidad soluciona totalmente la problemática planteada en la pregunta. En la Tabla 6.1 se adjunta el detalle de cada respuesta y su valor de Puntaje de satisfacción asociado.

Respuesta	Puntaje de satisfacción
No soluciona nada del problema	1
Soluciona mínimamente el problema	2
Soluciona parcialmente el problema	3
Soluciona la mayoría del problema	4
Soluciona el problema completamente	5

Tabla 6.1: Tabla explicativa de los puntajes de satisfacción relacionados a las respuestas de los usuarios

6.0.1. Conductores

Las preguntas a los conductores tenían como objetivo medir cómo ellos perciben que las funcionalidades solucionan sus problemas, en especial, los de información planteados en la sección 3. Las preguntas hechas y sus resultados se muestran a continuación, donde **PSP** significa **Puntaje de satisfacción promedio**.

Pregunta		PSP
1	La aplicación me muestra información suficiente de mi colega para poder tomar decisiones, como por ejemplo, empezar una ruta en un determinado tiempo.	**4.8**
2	La aplicación me muestra información suficiente de los pasajeros en espera en el recorrido para poder tomar decisiones, como por ejemplo, empezar una ruta en un determinado tiempo.	**4.6**
3	La aplicación me permite poder informarle a los pasajeros en espera en el recorrido mi estado en la ruta.	**4.8**
4	En general, la aplicación soluciona los problemas de falta de información y comunicación por parte del conductor en el mundo de los taxi colectivos.	**4.6**

Tabla 6.2: Puntajes de satisfacción relacionados a las respuestas de la aplicación del conductor

6.0.2. Pasajeros

Al igual que para los conductores, las preguntas a los pasajeros apuntan a medirla satisfacción de cómo perciben que las funcionalidades solucionan sus problemas de comunicación en el mundo de los taxis colectivos. Las preguntas hechas y sus resultados se muestran a continuación.

Pregunta		PSP
1	La aplicación me permite encontrar nuevos recorridos de taxi colectivos.	**4.9**
2	La aplicación me permite saber exactamente el recorrido de los taxi colectivos.	**5.0**
3	La aplicación me permite saber un aproximado de cuanto tendré que esperar para poder ser transportado por un conductor de la línea de taxi colectivo.	**4.6**
4	La aplicación me da la información necesaria para saber si optar por el taxi colectivo o por otro medio de transporte.	**4.6**
5	La aplicación me da las herramientas necesarias para informar a una línea de taxi colectivos que estoy esperando para ser transportado.	**4.8**
6	En general, la aplicación soluciona los problemas de falta de información y comunicación por parte del pasajero en el mundo de los taxi colectivos.	**4.8**

Tabla 6.3: Puntajes de satisfacción relacionados a las respuestas de la aplicación del pasajero

6.0.3. Análisis

Durante el proceso de la evaluación preliminar se obtuvieron resultados positivos con respecto a cómo las aplicaciones solucionan los problemas de información y comunicación

en el ámbito de los taxi colectivos. Esto se puede atribuir a que la usabilidad y utilidad de las aplicaciones fueron puestas a prueba constantemente en la validación del diseño del ciclo de vida de las funcionalidades. Además, los requerimientos para esta aplicación fueron obtenidos directamente de los usuarios mediante el levantamiento de ellos, por lo tanto, todo lo desarrollado tenía una razón basada en la percepción de los usuarios sobre los problemas reales que los pasajeros y conductores de taxi colectivos perciben.

Finalmente, esta validación permite tener una base para futuras iteraciones sobre la percepción de los usuarios acerca de las aplicaciones y sus funcionalidades; si bien se obtuvieron resultados bastantes buenos, ellos expresaron de manera informal ciertos detalles y nuevas funcionalidades que a los usuarios les gustaría agregar a las aplicaciones. Por lo tanto, esta validación permite tener una referencia de cómo irá evolucionando las aplicaciones durante el tiempo.

