

Desarrollo de una Aplicación Móvil para un sistema de gestión de transportes para comedores solidarios

Ariel Ossandón Tudesca

arielossandon@usm.cl

Profesor Guía: Catherine Gomez

Profesor Correferente: Ricardo Cahe

Resumen: En el presente informe, se expondrá el desarrollo de una aplicación móvil enfocada en un sistema de gestión de transportes, que busca solucionar la problemática de la atención de comedores solidarios en la ciudad de Viña del Mar. El proyecto se llama NeoRoute, se compone de dos entornos diferentes, una aplicación web destinada al administrador de recursos, quien los asigna a los diversos comedores, y una aplicación móvil para los conductores, encargados de realizar las entregas a cada comedor. El entorno de la aplicación móvil está desarrollada con tres objetivos principales: la utilización de servicios de terceros de mapas, la integración de la API de obtención de rutas (Google Maps) y la incorporación del backend. La aplicación móvil para Android, NeoRouteApp, está construida bajo la arquitectura de desarrollo MVC (Modelo-Vista-Controlador), utilizando el framework Flutter con el lenguaje de programación Dart.

Palabras clave: Android, Flutter, Google Maps, MVC, Rutas

1. Introducción

Las catástrofes que azotan a la comunidad son en su mayoría inesperadas, y mucha gente en cosa de segundos pueden perder posesiones que han tomado mucho esfuerzo y sacrificio obtener. Y es el deber de las autoridades socorrer a quienes lo necesitan, de una forma eficiente y en el menor tiempo posible.

A mediados de enero del año 2024, la región de Valparaíso sufrió un mega-incendio que afectó las ciudades urbanas de Viña del Mar, Quilpué, Villa Alemana y Limache. Afectando empresas, industrias y consumiendo alrededor de 6.000 viviendas. [1]

La Municipalidad de Viña del Mar posee la ardua tarea de abastecer a las familias afectadas, mediante la entrega de recursos alimentarios a los diversos comedores solidarios que están presentes en las zonas afectadas.

Hoy en día, esta institución trabaja estancada en métodos imprácticos para organizar la entrega de los diversos alimentos a los comedores solidarios, lo que provoca que la entrega de los recursos sea lenta y desorganizada.

En vista de lo anterior, este informe abordará el desarrollo de una aplicación móvil, En un estado de producto mínimo viable, que cuente con las funcionalidades necesarias para demostrar que es de ayuda a esta institución a cumplir con su labor. Esta aplicación móvil será parte de un sistema de gestión de transportes, la aplicación estará orientada a un usuario: El Conductor, que es el encargado de realizar la entrega de los alimentos a los comedores solidarios. Este Usuario recibirá una planificación que contiene los diversos comedores a los que debe entregar los productos, cada comedor posee Encargados y estos son quienes reciben los productos. También existe un Administrador quien es la persona que debe decidir que comedor abastecer. El Administrador agrupa los comedores en sectores y genera una planificación a cada uno de estos sectores y designa las planificaciones por día los conductores. Al recibir una serie de planificaciones con entregas a realizar, el conductor, dentro de la



aplicación podrá conocer el detalle de una planificación, visualizando un mapa el cual le mostrará la ruta óptima para llegar a su destino y los comedores a entregar. La aplicación será desarrollada usando el framework Flutter, enfocado en aplicaciones multiplataforma, con el lenguaje de programación Dart, con el que se pueden desarrollar aplicaciones a través de Flutter. La aplicación móvil se comunica con un backend construido con Django. La API REST, desarrollada utilizando Django REST framework, expone los datos de los usuarios, comedores y otros datos del sistema.

1.1 Contexto

La Municipalidad de Viña de Mar es la institución encargada de gestionar los asuntos públicos de la comuna, siendo el abastecimiento de los comedores solidarios una gran tarea que cumplir, siendo la cantidad de comedores en momentos de emergencia más de 170. Debido a esto, la municipalidad se dispuso con personal encargado de gestionar los recursos a repartir entre los comedores solidarios. La municipalidad no posee un sistema de información que le permita agilizar este proceso, siendo la principal vía de comunicación las llamadas telefónicas entre los encargados de un comedor, el personal encargado de repartir los alimentos y el conductor. Este tipo de comunicación provocó algunos problemas que afectaron el proceso, por ejemplo, los siguientes:

- Debido a la forma de comunicación que poseen las partes, tiene como consecuencia tiempos de respuesta prolongados, lo que lleva a una pérdida de recursos o mermas de los alimentos a entregar ya que por la demora no alcanzan a ser consumidos o al llegar se encuentran en mal estado.
- La municipalidad y el equipo encargado del abastecimiento de los comedores carecen de un sistema de apoyo y gestión para los comedores solidarios. Esta situación genera tiempos de respuesta prolongados ante las solicitudes de los comedores, debido principalmente a la falta de organización.

1.2 Propuesta de solución

Las consecuencias y las incalculables pérdidas generadas por la catástrofe afectaron en gran medida el desarrollo de la región y de la zona, al estar conscientes de cómo opera la municipalidad para solventar esta necesidad, se debe considerar el desarrollo de un sistema de gestión de transportes para que sea utilizado por la Municipalidad. Este sistema se compone de dos entornos: una plataforma web que será usada por el Administrador, quien es el responsable de administrar los alimentos y la creación de planificaciones para la entrega en los comedores solidarios. Por otra parte, cuenta con una aplicación móvil que será usada para el Conductor. La aplicación móvil mostrará en un mapa interactivo la ubicación de los comedores que deben ser abastecidos, junto con una vista detallada de cada comedor por abastecer y herramientas de navegación, que guiarán al Conductor hasta llegar a su destino. El objetivo es desarrollar una aplicación móvil que facilite al Conductor saber a quién debe realizar la entrega y cómo llegar al lugar. Notificando al Administrador cuando finalice sus entregas.

Con el desarrollo de esta aplicación en un estado de producto mínimo viable se mostrará como una alternativa eficiente, reduciendo el tiempo de los diversos pasos y el ahorro de recursos por parte de la Municipalidad.

1.3 Objetivos

Objetivo general: Desarrollo de una aplicación móvil que permita encontrar la ruta óptima para la entrega de alimentos en diversos comedores solidarios de la ciudad de Viña del Mar.

Objetivos específicos:

- 1- Integrar un mapa de terceros que permita visualizar marcadores que representan los comedores solidarios a abastecer.
- 2- Incorporar el backend del sistema, que contiene la lógica de negocio.
- 3- Integrar el API de terceros que permite obtener la ruta óptima y las instrucciones de direccionamiento para la navegación.

1.4 Diagrama de Flujo

Este es el diagrama de flujo de la Aplicación NeoRouteApp, para mayor comprensión de este proyecto:

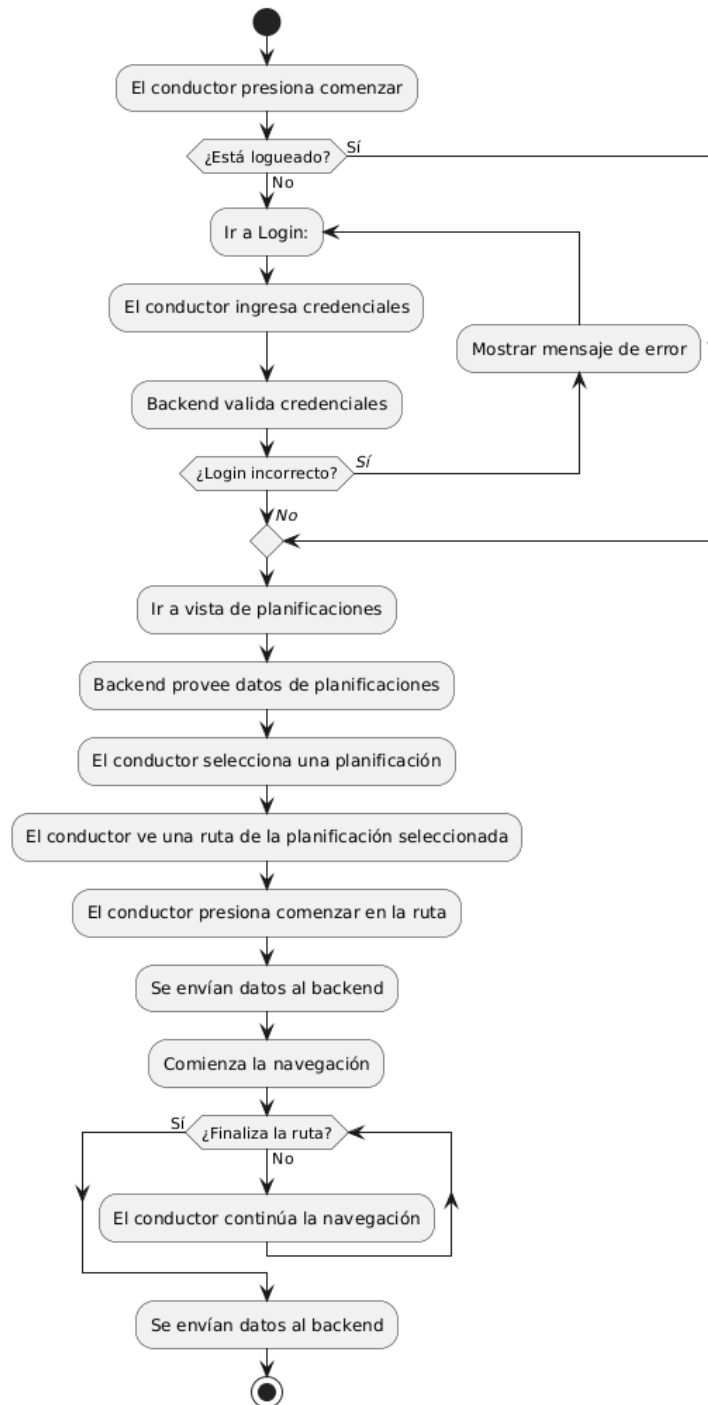


Figura 1. Diagrama de Flujo NeoRouteApp

Fuente: Elaboración propia

1.5 Tabla de requerimientos funcionales

Tabla 1. Tabla de requerimientos funcionales de NeoRouteApp

Fuente: Elaboración propia

Código	Nombre	Descripción
--------	--------	-------------



RF-01	Comenzar	La aplicación debe proporcionar una pantalla de inicio que verifique el estado de autenticación del usuario y realice la inicialización de componentes necesarios.
RF-02	Login	Los Conductores debe poder iniciar sesión, con nombre de usuario y contraseña
RF-03	Ver Planificaciones	La Aplicación permitirá ver las planificaciones asignadas al Conductor que se ha autenticado
RF-04	Seleccionar una planificación	Los Conductores deben ser capaces de seleccionar una planificación a partir de las planificaciones asignadas
RF-05	Ver Ruta	La Aplicación debe permitir la visualización de la ruta de los comedores pertenecientes a ella, mostrando a cada comedor con un marcador dentro del mapa
RF-07	Finalizar Ruta	La Aplicación debe permitir al conductor finalizar su ruta de entrega, con un proceso de actualización del estado de la planificación.

1.6 Diagrama Casos de Uso

Los diagramas de casos de uso son sumamente relevantes para un proyecto de desarrollo de Software ya que entregan una representación gráfica de los escenarios de uso para los que un sistema es desarrollado. En este diagrama se muestran las funcionalidades importantes que la aplicación debe manejar mediante casos de uso:

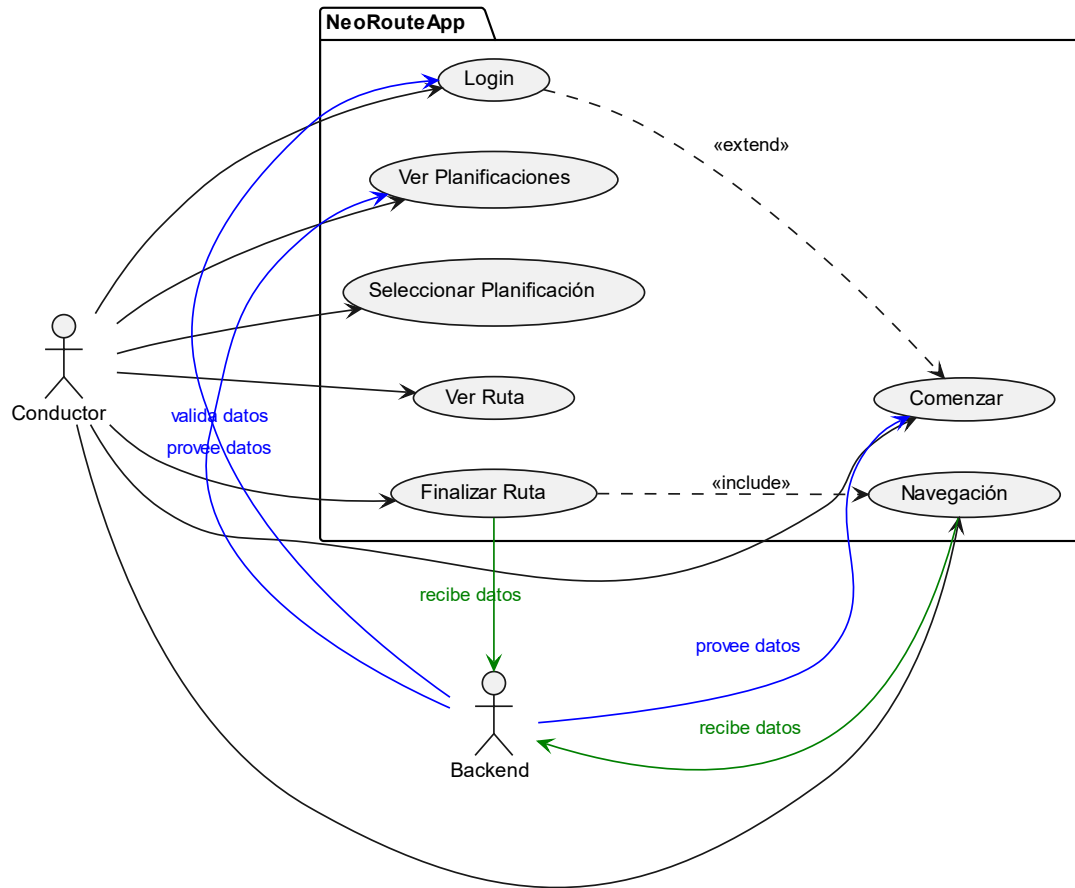


Figura 2. Diagrama de casos de uso
Fuente: Elaboración propia

1.7 Descripciones de casos de uso

Estas son las descripciones de cada caso de uso presente en el diagrama:

1.6.1 Caso de Uso: Comenzar

Tabla 2. Descripción del caso de uso: Comenzar
Fuente: Elaboración propia

Comenzar	
Actor Principal	Conductor
Descripción	El Conductor desea comenzar a utilizar la aplicación NeoRoute
Objetivo	Que el Conductor comience a utilizar la aplicación NeoRoute
Requerimiento funcional asociado	RF-01
Precondiciones	- El Dispositivo debe tener conexión a internet



Postcondiciones	- Se debe navegar a la vista siguiente
Flujo Básico	1. La Aplicación inicia los componentes y controladores necesarios 2. El Usuario presiona el botón de “Comenzar” 3. La Aplicación revisa si posee credenciales dentro de su memoria 4. La Aplicación redirige a la vista de Planificaciones
Flujo Alternativo	4. La Aplicación no registra credenciales de ingreso, redirige al Usuario a la vista de Login

1.6.2 Caso de uso: Login

Tabla 3. Descripción del caso de uso: Login

Fuente: Elaboración propia

Login	
Actor Principal	Conductor
Descripción	El Conductor no está autenticado en la Aplicación NeoRoute, por lo tanto, es necesario realizar el proceso de Login
Objetivo	El Usuario desea iniciar sesión en la aplicación NeoRoute
Requerimiento funcional asociado	RF-02
Precondiciones	- El Conductor no debe estar autenticado
Postcondiciones	- Se debe navegar a la pantalla de Ver Planificaciones si las credenciales son correctas
Flujo Básico	1. El usuario ingresa sus credenciales (correo y contraseña). 2. La aplicación valida las credenciales con el backend. 3. Si son válidas, el usuario es redirigido a la pantalla que muestra las planificaciones diarias asignadas a él.
Flujo Alternativo	1. Si las credenciales no son válidas, se muestra un mensaje de error.

1.6.3 Caso de Uso: Ver Planificaciones

Tabla 4. Descripción del caso de uso: Ver Planificaciones

Fuente: Elaboración propia

Ver Planificaciones	
Actor Principal	Conductor

Descripción	En este caso de uso se muestran las planificaciones asignadas al Conductor ya autenticado
Objetivo	El Conductor tras autenticarse desea ver las planificaciones asignadas a él
Requerimiento funcional asociado	RF-03
Precondiciones	- El Conductor debe estar autenticado dentro de la Aplicación
Postcondiciones	- Se deben mostrar planificaciones asignadas al Conductor, si las hay.
Flujo Básico	1. La Aplicación utilizando las credenciales del Conductor autenticado, realiza las consultas necesarias en el Backend para obtener las planificaciones asignadas a este Conductor 2. El Conductor visualiza las planificaciones asignadas a él 3. El Conductor selecciona una planificación.
Flujo Alternativo	1. Si el Conductor no posee planificaciones asignadas, la aplicación muestra un mensaje de aviso

1.6.4 Caso de Uso: Seleccionar Planificación

Tabla 5. Descripción del caso de uso: Seleccionar planificación

Fuente: Elaboración propia

Seleccionar Planificación	
Actor Principal	Conductor
Descripción	El conductor selecciona una de las planificaciones que le fueron asignadas.
Objetivo	El Conductor seleccione la planificación que le interesa
Requerimiento funcional asociado	RF-04
Precondiciones	- El Conductor debe tener planificaciones asignadas
Postcondiciones	- La Aplicación continúa con la navegación a la siguiente pantalla.
Precondición	El conductor debe tener planificaciones asignadas y estar en la vista de Planificaciones Diarias
Flujo Básico	1. El conductor selecciona una planificación 2. La Aplicación lo dirige a la vista de Ruta (Ver: Caso de uso: Ver Ruta)



Flujo Alternativo	1. De suceder un inconveniente, la aplicación mostrará un mensaje de error
--------------------------	---

1.6.5 Caso de Uso: Ver Ruta

Tabla 6. Descripción del caso de uso: Ver Ruta

Fuente: Elaboración propia

Ver Ruta	
Actor Principal	Conductor
Descripción	El Conductor tras seleccionar una planificación diaria la Aplicación muestra una ruta sugerida, la cual pasa por los comedores presentes en la planificación
Objetivo	El Conductor desea ver la ruta para llegar a los comedores
Requerimiento funcional asociado	RF-05
Precondiciones	- El Dispositivo debe tener el GPS encendido - Se debe haber seleccionado una planificación
Postcondiciones	- La Aplicación debe mostrar la ruta en la pantalla
Flujo Básico	1. La Aplicación inicializa los controladores y consultas a las APIs necesarias para obtener y mostrar la ruta 2. El Conductor visualiza la ruta
Flujo Alternativo	1. De suceder un inconveniente, la aplicación mostrará un mensaje de error

1.6.6 Caso de Uso: Navegación

Tabla 7. Descripción del caso de uso: Navegación

Fuente: Elaboración propia

Navegación	
Actor Principal	Conductor
Descripción	El Conductor tras observar la ruta, presiona el botón de “Comenzar” el cual inicializa la navegación asistida, proveyendo de las instrucciones que le permitirán llegar a los comedores
Objetivo	El conductor desea comenzar con la navegación asistida



Requerimiento funcional asociado	RF-06
Precondiciones	- Se debe estar dentro de la vista “Ver Ruta”
Postcondiciones	- Cuando se finalice el recorrido, el Conductor deberá finalizar la ruta (Ver Caso de uso: Finalizar Ruta)
Flujo Básico	1. La Aplicación realiza las consultas necesarias e inicializa los controladores que permiten obtener las instrucciones de navegación para los comedores presentes en la ruta 2. El conductor visualiza la ruta y comienza a conducir 3. Conforme se avance en la ruta, la Aplicación mostrará las instrucciones adaptadas al progreso realizado
Flujo Alternativo	1. De suceder un inconveniente, la aplicación mostrará un mensaje de error y mostrará la ubicación del Conductor y los puntos que representan a cada comedor

1.6.7 Caso de Uso: Finalizar Ruta

Tabla 8. Descripción del caso de uso: Finalizar Ruta

Fuente: Elaboración propia

Finalizar Ruta	
Actor Principal	Conductor
Descripción	El Conductor ha completado las entregas para cada comedor y desea marcar la ruta de entrega como finalizada
Objetivo	El conductor desea finalizar la ruta asignada
Requerimiento funcional asociado	RF-07
Precondiciones	- Se debe estar en la navegación
Postcondiciones	
Precondición	Se debe estar dentro de la vista “Ver Ruta”
Flujo Básico	1. El Conductor finalizó la ruta de entrega 2. La Aplicación muestra un mensaje de aviso 3. El Conductor presiona el botón de la esquina superior derecha

	4. La Aplicación muestra los comedores pertenecientes a la planificación, junto con un botón de finalizar ruta 5. El Conductor presiona el botón de finalizar ruta 6. La Aplicación le entrega un cuadro de confirmación 7. El Conductor presiona confirmar 8. La Aplicación envía la actualización para el estado de esa planificación, cambiándola a realizada.
Flujo Alternativo	1. El Conductor finalizó la ruta y presiona el botón de finalizar ruta 2. La Aplicación le entrega un cuadro de confirmación 3. El Conductor presiona que no confirma 4. La Aplicación seguirá mostrando indicaciones, de haber llegado a su destino, le mostrará un mensaje de aviso

1.8 Alcance y Limitaciones

El presente trabajo abordará el desarrollo de una aplicación móvil para un sistema de gestión de transporte para comedores solidarios en un estado de producto mínimo viable orientado a dispositivos con sistema operativo Android, esta aplicación está orientada a ser utilizada por conductores.

Este desarrollo contempla que el conductor pueda visualizar un mapa de la ciudad con marcas que indican la ubicación de los comedores solidarios a abastecer. El conductor puede recibir planificaciones de entrega asignadas a él por el Administrador, encargado de gestionar el abastecimiento de los comedores. El Conductor al recibir una planificación de entrega, se mostrará un mapa, con la ubicación actual del conductor además de una ruta que le asistirá a cumplir con las entregas.

La ruta que se define como óptima será obtenida desde los servicios de una API de terceros, así también el mapa a desplegar en la aplicación.

El SDK que permite construir el mapa y El API de Direccionamiento se obtendrán desde Google Maps Services, mediante la plataforma de Google Cloud Console.

Google Maps Services es el conjunto de herramientas que proporciona mapeo y localización de Google, además proporciona variedad de utilidades que se pueden activar mediante la Google Cloud Console, entre ellos se destacan:

- Mapas Personalizables: Es posible diseñar mapas, cambiando colores, estilos y marcadores
- Servicios de Geocodificación: Permite convertir direcciones en coordenadas geográficas y viceversa
- Rutas y Direcciones: Ofrece servicios para calcular rutas y proporcionar direcciones entre coordenadas. Esta función permite obtener direcciones entre diversos tipos de transporte (a pie, vehículo y bicicleta). Pero en este desarrollo sólo se utilizará el direccionamiento para un vehículo.
- APIs y SDKs: Ofrece interfaces de programación y kits de desarrollo de software para integrar mapas y funcionalidades de localización en aplicaciones web y móviles.

2. Marco teórico

2.1 Fundamentos del Desarrollo Móvil

El desarrollo de aplicaciones móviles (a diferencia de PC) posee un enfoque diferente, debido al tipo de dispositivo que ejecutará dicha aplicación. Ya que el formato de visualización como la interacción que ofrece es diferente, el tamaño de pantalla es mucho menor al de un monitor, por lo tanto, las aplicaciones móviles deben ofrecer un diseño minimalista que permita visualizar los datos más relevantes para el usuario. Ejecutar las tareas de una manera cómoda e intuitiva, es sumamente relevante ya que, al ser un dispositivo portable, se suele utilizar mientras se realiza otra tarea en el entorno físico, por lo tanto, el diseño de la aplicación busca evitar el uso del teclado como método de entrada, ya que resulta incómodo en muchas situaciones y se priorizan los gestos en la pantalla. Además, la atención que se presta al dispositivo móvil en este contexto es menor. Debido a esto, el diseño y definir qué tipo de aplicación móvil se va a desarrollar es importante. [2]

2.1.1 Tipos de Aplicaciones Móviles:

Aplicaciones Nativas: Son un tipo de aplicaciones desarrolladas para una plataforma en específico (Android o iOS), utilizando un lenguaje de programación que permita el desarrollo nativo, como por ejemplo Kotlin y Java para Android y Swift para iOS. Estas aplicaciones permiten utilizar de una manera amplia el hardware del dispositivo en donde se ejecutan.

Aplicaciones Multiplataforma: Este desarrollo es posible gracias a diversos Frameworks diseñados para permitir un desarrollo multiplataforma. En un desarrollo de este tipo se reutiliza gran parte del código y luego es compilado para el sistema operativo en el cual se va a ejecutar la aplicación.

Aplicaciones Web: Se ejecutan en los navegadores web y poseen un diseño que facilita la interacción mediante dispositivos móviles, se comportan de manera responsiva y son programadas utilizando lenguajes de programación que permiten interactuar con el hardware del dispositivo, permitiendo enviar notificaciones *push*.

Aplicaciones Híbridas: Estas aplicaciones funcionan como contenedores nativos de una aplicación web, permitiendo, reutilizar código, utilización del hardware del dispositivo y una mayor compatibilidad.

2.1.2 Patrones arquitectónicos del Desarrollo de aplicaciones móviles:

Un patrón arquitectónico posee una definición similar al de un patrón de diseño de software, es decir, un conjunto de **reglas reutilizables** que son aplicables independiente del Framework o librerías.

Tipos de patrones:

Model-View-Controller (Modelo-Vista-Controlador): Este patrón busca separar la aplicación en tres partes:

Modelo: Representa los datos almacenados, las consultas a una API y es quien envía estos datos a una Vista.

Vista: Comúnmente definida como la Interfaz de Usuario y puede interactuar directamente con el modelo.

Controlador: Es el intermediario de la Vista y el Modelo, responde las solicitudes generadas por las interacciones del usuario y recarga la Vista.