Conclusión

Durante el presente trabajo de título se implementó una solución a los problemas que pasajeros y conductores de taxis colectivos perciben, las que limitan la disposición de las personas a usar este servicio. la idea principal es que tanto conductores como pasajeros Esta solución está basada en un sistema de tres aplicaciones, una para el conductor de taxi colectivos, otra para sus pasajeros y una tercera para administrar el sistema. Las funcionalidades de las aplicaciones para los conductores y pasajeros fueron obtenidas mediante un levantamiento de requerimientos orientado a registrar la percepción de las problemáticas actuales que ellos perciben al decidir si usar o no el servicio de taxis colectivos, las cuales están asociadas a falta de información. Esto es precisamente lo que este trabajo pretende resolver.

En la aplicación del conductor se implementó un mapa que muestra a sus colegas activos y pasajeros en espera en el recorrido, con intención de brindarle información sobre su recorrido y así poder una decisión de cuando salir al recorrido en el momento más conveniente para captar pasajeros. Para los pasajeros se implementó una aplicación que los informara de la localización de los taxis y cuantos asientos disponibles tienen, así los pasajeros pueden decidir si esperarlo u optar por otro medio de transporte. Finalmente, se implemento un perfil del conductor para darle la oportunidad de compartir a futuro ciertos datos para fidelizar a los clientes, como foto de perfil y teléfono celular.

En la aplicación del pasajero, se implementó un buscador de recorridos mediante lugares, así el usuario puede conocerlos y verificar si existe algún recorrido que le sirve para su trayecto. En complemento, se creó un perfil del recorrido para que el pasajero pueda informarse de mejor manera sobre el recorrido y sus conductores activos, y a la vez poder informar a los conductores que está esperando en el recorrido. Por último, se implemento una lista de recorridos favoritos del pasajero, en donde este podrá tener un rápido acceso a los recorridos que utiliza frecuentemente.

Para poder introducir nuevas líneas al sistema, se creó una aplicación administrativa en donde el usuario administrador tiene la capacidad de agregar una nueva línea de taxi colectivo, modificar y eliminar una línea existente. Las líneas de taxi colectivos pertenecientes al sistema cuentan con información de sus conductores asociados, trazado del recorrido e información general, como por ejemplo, nombre de la línea, horario de funcionamiento y tarifa.

Todas las aplicaciones se llevaron a producción y se procedió a hacer una evaluación preliminar de la solución con una encuesta de satisfacción con un grupo de usuarios cerrado de conductores y pasajeros. El resultado final de esta encuesta muestra resultados satisfactorios de cómo las aplicaciones solucionan los problemas levantados por estos mismos, desde la

perspectiva de los pasajeros y conductores. Este proceso fue un poco obstaculizado por la contingencia actual del COVID-19, debido a que se pensaba hacer una prueba más masiva de la solución, pero las medidas sanitarias restringieron la validación a un grupo reducido, pero aun así, esta fue importante para dar una primera evaluación de la solución.

Se espera que estos primeros pasos de una solución tecnológica basada en el desarrollo de software para el mundo de taxi colectivos tengan impacto en un medio de transporte importante como los taxi colectivos. La posibilidad de tener la información disponible en los teléfonos móviles de usuarios, y posibles usuarios, hace que este medio de transporte pueda entrar a la competencia de los otros medios de transporte que si cuentan con múltiples soluciones tecnológicas, y así puedan escalar en el tiempo.

Los próximos pasos a seguir son llevar esta solución a toda la población que utiliza taxi colectivos, para potencialmente, atraer nuevos usuarios.

Además, se espera en el futuro agregar funcionalidades con una complejidad mayor, como por ejemplo, en la aplicación del pasajero, un recomendador de ruta dado un punto de origen y un punto de destino, o que puedan pagar con otro medio de pago que no sea efectivo. Este tipo de funcionalidades ayudarían a enriquecer la solución actual y daría el primer paso a una nueva forma de utilizar los taxi colectivos.