Model-View-Presenter (Modelo-Vista-Presentador):

Modelo: Contiene todos los datos de la aplicación y la lógica del negocio

Vista: Muestra los datos que recibe el presentador y redirige a él toda la interacción recibida por parte del Usuario.

Presentador: Es el intermediario, este contiene la lógica de la Vista, envía las solicitudes al modelo.

Model-View-ViewModel (Modelo-Vista-ViewModel):

Modelo: Contiene los datos que se mostrarán en la interfaz de Usuario.

Vista: Solo muestra los datos y toda la interacción la envía al ViewModel

ViewModel: Envía solicitudes al modelo y contiene la lógica de la vista, pero, está “desacoplado” de ella. [3]

2.2 Sistema Operativo:

Al igual que las computadoras, los dispositivos móviles cuentan con un sistema operativo encargado de gestionar los recursos de hardware asignados a los diferentes programas. En la actualidad, este mercado está dominado por dos competidores principales: Android e iOS.

2.2.1 Android: Es un sistema operativo basado en Linux, de código abierto enfocado principalmente a los teléfonos inteligentes, pero su uso se extiende a tabletas, relojes inteligentes, televisores inteligentes y automóviles. La primera versión de Android fue lanzada al mercado por Google en el año 2008. Al ser de código abierto las empresas fabricantes de Smartphones desarrollan sus propias versiones de Android, adaptándolas a su hardware. Lo que conlleva a que cada fabricante deba realizar actualizaciones a su versión de Android, lo que provoca que muchos dispositivos queden sin actualizaciones.

Actualmente, el entorno de ejecución de las aplicaciones de Android es Android Runtime (ART) que trabaja sobre un Framework llamado Android Framework, que es el encargado de ofrecer los servicios esenciales que una aplicación necesita para utilizar los recursos y componentes del sistema. [4]

2.2.2 iOS: Es un sistema operativo desarrollado por Apple, utilizado en los dispositivos móviles iPhone, iPad y iPod de Apple. Su primera versión fue lanzada el año 2007. Este es un sistema operativo de código cerrado, es decir, solo los desarrolladores de Apple tienen acceso a él y solo ellos pueden realizar modificaciones.

2.3 Lenguaje de programación: Es un conjunto de instrucciones y reglas que son entendidas y ejecutadas por un computador, éstos poseen reglas de escritura y semántica para que puedan ser interpretadas correctamente por un computador. [5]

2.3.1 Dart: Es un lenguaje de programación, de código abierto creado por Google en el año 2011, diseñado para ser utilizado en la creación de aplicaciones web y móviles. Está optimizado para tener un gran rendimiento en varias plataformas y es completamente orientado a objetos. [6]

2.4 Framework: Un Framework es un conjunto de herramientas y librerías reutilizables para desarrollar algo en específico, ahorrando tiempo en el proceso, ya que poseen estructuras ya predefinidas. En el desarrollo móvil existen 2 frameworks principales enfocados en el desarrollo en Android: Flutter y React Native. [7]



2.4.1 React Native: El framework lanzado por Meta (antes Facebook) en el año 2015, enfocado en el desarrollo móvil, es de código abierto, utiliza JavaScript como lenguaje de programación y React (la librería de JavaScript para construir interfaces de usuario). [8]

2.4.2 Flutter: Es un framework de código abierto lanzado por Google en el año 2017, Flutter utiliza Dart como lenguaje de programación y permite desplegar aplicaciones para diversas plataformas (Web, iOS, Android) con una sola base de código. Flutter para construir su interfaz de usuario utiliza una estructura llamada Widget, la cual representa a todos los elementos visibles en la pantalla. [9] Flutter utiliza el motor gráfico de renderizado Skia, de Google, lo que le otorga un rendimiento superior, ofreciendo un rendimiento similar al de las aplicaciones nativas. [10]

2.5 Backend y API:

El Backend es la parte no visible para un usuario de una aplicación o sistema, contiene la base de datos y la interacción entre los datos del sistema. Por otra parte, un API (Application Programming Interface) posee estrecha relación con un backend, ya que es el puente entre la interfaz visible por un usuario y el backend, este posee las reglas que definen como deben interactuar las diferentes partes de un sistema. [11]

2.5.1 PostgreSQL: PostgreSQL es una base de datos relacional de código abierto, lanzada en el año 1996. Lleva muchos años presente en el mundo de las bases de datos, lo que la convierte en una de las opciones preferidas por los desarrolladores ya que está disponible para los tres sistemas operativos más utilizados (Windows, Mac y Linux), además posee una gran comunidad y una vasta documentación.

2.5.2 Django REST Framework: Django, que utiliza el lenguaje de programación Python, posee un Framework que permite construir una API Web, éste permite configurar recursos de backend para que sea sencillo interactuar con ellos, este framework no se limita a solamente un patrón de diseño, sino que se basa en varios, por ejemplo, el modelo-vista-serializador (MVS). Este modelo permite obtener los datos en un formato JSON, además de configurar las diversas solicitudes HTTP para la obtención y modificación de los datos.

2.6 Servicios de Mapas: Consisten en la representación geográfica y espacial en un formato visual, a través de un mapa.

2.6.1 MapQuest: Lanzado en 1996, fue uno de los primeros servicios de mapas en internet. Proporcionaba direcciones e información sobre rutas, con un enfoque en mapas diseñados para ser impresos.

2.6.2 Google Maps: Lanzado en 2005, cambió los mapas por internet, ya que ofreció una interfaz de usuario fácil de utilizar, además introdujo en ese año la Google Maps API, que permitió a los desarrolladores integrar Google Maps en sus aplicaciones y páginas web.

2.6.3 Mapbox: Es un proveedor de mapas por internet, comenzó en 2010 y es una gran alternativa a Google Maps. Mapbox provee APIs para integrar en las aplicaciones web, de escritorio y móviles. Además de facilitar un editor de mapas, abriendo la posibilidad de personalizar el estilo del mapa a mostrar

2.7 Geolocalización: Es el proceso que permite determinar la ubicación geográfica de un dispositivo, éste es posible gracias a varias tecnologías, tales como: GPS, Wifi y Redes Móviles. Dando a conocer las coordenadas (latitud y longitud).

2.8 Enrutamiento: Se refiere al proceso de encontrar la ruta óptima para dos puntos geográficos, basándose en parámetros, como, por ejemplo: la distancia, cantidad de giros y la duración estimada.

2.8.1 Mapbox Directions API: Posee indicaciones para llegar al destino asignado, permite variar el tipo de direccionamiento dependiendo del tipo de movilización (caminando, bicicleta y vehículo) que se utilice. La ruta se marca mediante un conjunto de puntos, los cuales son retornados por la API en formato JSON.

2.8.2 Google Maps Directions API: Es el servicio de direccionamiento de Google, permite obtener indicadores (al igual que Mapbox) dependiendo del medio de transporte que se utilice. La API retorna la ruta en formato JSON y XML.

2.9 Optimización de Rutas: Es la obtención de la ruta óptima entre un origen y varios destinos, posee varias aplicaciones, como la logística y la navegación.

2.9.1 Mapbox Optimization API: Esta es la API de optimización de rutas de Mapbox, la diferencia entre la Directions API radica en que la Optimization API soporta múltiples destinos o paradas, así que éstas son consideradas para el cálculo de la ruta óptima para llegar a todos los destinos determinados.

2.9.2 Google Maps Route Optimization API: Esta API de Google puede calcular la ruta óptima para varios destinos de una flota de vehículos, considera atributos propios de un vehículo (peso de la carga, capacidad del vehículo) para realizar los cálculos.

3. Metodología

3.1 Enfoque para el desarrollo

Para el desarrollo del Sistema, se optó por aplicar una metodología de **desarrollo ágil Scrum**. **Scrum** es un marco de desarrollo ágil enfocado al desarrollo de software, posee un enfoque iterativo e incremental en donde el desarrollo se divide en pequeños periodos, llamados “sprints”.

Artefactos de Scrum: Se define como un artefacto de Scrum cada elemento que representa al trabajo por completar y el trabajo terminado, el cual sirve para medir el progreso del proyecto. Lo que permite una toma de decisiones y planificación. Los artefactos principales son los siguientes:

Product Backlog: Es una lista que muestra lo que se necesita para mejorar el producto

Sprint Backlog: Es un conjunto de tareas a partir del Product Backlog, que definen los objetivos para ese Sprint

Incremento: Es el resultado obtenido al finalizar un Sprint.

El equipo de Scrum consiste en:

Scrum Master: Es el encargado de conocer y aplicar la teoría del Scrum, motivando al equipo y aplicando las prácticas para una correcta aplicación del Scrum.

Product Owner: Posee la responsabilidad de tener la mayor comunicación con el cliente, es quien debe desarrollar y siempre tener presente el objetivo del proyecto.

Desarrolladores: Son los miembros del equipo encargados de crear las funcionalidades propuestas en cada entrega.

Eventos de Scrum: Son instancias para aplicar o verificar cada artefacto de Scrum, los eventos principales son los siguientes:

- **Sprint Planning:** En esta reunión se definen las tareas a realizar para ese Sprint y marca su inicio.
- **Sprint Review:** Es la instancia definida para ver y comparar el resultado del Sprint respecto a lo definido en su inicio.
- **Daily Scrum:** Son reuniones diarias, con la intención de verificar el progreso hacia el objetivo definido en la Sprint Planning.
- **Sprint Retrospective:** Esta es la reunión que marca el final de un Sprint y busca establecer las mejoras para futuras entregas. [12]

3.2 Herramientas y entornos de desarrollo utilizados

Para el desarrollo de esta aplicación móvil se utilizarán las siguientes tecnologías:



- **Flutter:** Será utilizado para la construcción de la interfaz de usuario y la implementación de los componentes como SDKs.
- **Dart:** Este es el lenguaje de programación base de Flutter y será utilizado para programar la lógica de la aplicación y la interacción con las APIs.
- **Google Maps API:** Se utilizará para la obtención de las rutas y la geocodificación de las calles y puntos
- **VSCode:** Este IDE (Entorno de Desarrollo Integrado) será utilizado para la programación de la aplicación y la depuración de ésta ya que posee integración con emuladores de Android.
- **GitHub/Git:** La plataforma de desarrollo cooperativo y sistema de control de versiones que será utilizada para almacenar el proyecto.
- **PostgreSQL:** El Sistema (completo) utilizará esta base de datos, está contendrá la lógica del negocio y se utilizará mediante el Framework **Django REST**.

3.3 Plan de trabajo

El desarrollo de esta aplicación se apegará al patrón de desarrollo Modelo-Vista-Controlador

Este desarrollo al ser trabajo bajo la metodología ágil Scrum, se dividirá en tres entregas, que son las siguientes:

Entrega 1 (1 Mes):

- Comprensión del problema
- Investigación de plataformas y recursos
- Integración Maps SDK
- Conexión API Backend

Entrega 2 (1 Mes):

- Utilización APIs Google Maps Platform
- Creación de interfaz de navegación asistida
- Mejoras en interfaz de usuario

Entrega 3:

- Aplicación guía de estilos
- Autenticación e inicios de sesión
- Envío y recepción de datos de la aplicación web

4. Desarrollo

4.1 Diagrama de modelo de datos relacional

Este es el diagrama que muestra como está estructurada la base de datos, la cual NeoRouteApp interactúa mediante solicitudes a la **API REST**

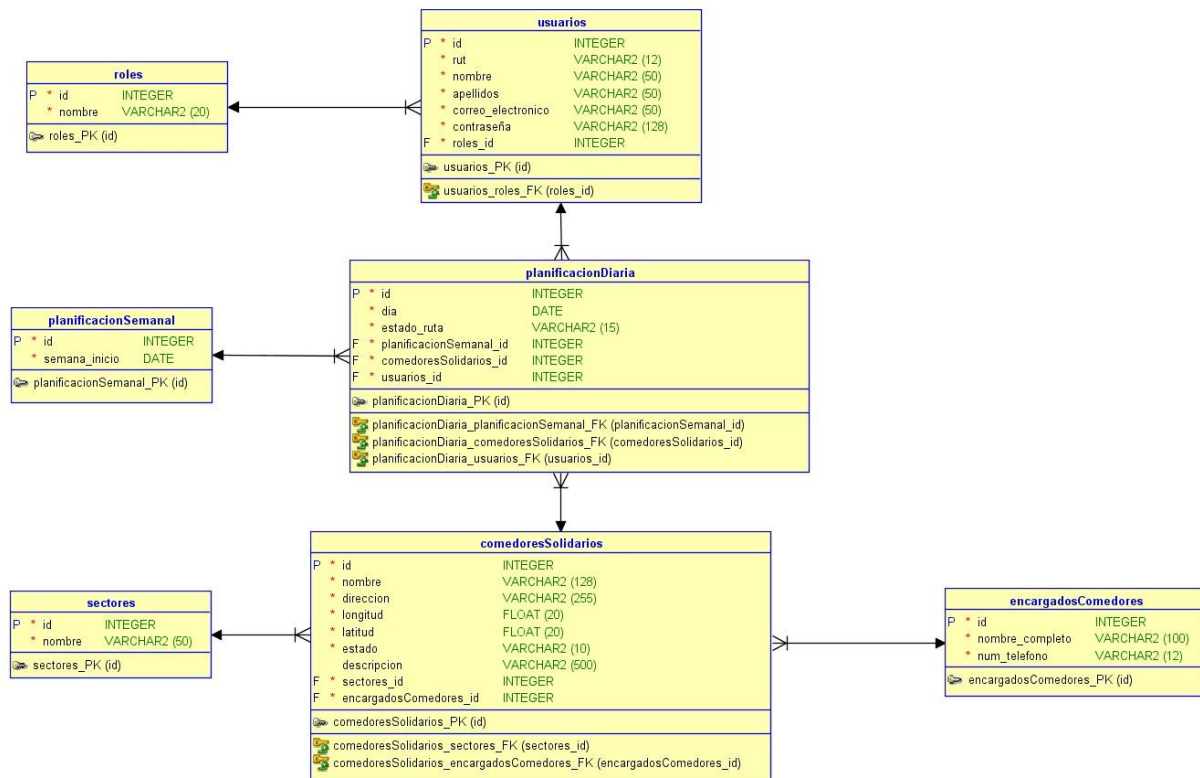


Figura 3. Diagrama de modelo de datos relacional

Fuente: Carlos Donoso – Desarrollo de sistema Backend para Optimización de Rutas y Gestión de Entregas en Comedores Solidarios de Viña del Mar

4.2 Arquitectura de la aplicación móvil

Como se especificó anteriormente, el desarrollo será hecho utilizando la arquitectura Modelo-Vista-Controlador.

4.2.2 Modelo

Cada modelo representa los datos obtenidos desde la API del Backend, estos datos son generados desde el entorno web, por el Administrador.

Clases:

- Usuario:** Posee los datos que el sistema utilizará para un usuario conductor, además de la función que permite crear una instancia de un usuario
- Encargado:** Posee la representación de un encargado de un comedor, con el nombre y su número telefónico, además de la función que permite crear una instancia del modelo
- Sector:** Esta es la representación de un Sector, con un identificador y el nombre, además de la función que permite crear una instancia de este modelo
- Comedor:** Modelo para representar un comedor, con los datos relevantes de éste, como el nombre, coordenadas, al sector que pertenece, encargado (utiliza el modelo Encargado) y la dirección. Además de una función que retorna una instancia del Comedor
- Planificaciones Diarias:** Este modelo representa las planificaciones diarias generadas por un Administrador, posee datos relevantes, como la fecha a la pertenece la planificación, el estado de esta planificación (si fue completada, está en camino o acaba de ser asignada), una lista de comedores que serán abastecidos en esa fecha (Se utiliza el modelo de Comedores) y el sector en el que se realizará el abastecimiento (se usa el modelo Sector). Además, incluye una función que permite generar una planificación diaria.



4.1.3 Controlador

Intermediario entre el Modelo y la vista, este define qué tipo de datos se deben mostrar en la vista y también posee las funciones que permiten obtener las instancias de los modelos, ya que los datos que componen cada modelo están en una API Backend. Cada controlador posee las funciones respectivas que permiten obtener los datos desde esa API.

Clases:

- a. **Controlador Usuario:** Este controlador utiliza una instancia del modelo Usuario, además de un valor booleano que permite controlar si este está autenticado o no. Las funciones que posee permiten comunicarse con el API para realizar el login y actualizar el modelo Usuario si las credenciales enviadas son correctas, guardar el token de sesión y obtener el token de sesión. Los tokens y los datos del Usuario autenticado son guardados dentro de la memoria del dispositivo para ser utilizados posteriormente.
- b. **Controlador Encargado:** Los encargados de los comedores no poseen interacción con la aplicación ni con el sistema, entonces se guardan sus datos en un modelo para mantener la consistencia, estos no poseen credenciales de ingreso. Este controlador posee una lista de Encargados, una función que llama a la API para obtener todos los encargados y actualizar la lista con ellos. Este controlador cuando se inicializa automáticamente ejecuta la función mencionada. Además, posee una función que permite encontrar un encargado que esté en la lista.
- c. **Controlador Sector:** Los sectores separan un conjunto de comedores de otro. Este controlador posee una lista de Sectores, esta lista se actualiza cuando se ejecuta una función que obtiene todos los sectores presentes en la API backend. También una función que permite retorna un sector dado su identificador.
- d. **Controlador Comedor:** Este Controlador utiliza el controlador de Encargados, este se utiliza para obtener una instancia de un Encargado para un comedor. Posee una función que llama a la API Backend y obtiene los comedores, retorna una lista de Comedores. Una función que retorna un Comedor dado su identificador y una función que dado una lista de Comedores retorna exclusivamente sus coordenadas dentro de un tipo de dato reconocible por el mapa.
- e. **Controlador Planificación Diaria:** El controlador posee una lista observable de Planificaciones Diarias, utiliza el controlador Usuario (para obtener el id del usuario), el controlador Sector para obtener un sector y genera una instancia del controlador Comedor. Posee una función que permite obtener todas las planificaciones asociadas a un Usuario dado el id de este usuario, comunicándose con el API Backend y retornando una lista de Planificación Diaria, los controladores especificados se utilizan para darle al modelo Planificación Diaria las instancias de los demás modelos requeridos por este. Cuando el controlador se inicializa se ejecuta esta función.
- f. **Controlador Ruta:** Este controlador es sumamente relevante para el funcionamiento del Mapa, este controlador permite manejar las variables, datos y funcionalidades que requiere el Mapa. Posee datos observables como los marcadores (marcan la ubicación de los comedores), polilíneas, variables que manejan la cámara del Mapa y las coordenadas de las paradas (estas son las que representan las coordenadas de cada comedor en una ruta). Además de una serie de funciones, que permiten obtener los marcadores, calcular la ruta (llamando a la API de Google Maps), obtener la ubicación actual, entre otras.
- g. **Controlador Navegación:** Este controlador depende del controlador del Mapa, posee las funcionalidades referentes a las instrucciones de navegación y redireccionamiento en caso de que suceda. Estas dos funciones requieren la comunicación con la API de Google Maps. Las instrucciones de navegación son almacenadas juntamente con los puntos geográficos (latitud y longitud) a los cuales pertenece, entonces se realizan las consultas para medir la distancia entre la ubicación del usuario y la coordenada perteneciente a la instrucción, superado un umbral definido por una función, se muestra la instrucción siguiente.

4.1.4 Vista

La vista es la interfaz de usuario (UI) y la capa visual con la que interactúa el usuario. En Flutter, las vistas están compuestas principalmente por Widgets.



- a. **App Initializer:** Esta es la vista de inicio, en esta vista se verifica si el usuario está autenticado, esto se hace mediante la inicialización del Controlador Usuario, así también se inicializan los controladores de Encargado y Sector. Posee un mensaje de bienvenida y un botón para comenzar. Si el Usuario está autenticado lo llevará a la vista de Planificaciones Diarias, de lo contrario se redirigirá a la vista de login.
- b. **Login:** Esta vista se solicita el ingreso de las credenciales del Usuario (Conductor), si son correctas se almacenan para posterior uso, esto se logra mediante el uso del Controlador Usuario, el cual es llamado en esta vista y es dirigido a la vista de Planificaciones Diarias.
- c. **Planificaciones Diarias:** En esta vista se muestran las planificaciones que fueron asignadas a el Conductor, para obtenerlas se inicializa el Controlador Planificaciones Diarias y muestra una lista con las planificaciones obtenidas del controlador con información relevante de estas. Si el conductor selecciona una de ellas es dirigido a la lista de Ruta View.
- d. **Ruta View:** Cuando el conductor selecciona alguna planificación con entregas a realizar, lo lleva a esta vista, muestra un mapa, que es una instancia de Google Map Widget, que muestra su ubicación actual y una ruta sugerida que pasa por cada comedor presente dentro de esa planificación. Cada comedor está destacado con un marcador, en el cual se puede conocer el nombre del comedor al presionar el marcador. Además, posee un botón que permite comenzar la navegación asistida, que muestra dentro de la vista instrucciones que permiten llegar a cada comedor. También un botón en la parte superior derecha que le permite conocer a detalle cada comedor presente en la planificación, mostrando datos como el Encargado del comedor, dirección y además permite finalizar la Ruta, con un botón inferior.

4.3 Diagrama de clases

A continuación, se presenta el diagrama de clases, que muestra cada una de las clases de la aplicación móvil y sus métodos:

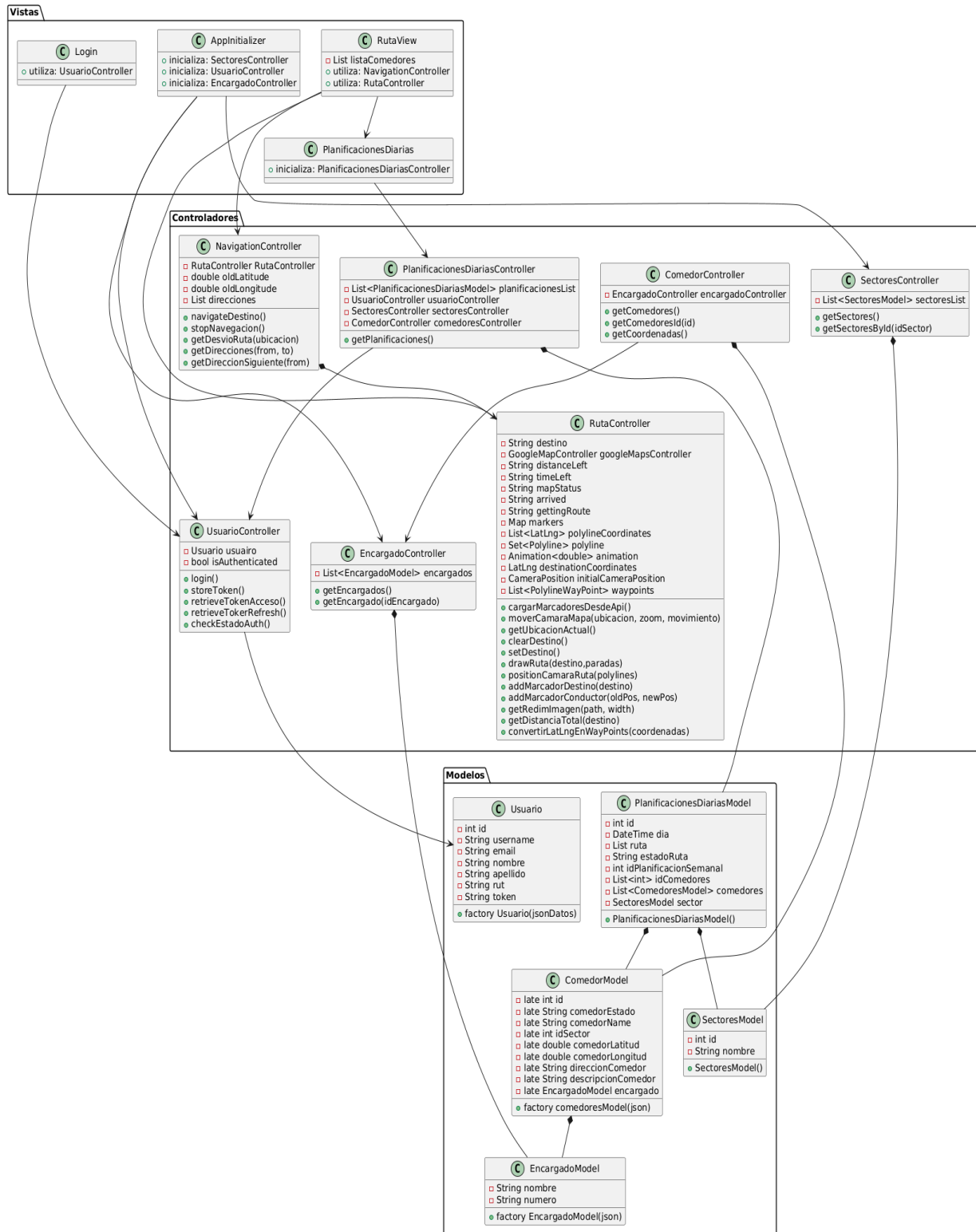


Figura 4. Diagrama de Clases
Fuente: Elaboración propia

4.4 Diagrama de Arquitectura

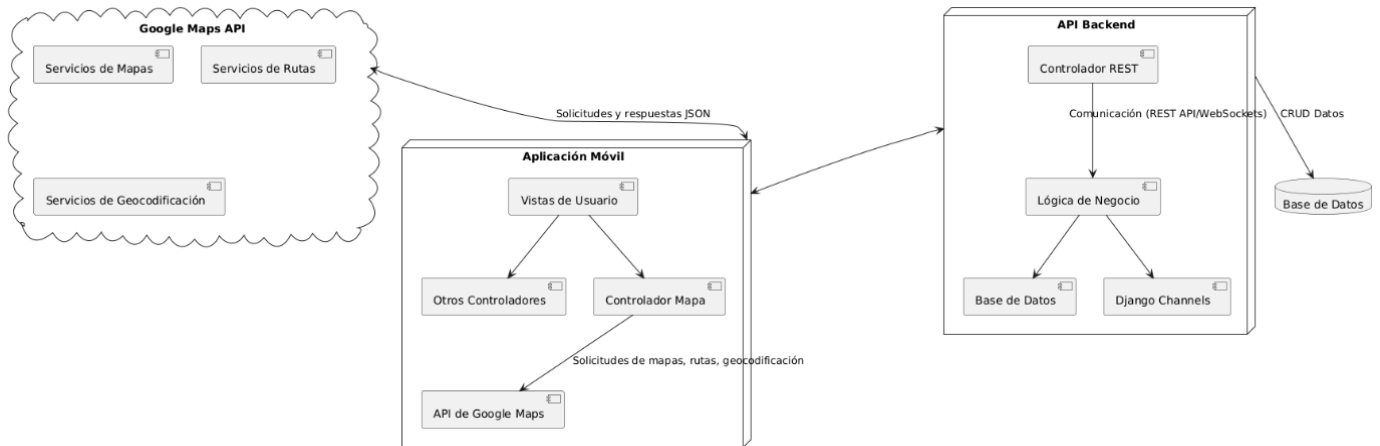


Figura 5. Diagrama de arquitectura
Fuente: Elaboración propia

4.5 Patrones de Diseño

Dentro de la arquitectura se utilizan dos patrones de diseño dentro de 2 clases diferentes.