Bibliografía

[1] Amazon web services. https://aws.amazon.com/es/.

[2] Api directions https://developers.google.com/maps/documentation/javascript/directions.

[3] Api places search https://developers.google.com/places/web-service/search.

[4] Archivo apk. https://es.wikipedia.org/wiki/APK$_{(}formato$).

[5] Azure. https://azure.microsoft.com/es-es/.

[6] Backend serverless. https://www.genbeta.com/desarrollo/que-serverless-que-adoptarlo-desarrollo-tu-proxima-aplicacion.

[7] Bases de datos de parques vehiculares, septiembre 2020, chile. https://usuarios.subtrans.gob.cl/estadisticas/parques-vehiculares.html.

[8] Bitrise. https://www.bitrise.io/.

[9] Codecov. https://codecov.io/.

[10] Costo para publicar en app store. https://developer.apple.com/support/compare-memberships/.

[11] Deploy google play store on bitrise. https://blog.bitrise.io/how-to-deploy-android-app-to-google-play-store.

[12] Desarrollo agil de software. https://es.wikipedia.org/wiki/Desarrollo_Ágil_de_software.

[13] Desarrollo en cascada https://es.wikipedia.org/wiki/Desarrollo$_{e}n_{c}asc$ada.

[14] Desarrollo iterativo y creciente. https://es.wikipedia.org/wiki/Desarrollo_iterativo_y_creciente.

[15] Ejemplo de geoquey en geofire https://github.com/firebase/geofire-js/blob/master/examples/fish2/js/fish2.js.

[16] Entrega continua. https://es.wikipedia.org/wiki/Entrega_continua.

[17] Figma. https://www.figma.com.

[18] Firebase anonymous authentication https://firebase.google.com/docs/auth/web/anonymous-auth?hl=es.

[19] Firebase authentication. https://firebase.google.com/docs/auth?hl=es-419.

[20] Firebase cloud storage.

[21] Firebase https://firebase.google.com/?hl=es.

[22] Flutter. https://flutter.dev/.

[23] Front end y back end. https://es.wikipedia.org/wiki/Front_end_y_back_end.

[24] Geofire. https://github.com/firebase/geofire-js.

[25] Github. https://github.com/.

[26] Google developer console. https://console.developers.google.com/?hl=ESpli=1.

[27] Google play services. https://developers.google.com/android/guides/overview.

[28] Historia de usuario https://es.wikipedia.org/wiki/Historias$_{\mathrm{d}}$e$_{u}suario$.

[29] Jest. https://jestjs.io/.

[30] Keystore. https://developer.android.com/training/articles/keystore?hl=es-419.

[31] Quien esta usando flutter. https://flutter.dev/showcase.

[32] Quien esta usando react native. https://reactnative.dev/showcase.

[33] React js https://es.reactjs.org/.

[34] React nativa maps https://github.com/react-native-maps/react-native-maps.

[35] React native geolocation service for ios and android.. https://www.npmjs.com/package/react-native-geolocation-service.

[36] React native. https://reactnative.dev/.

[37] Redux https://es.redux.js.org/.

[38] Reglas base de datos firebase https://firebase.google.com/docs/database/security?hl=es.

[39] Rnfirebase. https://rnfirebase.io/.

[40] Satisfacción del cliente https://es.wikipedia.org/wiki/Satisfacción$_{\mathrm{d}}$e$l_{c}liente$.

[41] Scrum https://es.wikipedia.org/wiki/Scrum$_{(}$de$sarrollo_{\mathrm{d}}$e$_{s}oftware$).

[42] Timestamp https://es.wikipedia.org/wiki/Marca$_{t}emporal$.

[43] Trello. https://trello.com/es.

[44] Why agile development needs continuous delivery. https://www.atlassian.com/continuous-delivery/principles/why-agile-development-needs-continuous-delivery.