4.5.1 Patrón Facade

El Patrón de diseño Facade sirve para proveer una interfaz simplificada a una biblioteca compleja, proveyendo solamente las funcionalidades relevantes para el Cliente y el funcionamiento de la Aplicación. [13] En este desarrollo se utilizó este patrón de diseño dentro de Ruta Controller que permite interactuar con las bibliotecas y APIs de Google Maps, además de proveer interacción con los componentes presentes dentro de una instancia de Google Map en la Vista Ruta View. A Continuación, se presenta el diagrama que muestra el patrón aplicado:

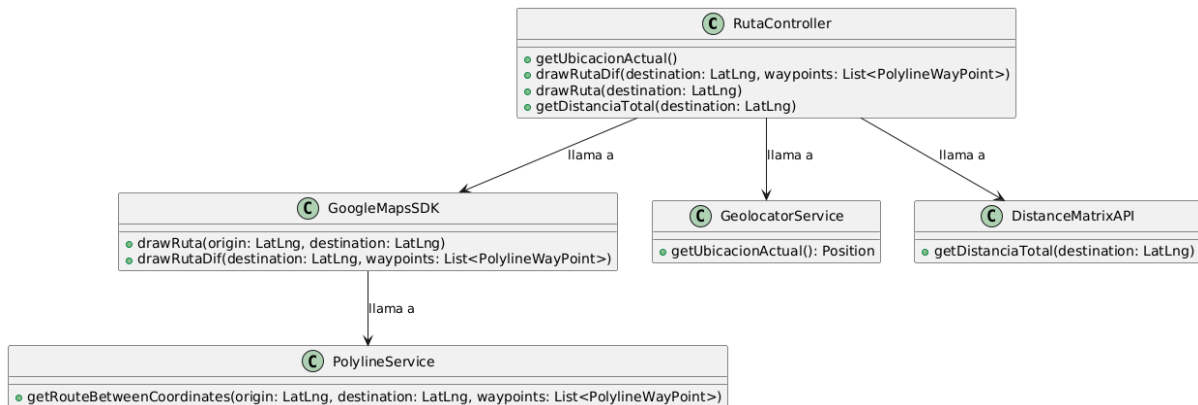


Figura 6. Patrón Facade
Fuente: Elaboración propia

4.5.2 Patrón Factory

El patrón Factory permite crear objetos en una superclase mientras permite a las subclases alterar el tipo de objetos que se crearán. [14] Dentro de la clase Modelo Planificaciones Diarias, se estableció que una Planificación posee Comedores (que poseen encargados) y un Sector. Por lo tanto, se optó por separar a Comedores, Sectores y Encargados de esta clase superior. Este es el diagrama que muestra el patrón aplicado:

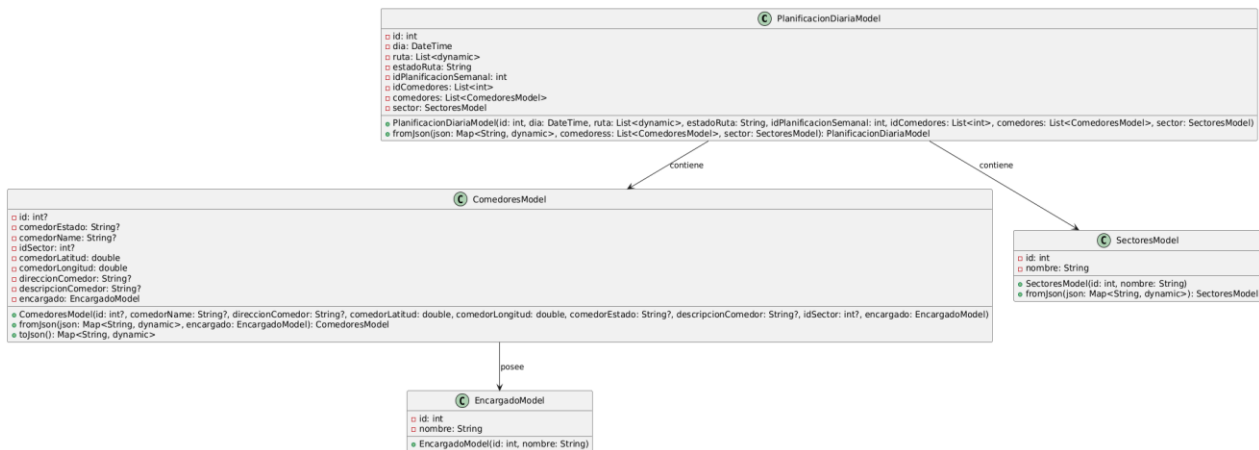


Figura 7. Patrón Factory
Fuente: Elaboración propia

4.6 Diseño de la Interfaz de Usuario (UI)

El diseño de NeoRouteApp fue realizado considerando el usuario objetivo: El Conductor, por lo tanto, se decidió utilizar botones y gestos para la navegación dentro de la aplicación. Por ejemplo, para interactuar con el mapa es posible utilizar los botones si se desea hacer zoom, como también realizar el gesto de pellizcar.

Evitando en lo posible la escritura a mano. Así también con un diseño sencillo y directo, mostrando solo la información relevante para el Conductor.

Se prefirió un diseño coherente, utilizando una paleta de colores y tipografía en toda la aplicación, reforzando el reconocimiento de los botones, texto informativo y evitando la sobrecarga cognitiva. Cada componente, como los botones y cuadros informativos se adaptaron al ecosistema Android, utilizando la librería de Material Design. Material Design posee una directriz de diseño, una de ellas es el botón de retroceso para volver a la pantalla anterior. Lo que fue considerado en el diseño y aplicado en el desarrollo.

A continuación, se presenta la interfaz de usuario, se presentará el diseño hecho en formato *mockup* y su implementación real:

4.6.1 App_INITIALIZER



Figura 8. Mockup - App_INITIALIZER

Fuente: Sebastian Tang - Diseño de interfaces y experiencia de usuario de un sistema de gestión de transporte para el abastecimiento de comedores solidarios

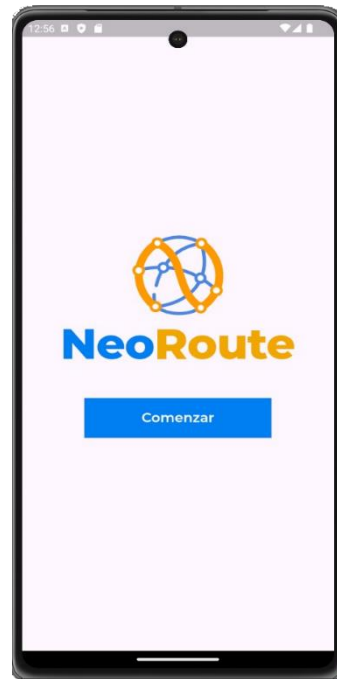
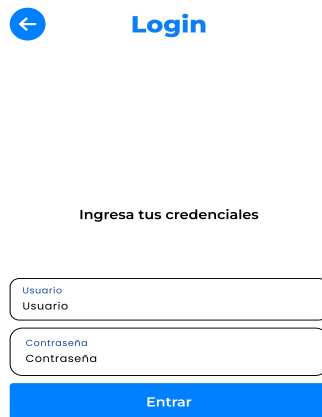


Figura 9. NeoRouteApp – App_INITIALIZER

Fuente: Elaboración Propia

App_INITIALIZER, provee el logo de la aplicación y su nombre. Además de presentar los colores característicos de la aplicación. Como se explicó en la sección de **Arquitectura de la aplicación móvil** esta es una vista que posee funcionalidades que el usuario no ve ni requiere conocer que suceden. Esto permite que la experiencia de Usuario no se vea afectada.

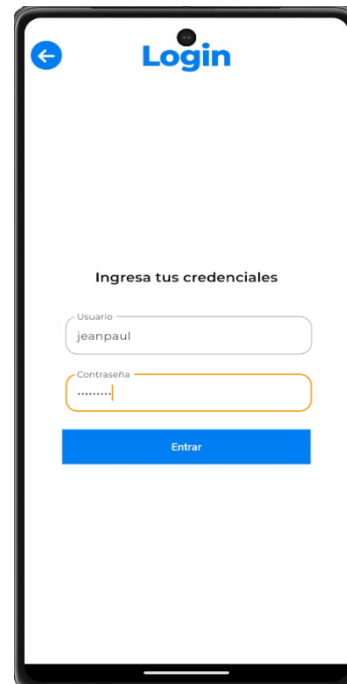
4.6.2 Login



A mockup of a login screen. At the top left is a blue circle with a white left-pointing arrow. To its right is the word "Login" in blue. Below this is the text "Ingresa tus credenciales". There are two input fields: the first is labeled "Usuario" and contains the text "Usuario"; the second is labeled "Contraseña" and contains the text "Contraseña". Below these fields is a blue button with the text "Entrar" in white.

Figura 10. Mockup - Login

Fuente: Sebastian Tang - Diseño de interfaces y experiencia de usuario de un sistema de gestión de transporte para el abastecimiento de comedores solidarios



A screenshot of the NeoRouteApp login screen. At the top left is a blue circle with a white left-pointing arrow. To its right is the word "Login" in blue. Below this is the text "Ingresa tus credenciales". There are two input fields: the first is labeled "Usuario" and contains the text "jeanpaul"; the second is labeled "Contraseña" and contains the text ".....". Below these fields is a blue button with the text "Entrar" in white.

Figura 11. NeoRouteApp - Login

Fuente: Elaboración Propia

Login, se puede acceder a esta pantalla solo si no se ha iniciado sesión, provee de formularios con validaciones en tiempo real, por ejemplo, en el formulario de ingreso para el nombre de usuario sólo se permiten letras y números, sin espacios ni símbolos. Esto permite 2 cosas importantes dentro de la experiencia de usuario y la seguridad de la API, al ser la validación en tiempo real esto permite que el usuario se dé cuenta inmediatamente que ha completado uno o más campos de forma errónea. Cada campo posee su propia validación, por lo tanto, si el usuario se equivoca en llenar un campo y no otro, la aplicación le mostrará un aviso en el campo en cuestión. La seguridad de la API en los campos de formularios es muy importante, ya que cuando se llena un formulario (especialmente los de inicio de sesión) se llevan estos datos a la base de datos para que esta los valide y entregue la respuesta correspondiente. Con esta validación dentro de los campos se evitaría la inyección de código, puesto que los datos tendrían que pasar la validación del formulario (propio de la vista de Login) antes de ser enviados a la API Backend.

4.6.3 Planificaciones Diarias



Figura 12. Mockup - Planificaciones Diarias
Fuente: Sebastian Tang - Diseño de interfaces y experiencia de usuario de un sistema de gestión de transporte para el abastecimiento de comedores solidarios



Figura 13. NeoRouteApp – Planificaciones Diarias
Fuente: Elaboración Propia

Planificaciones Diarias, muestra las planificaciones asignadas al Conductor que ha iniciado sesión, están ordenadas por fecha, de más antigua a la más reciente. Las Planificaciones poseen solo los datos relevantes para el Conductor, como lo es la fecha, el sector y la cantidad de comedores que requieren que se les entreguen los recursos. Además, muestra el estado de cada una con colores que poseen como objetivo transmitir visualmente diferentes estados emocionales o de urgencia, facilitando la comprensión rápida y la toma de decisiones, como el verde para indicar éxito ("Completada") y el anaranjado para señalar advertencia y necesidad de atención ("Asignada"). Lo que es importante para el contexto y el problema que busca solucionar la Aplicación. Título de la barra superior: pese a que durante todo este proyecto se les ha llamado planificaciones o planificaciones diarias, esto no es parte del vocabulario de los Conductores, por lo tanto, para mejorar la experiencia de usuario y evitar confusiones, se decidió utilizar el título de Entregas, ya que este es más natural para los usuarios, además de que por el largo de la palabra "Planificaciones" no quedaba bien en el diseño del mockup ni en la vista dentro de la Aplicación.

4.6.4 Ruta View: Vista de Ruta

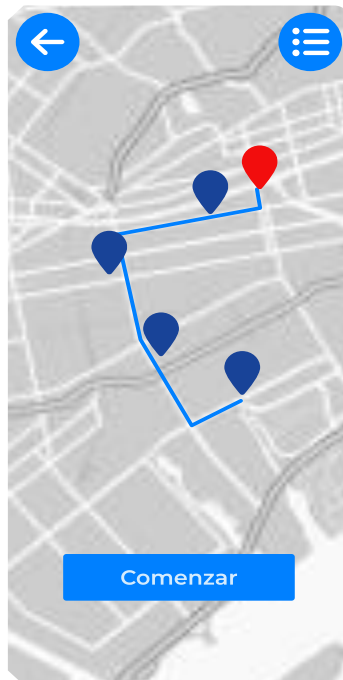


Figura 14. Mockup – Ruta View: Vista de Ruta

Fuente: Sebastian Tang - Diseño de interfaces y experiencia de usuario de un sistema de gestión de transporte para el abastecimiento de comedores solidarios

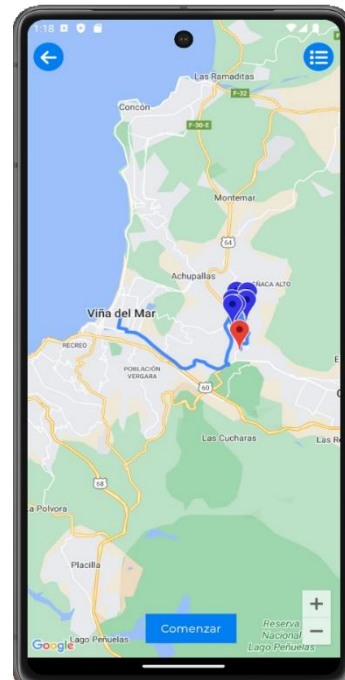


Figura 15. NeoRouteApp – Ruta View: Vista de Ruta

Fuente: Elaboración Propia

Ruta View: Vista de Ruta: La interfaz muestra un mapa interactivo que, tras realizar las consultas necesarias, despliega la ruta hacia los comedores asignados en cada planificación. Los puntos de entrega se distinguen del banco de alimentos de la municipalidad mediante un color distintivo (azul), facilitando la diferenciación de los destinos.

El botón "Comenzar" cumple una doble función: iniciar la navegación asistida e informar a la API sobre el cambio de estado de la planificación, actualizándola de "Asignada" a "En Camino". Este cambio permite que el Administrador, a través de la Aplicación Web, sea notificado del inicio del recorrido por parte del Conductor. La ubicación estratégica del botón "Comenzar" permite una interacción ergonómica con una sola mano, evitando posibles errores durante la interacción con la NeoRouteApp.

4.6.5 Ruta View: Navegación Asistida

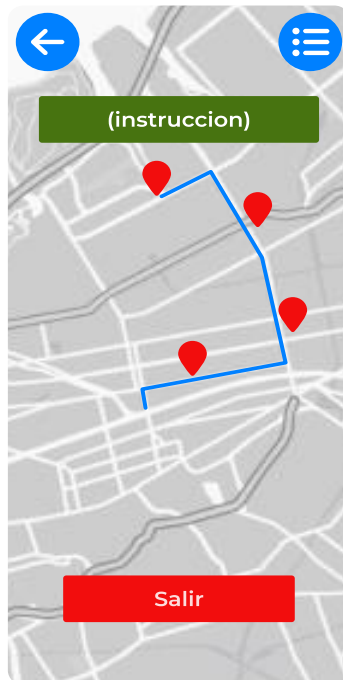


Figura 16. Mockup – Ruta View: Navegación Asistida
Fuente: Sebastian Tang - Diseño de interfaces y experiencia de usuario de un sistema de gestión de transporte para el abastecimiento de comedores solidarios

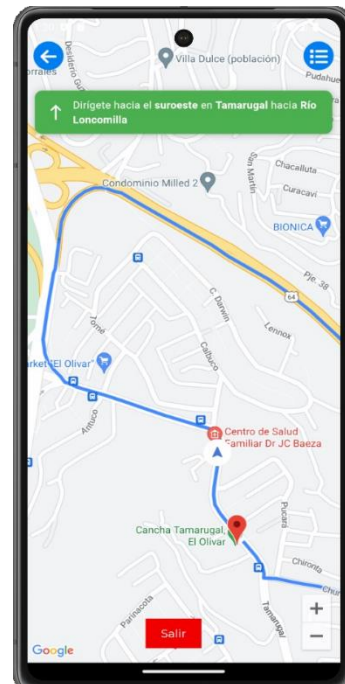


Figura 17. NeoRouteApp – Ruta View: Navegación Asistida
Fuente: Elaboración Propia

Ruta View: Navegación Asistida, cuando comienza la navegación, tras presionar el botón de comenzar, éste cambia de color y de mensaje, dando la posibilidad de finalizar la navegación de no requerirse. Cuando la navegación está en curso, se muestra un icono representativo para el usuario que es la flecha dentro del círculo blanco. Además de las instrucciones paso a paso que facilitan este proceso. El color azul para la ruta es distintivo de la textura del mapa, facilitando la visualización de la ruta.

4.6.6 Ruta View: Detalle de Entrega y finalización de Ruta

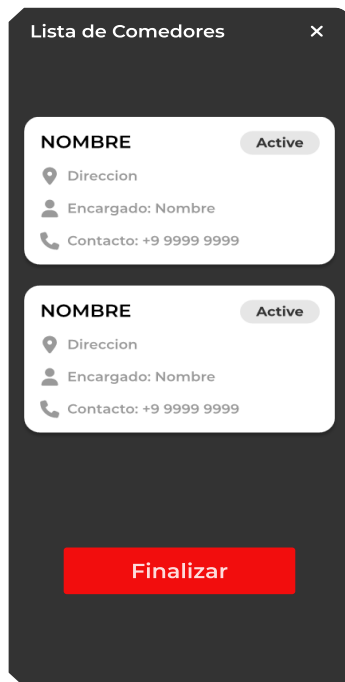


Figura 18. Mockup – Ruta View: Detalle de Entrega y finalización de Ruta

Fuente: Sebastian Tang - Diseño de interfaces y experiencia de usuario de un sistema de gestión de transporte para el abastecimiento de comedores solidarios



Figura 19. NeoRouteApp – Ruta View: Detalle de Entrega y finalización de Ruta

Fuente: Elaboración Propia

Ruta View: Detalle de Entrega y finalización de Ruta, esta es una sección de la vista de Ruta, ya que técnicamente se trata de un *Overlay* (en español, capa), el cual permite que no se redibuje la vista de la Ruta ni la Navegación, de hecho, sigue funcionando “de fondo”, lo que permite que no se vuelva a generar ningún componente ni widget, continuando con la navegación. Entonces, esto permite que el Conductor durante la navegación asistida pueda consultar esta sección para ver los comedores a los que debe realizar la entrega. Dentro de esta sección se muestran detalles de cada comedor perteneciente a una planificación, proporcionando datos relevantes para el Conductor, como puede ser el nombre del Encargado de un comedor y su número telefónico, además posee un indicador de estado del comedor, ya que es posible que el comedor se encuentre “Cerrado” pero aun recibiendo recursos para un posterior uso. Además, posee el botón que permite, como lo dice su texto: finalizar la ruta, que, al presionarse, nuevamente se comunica con el API backend indicando un nuevo cambio de estado, cambiándose éste a “Completada” refiriéndose a la Planificación y a nivel de usuario, la entrega.

5. Implementación de funcionalidades

5.1 Integración con servicios backend y APIs

El Sistema NeoRoute como conjunto (Aplicación Web y Móvil) se describe en el siguiente diagrama, destacando las interacciones que poseen:

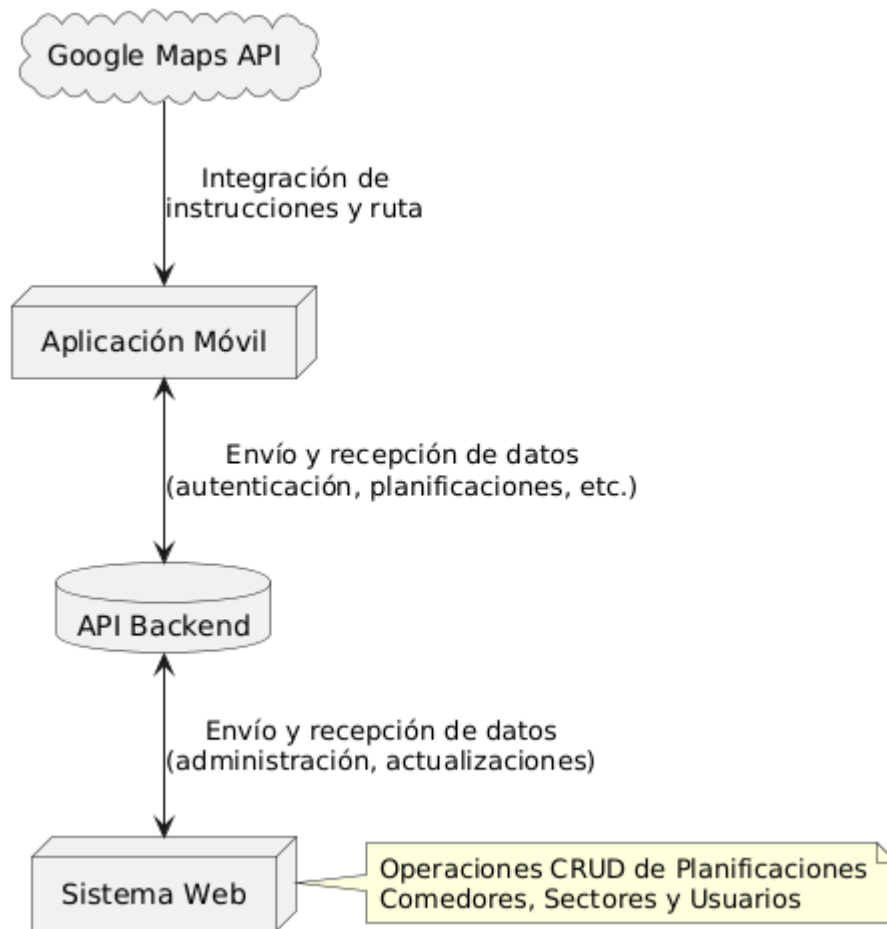


Figura 20. Diagrama de Arquitectura de NeoRoute
Fuente: Elaboración propia

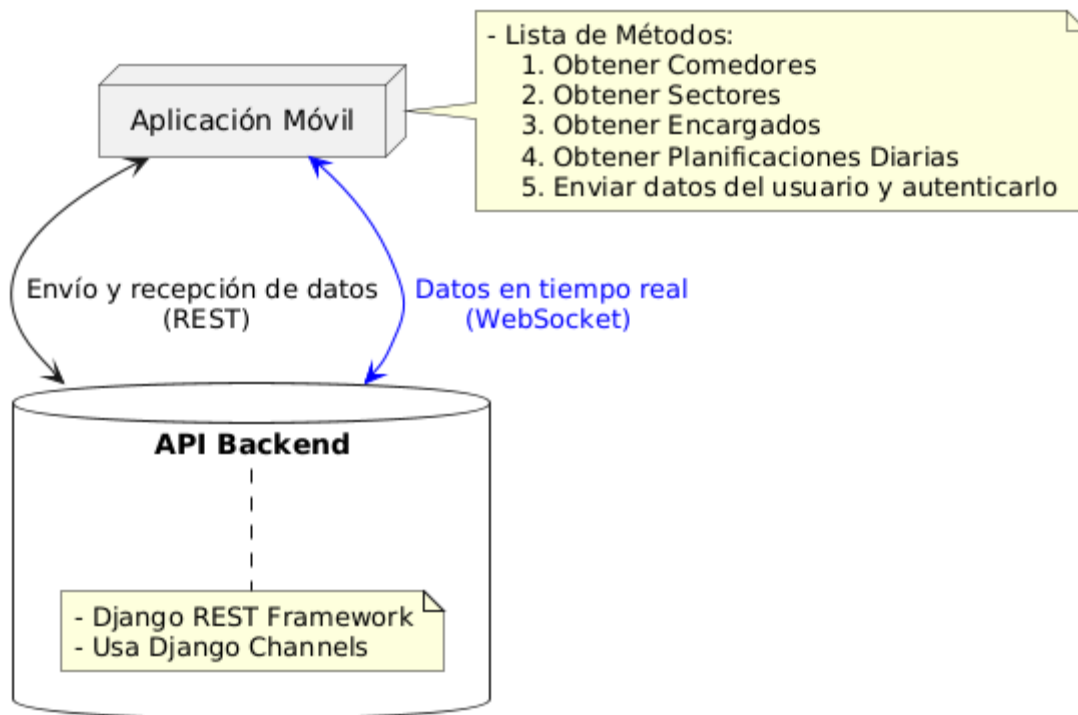
5.1.1 Backend

Como se especificó en el apartado de tecnologías a utilizar, el backend de la aplicación, desarrollado con Django REST Framework, proporciona una arquitectura de comunicación flexible y eficiente con la aplicación móvil mediante dos canales principales:

- 1- Comunicación REST
 - a. Permite enviar información estructurada mediante JSON
 - b. Soporta operaciones **CRUD** (Crear, Leer, Actualizar, Eliminar)
- 2- Comunicación **WebSocket** (tiempo real)
 - a. Conexión en tiempo real mediante la utilización de **Django Channels**
 - b. Realiza notificaciones cuando se crea y/o edita un dato dentro de un modelo

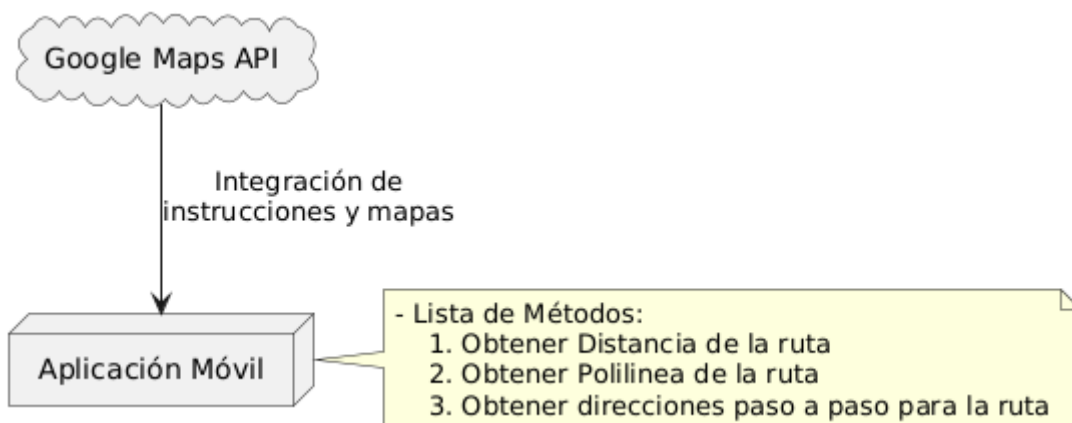
Las funcionalidades principales del Backend API son los siguientes:

1. Gestión de Comedores
2. Administración de Sectores
3. Control de Encargados
4. Manejo de Planificaciones Diarias
5. Autenticación y gestión de usuarios

**Figura 21.** Diagrama de Arquitectura Aplicación-Backend**Fuente:** Elaboración propia

5.1.2 Google Maps API

Como se especificó en el apartado de tecnologías a utilizar, la Aplicación tendrá una conexión a la API de Google Maps, con la intención de obtener los datos necesarios para construir una ruta y mostrar aquella ruta en el mapa. La respuesta del API se trata de un JSON, el cual contiene, una ruta, la que es sugerida por Google Maps como la óptima, así también con las instrucciones paso a paso de la navegación. Este es el diagrama que describe que es lo que se le solicita a esta API:

**Figura 22.** Diagrama de Arquitectura Aplicación-API Google Maps**Fuente:** Elaboración propia

6. Pruebas

Las pruebas en el desarrollo de aplicaciones móviles son sumamente importantes, ya que estas permiten determinar si las características y funcionalidades implementadas funcionan según lo previsto o no

6.1 Pruebas funcionales

Las pruebas funcionales son un tipo de prueba que evalúan las funciones, en un entorno simulado y aislado. En Flutter, principalmente, es posible realizar tres tipos de pruebas funcionales:

6.1.1 Pruebas unitarias

En este tipo de pruebas se evalúa cada “unidad” de la aplicación, que es el componente más pequeño del Software, las pruebas unitarias son automatizadas, es decir, sin intervención manual. Se mostrarán tres pruebas unitarias a componentes importantes de NeoRouteApp.

a. Prueba unitaria: Validación

Se realizó una prueba unitaria a la clase Validación que permite realizar la validación de los campos de texto para la vista de Login, se consideró esta prueba porque las validaciones dentro de los formularios son importantes. Se realizaron dos pruebas principales:

Prueba de Validación: Texto

Esta prueba validó que se ingresara un texto que cumple con los parámetros establecidos en el campo de nombre de usuario dentro de la vista de Login. Los parámetros son texto, sin espacios. Se realizó la prueba considerando dos casos: un texto que cumpla los parámetros y otro texto que no cumpla los parámetros.

Prueba de Validación: Campos vacíos

Esta prueba buscó demostrar si la clase Validación era capaz de detectar si a un campo de texto requerido era enviado sin texto en su interior. Esta prueba consideró dos casos: campo de texto vacío requerido y campo de texto vacío no requerido.

Se realizaron más casos de los presentados, ya que la clase Validación puede validar campos de texto con diversos parámetros. Esta es la salida de la prueba:

```
00:00 +0: Text Validation Tests valid text input
00:00 +1: Text Validation Tests invalid text input with numbers
00:00 +2: Email Validation Tests valid email
00:00 +3: Email Validation Tests invalid email format
00:00 +4: Phone Validation Tests valid phone number
00:00 +5: Phone Validation Tests invalid phone format
00:00 +6: Date Validation Tests valid date
00:00 +7: Date Validation Tests invalid date format
00:00 +8: Required Field Tests empty required field
00:00 +9: Required Field Tests empty non-required field
00:00 +10: Length Validation Tests text below minimum length
00:00 +11: Length Validation Tests text above maximum length
00:00 +12: IP Address Validation Tests valid IP address
00:00 +13: IP Address Validation Tests invalid IP address
00:00 +14: RUT Validation Tests valid RUT
00:00 +15: RUT Validation Tests invalid RUT
00:00 +16: All tests passed!
```

Figura 23. Prueba unitaria: Clase validación

Fuente: Elaboración propia

b. Prueba unitaria: Modelo de Planificación Diaria

Esta prueba está destinada a probar si el modelo de Planificaciones Diarias es capaz de efectivamente generar una instancia del modelo, esta prueba es importante ya que las planificaciones son la pieza

fundamental de NeoRouteApp (puesto que estas son las que se les asignan a los Conductores), cada solicitud o dato recibido de alguna entidad externa ya sea realizando un API request o recibiendo mensajes de un WebSocket son simulados, ya que las pruebas buscan ser independientes de factores externos. Se evaluaron 6 casos:

Prueba de Modelo de Planificación Diaria: Constructor crea modelo correctamente a partir de un API request

En esta prueba se intentó probar que el constructor del modelo cree correctamente una instancia de Planificación Diaria tras un API request solicitando datos.

Prueba de Modelo de Planificación Diaria: Constructor crea modelo correctamente recibiendo datos desde un WebSocket

En esta prueba se intentó probar que el constructor del modelo es capaz de crear una instancia de Planificación Diaria tras recibir un mensaje del WebSocket.

Prueba de Modelo de Planificación Diaria: Constructor maneja caso sin Comedores (lista vacía) desde un WebSocket

En esta prueba se intentó probar que el constructor del modelo es capaz de manejar el caso de recibir una lista vacía de comedores tras recibir un mensaje del WebSocket.

Prueba de Modelo de Planificación Diaria: Constructor maneja caso sin Comedores (datos inválidos) desde un WebSocket

En esta prueba se intentó probar que el constructor del modelo es capaz de manejar el caso de recibir una lista con datos inválidos de un mensaje del WebSocket.

Prueba de Modelo de Planificación Diaria: Constructor maneja caso de atributos opcionales vacíos desde WebSocket y API request

En esta prueba se intentó probar que el constructor del modelo es capaz de responder frente a recibir atributos opcionales nulos desde el WebSocket y tras realizar un API request al backend.

Estos son los resultados:

```
00:00 +0: PlanificacionDiariaModel fromJson constructor creates model correctly
00:00 +1: PlanificacionDiariaModel fromWebSocket constructor creates model correctly
00:00 +2: PlanificacionDiariaModel fromWebSocket handles empty comedores list
00:00 +3: PlanificacionDiariaModel fromWebSocket handles invalid comedores data
00:00 +4: PlanificacionDiariaModel fromJson handles missing optional id_envio
00:00 +5: PlanificacionDiariaModel fromWebSocket handles missing id_envio
00:00 +6: All tests passed!
```

Figura 24. Prueba unitaria: Modelo de Planificaciones Diarias

Fuente: Elaboración propia

c. Prueba unitaria: Controlador Usuario

Esta prueba está destinada a ver si el Controlador Usuario es capaz de cumplir la función de *login* (enviando los datos al API y creando un Modelo de Usuario) y realizar el *logout* (limpiando el modelo Usuario previamente creado). Las pruebas realizadas simulan solicitudes y datos recibidos de alguna entidad externa, ya sea un API request o mensajes de un WebSocket, ya que las pruebas buscan ser independientes de factores externos.

Prueba de Controlador Usuario: Login

Esta prueba busca que, tras recibir credenciales, almacene cada una de ellas dentro del Modelo Usuario y actualice valores característicos del controlador.

Prueba de Controlador Usuario: Logout

Esta prueba busca realizar lo contrario a lo que se busca en la prueba Login, ya que debe realizar una limpieza del Modelo Usuario y regrese los valores del controlador a sus valores por defecto.

Estos son los resultados:

```
00:00 +0: UsuarioController Login debe crear usuario con datos esperados
00:00 +1: UsuarioController Logout debe resetear el usuario
00:00 +2: All tests passed!
```

Figura 25. Prueba unitaria: Controlador Usuario

Fuente: Elaboración propia

6.1.2 Pruebas de Widgets

En Flutter, todo es un Widget, y es posible realizar pruebas de cada uno, por ejemplo, un botón en una vista. Se puede evaluar si ese widget se cargó correctamente en la vista. Se mostrarán tres pruebas automatizadas de widgets presentes en NeoRouteApp.

a. Prueba de Widget: Vista de Login

En esta prueba se busca verificar si la vista es capaz de cargar los componentes más relevantes de ella, como lo son los campos de texto y el botón. Se ingresan datos de ingreso y se navega a la siguiente pantalla del flujo.

Este es el resultado:

```
00:00 +0: Login view muestra login form y maneja navegacion entre pantallas
00:00 +1: All tests passed!
```

Figura 26. Pruebas de Widgets: Vista de Login

Fuente: Elaboración propia

b. Prueba de Widget: Vista de Planificaciones Diarias

Esta prueba evalúa si todos los componentes pertenecientes a la Vista de Planificaciones Diarias son cargados correctamente. Se evalúa el texto del AppBar, Cada planificación y si el botón de “Ver” perteneciente a cada Planificación está presente.

Este es el resultado:

```
00:00 +0: Titulo appbar
00:00 +1: planificaciones desplegadas
00:00 +2: Debería tener botones Ver funcionales
00:00 +3: All tests passed!
```

Figura 27. Pruebas de Widgets: vista de Planificaciones Diarias

Fuente: Elaboración propia

c. Prueba de Widget: Vista de Ruta: Se muestran los marcadores

Esta prueba verifica que en la vista de Ruta se carguen marcadores. Para realizar esta prueba, es necesario que se carguen comedores dentro de una planificación y que estos cuenten con coordenadas para ser mostradas en el mapa mediante marcadores. Al ser una prueba aislada, se simuló la obtención de los datos. Estos son los resultados:

```
00:00 +0: RutaView muestra marcadores correctamente
00:00 +1: All tests passed!
```

Figura 28. Prueba de Widget: Vista de Ruta

Fuente: Elaboración propia

6.1.3 Pruebas de integración

Las pruebas de integración comprenden desde vistas completas a toda la aplicación, se prueban componentes pequeños, widgets y vistas, para evaluar como estos se comportan en un conjunto. Las pruebas de integración son automatizadas y son ejecutadas en un emulador o en un dispositivo real, por lo tanto, son el tipo de pruebas más lentas.

a. Prueba de Integración: Selección de una Planificación y se navega a la vista de Ruta (mapa)

En esta prueba se utiliza la mayoría de los componentes creados para las pruebas unitarias mostradas con anterioridad, se realiza la prueba como conjunto ya que son parte del flujo que se desea poner a prueba. Se cargan comedores, se crean planificaciones, se muestran en la vista de Planificaciones Diarias, se selecciona una y se carga la vista de Ruta.

Este es el resultado:

```
Running Gradle task 'assembleDebug'... 23,4s
✓ Built build\app\outputs\flutter-apk\app-debug.apk
Installing build\app\outputs\flutter-apk\app-debug.apk... 2.111ms
00:00 +0: Se muestran planificaciones y se presiona la primera
    para luego ir al mapa
00:02 +1: (tearDownAll)
00:03 +1: All tests passed!
```

Figura 29. Prueba de integración: Selección de una Planificación y se navega a la vista de Ruta (mapa)

Fuente: Elaboración propia

6.2 Pruebas de rendimiento

Las pruebas de rendimiento son importantes para garantizar una buena experiencia de usuario, por esto, se decidió evaluar la utilización de red, rendimiento y tamaño de la Aplicación NeoRouteApp. Se utilizará la herramienta *DevTools* del framework Flutter. Se mostrarán resultados de una ejecución de la aplicación (flujo completo), la aplicación que será sometida a pruebas está compilada en modo **profile**, ya que esta es la indicada para realizar este tipo de pruebas de rendimiento y es más cercana al modo **release**.

Flutter DevTools

Flutter DevTools se trata de una suite de herramientas para *debug* y estudio del rendimiento para aplicaciones y programas de Dart y Flutter [15]

Modos de compilación de una App de Flutter

Existen tres modos en los que una aplicación móvil en Flutter se puede compilar y son representativos en el estado de desarrollo en el que se encuentra la aplicación

1. Modo Debug

Este es el modo depuración para una aplicación en Flutter, en el cual es posible ver los cambios en caliente (*hot reload*), característica sólo disponible en el modo Debug, el rendimiento en general es inferior, ya que está destinada a ser utilizada en el desarrollo y probar características y funcionalidades. El tamaño de una aplicación en modo debug es el más grande de los tres.

2. Modo Profile

En el modo Profile, algunas herramientas y características se encuentran desactivadas y el modo Profile está desactivada su ejecución en un simulador puesto que no es representativo de una ejecución real, esto es, debido a que es una ejecución cercana a el modo Release, por este motivo es la recomendada para ser sometida a evaluaciones mediante DevTools.

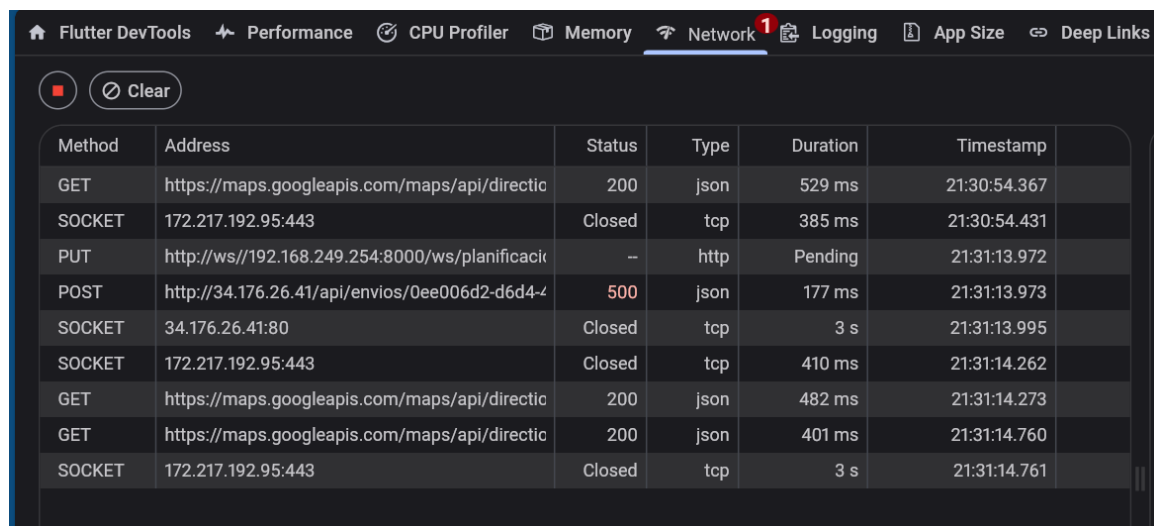
3. Modo Release

Esta es la distribución real, es decir, la versión destinada a ser utilizada fuera del entorno de desarrollo. Está desactivada la depuración por completo, los mensajes de log, no se pueden utilizar herramientas de desarrollo (DevTools) y el código está optimizado para que sea ejecutado a nivel nativo, así que, el tamaño del APK está reducido en comparación de los otros modos. [16]

Pruebas de rendimiento

a. Red

Lo importante de estos resultados son los relacionados con la **API de Google Maps**, son solicitudes del tipo GET. En la lista que se muestra en la captura se puede observar que la duración entre la solicitud y la respuesta es, en promedio de 470 ms (milisegundos). Estos resultados establecen que las llamadas a la API están bajo los parámetros aceptables.



Method	Address	Status	Type	Duration	Timestamp
GET	https://maps.googleapis.com/maps/api/directic	200	json	529 ms	21:30:54.367
SOCKET	172.217.192.95:443	Closed	tcp	385 ms	21:30:54.431
PUT	http://ws//192.168.249.254:8000/ws/planificaci	--	http	Pending	21:31:13.972
POST	http://34.176.26.41/api/envios/0ee006d2-d6d4-4	500	json	177 ms	21:31:13.973
SOCKET	34.176.26.41:80	Closed	tcp	3 s	21:31:13.995
SOCKET	172.217.192.95:443	Closed	tcp	410 ms	21:31:14.262
GET	https://maps.googleapis.com/maps/api/directic	200	json	482 ms	21:31:14.273
GET	https://maps.googleapis.com/maps/api/directic	200	json	401 ms	21:31:14.760
SOCKET	172.217.192.95:443	Closed	tcp	3 s	21:31:14.761

Figura 30. NeoRouteApp: Uso de Red

Fuente: Elaboración propia

b. Rendimiento

Se evaluará el rendimiento de la Aplicación respecto a la interfaz de usuario, cuando se crea la estructura visual. Flutter está diseñado para renderizar la interfaz cada 60 fps (cuadros por segundo). cada renderizado se llama cuadro, aproximadamente cada 16ms (milisegundos), la interfaz se actualiza para

reflejar cambios o animaciones. Si, un cuadro tarda más de 16ms en renderizarse provoca jank o saltos, es decir, una experiencia entrecortada en el dispositivo. Esto lo provocan tiempos largos de procesamiento o renderizado de los componentes presentes en la pantalla.

Utilizando **DevTools** se pueden visualizar en un gráfico los cambios de fotogramas en la aplicación de Flutter. El gráfico muestra cada fotograma, y, este está compuesto por dos barras que representan los trabajos que suceden cuando se renderiza un cuadro en Flutter, estos trabajos son principalmente dos:

UI Thread y Raster Thread

UI Thread es el encargado de manejar interacciones visuales y de usuario, es el responsable de renderizar la interfaz de usuario y actualizarla cuando suceden cambios.

Raster Thread es el código relacionado con el motor de Flutter, procesando y renderizando los gráficos entregados por el UI Thread y llevándolos a píxeles que, finalmente, son mostrados en la pantalla. [17]

Este es parte del gráfico que representa el rendimiento de la Aplicación en su inicio (Applnicializer):



Figura 31. Gráfico de Performance test – DevTools: Applnicializer

Fuente: Elaboración propia

Este es parte del gráfico que representa el rendimiento de la Aplicación en la navegación (RutaView):



Figura 32. Gráfico de Performance test – DevTools: RutaView

Fuente: Elaboración propia

¿Qué se puede extraer de estas capturas?

Tenemos que las capturas son en momentos diferentes en la aplicación, en el inicio se inicializan componentes esenciales, por eso en el inicio aparecen estos picos altos que indican *jank*, así también suelen aparecer cuando se actualiza la UI.

En la otra captura se muestra un escenario diferente, los trabajos de Raster y UI, están dentro de los parámetros normales.

6.3 Tamaño de la Aplicación

Ahora, se mostrará el tamaño de la Aplicación. Este es un tamaño que difiere de la realidad, puesto que la aplicación que está siendo sometida a estas evaluaciones está en modo Profile:

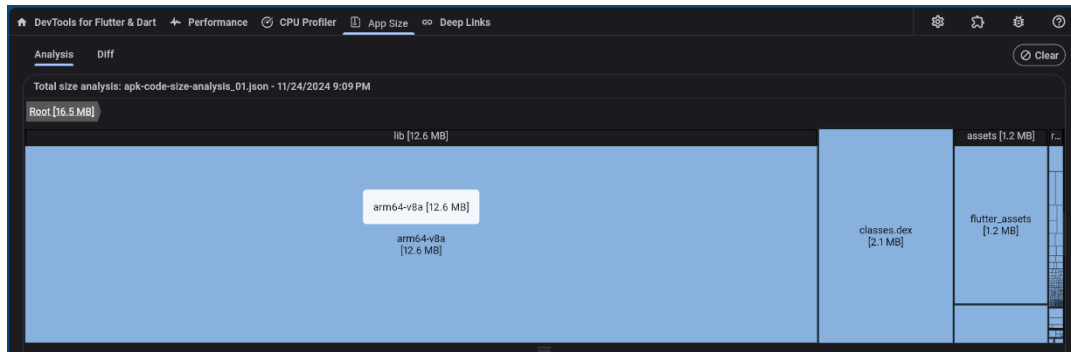


Figura 33. Tamaño de NeoRouteApp – Modo Profile

Fuente: Elaboración propia

Lo más relevante de lo que se muestra en la imagen es el tamaño total de la aplicación, que aparece en Root, que es de: 16.5 MB. Ahora, se comparará con el tamaño de la aplicación en modo Release:

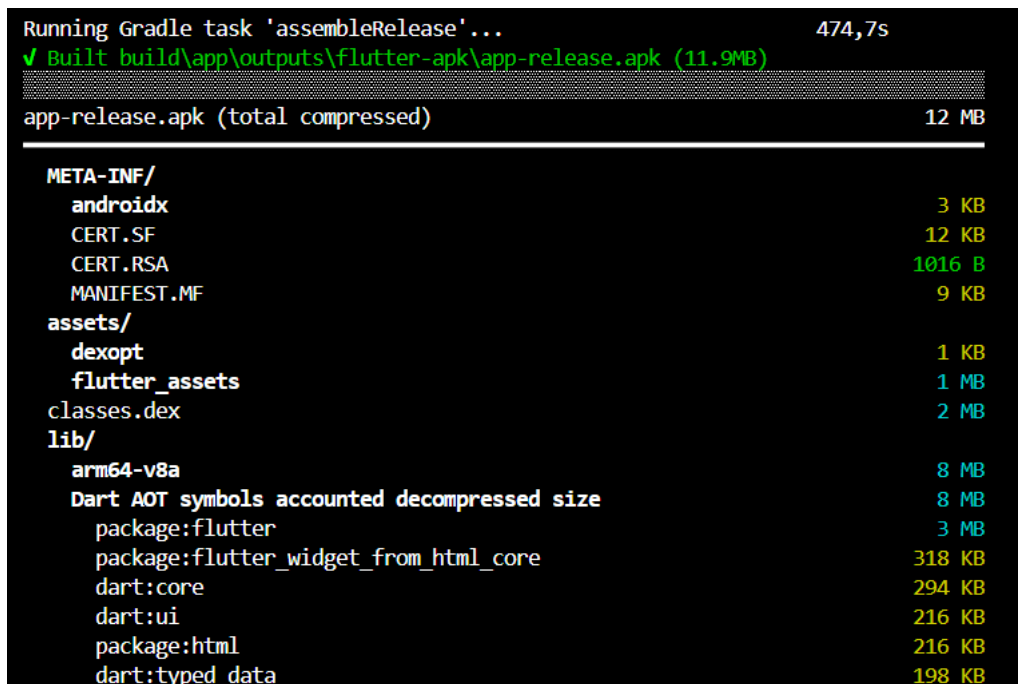


Figura 34. Tamaño de la NeoRouteApp en modo Release

Fuente: Elaboración propia

Ahora, el tamaño se ha reducido aún más a sólo 12 MB, lo que permite que la aplicación pueda ser utilizada en muchos dispositivos que no tienen espacio suficiente en memoria, esto es gracias a que la aplicación en modo Release posee optimizaciones hechas por el framework Flutter, compilando el código a lenguaje de máquina nativo [18]

7. Evaluación del desempeño de la aplicación.

Como se observó en las pruebas de rendimiento, a la aplicación le falta optimización respecto a que partes de la pantalla son actualizables y cuáles no. ¿Es necesario ante un cambio actualizar toda la pantalla? Esta pregunta conlleva a un análisis sobre los widgets que son complejos y se reconstruyen constantemente.



En dispositivos reales, la aplicación funciona bastante bien y es responsiva, esto es gracias al framework Flutter, que fue creado con el propósito de ser multiplataforma y que las aplicaciones sean responsivas. Sin embargo, es importante considerar el poder de cómputo de los smartphones más antiguos, ya que se desconoce qué clase de dispositivos usarán los usuarios finales.

8. Conclusiones

Este desarrollo tomó un tiempo total (aproximado) de 3 meses, esto es, considerando el desarrollo del Software como tal y además el trabajo de redacción de este documento. Esto consideró también el tiempo para realizar investigación adicional, es decir, lectura de documentación. Lo que permitió el aprendizaje de nuevos conceptos y de diversas técnicas para aplicar en el mundo del desarrollo de software. El desarrollo de esta aplicación no contó con la asesoría de ningún profesional del área. Mi formación profesional adquirida en el transcurso de la carrera de Ingeniería Informática me permitió afrontar este desafío. La Aplicación NeoRouteApp, actualmente no está siendo utilizada, las razones se deben a que se requiere un presupuesto para llevarlo a cabo, el cual no se llegó a un acuerdo. Ahora, el futuro de esta aplicación móvil va de la mano con las mejoras a realizar, en primer lugar, se debe mejorar el desempeño general, aplicando conceptos como la utilización de **Isolates** en Dart que permite la ejecución de tareas en paralelo, mejorando el desempeño. Así también mejoras en la utilización de los **gestores de estado**. Estas mejoras requieren un análisis y prácticamente reconstruir la Aplicación. Con estas mejoras la aplicación puede ser proyectada a utilizarse como una excelente propuesta para mi portafolio de desarrollador.

Referencias

- [1] R. Chávez, M. Pérez, S. Fuentes y G. Castro, «Mega-Incendio Valparaíso Febrero 2024,» 7 febrero 2024. [En línea]. Available: https://www.pucv.cl/uuaa/site/docs/20240209/20240209151003/incendio_valpo_2024_v01.pdf.
- [2] L. Williamson, «Fundamentos del desarrollo móvil,» 2012. [En línea]. Available: <https://ftpmirror.your.org/pub/misc/ftp.software.ibm.com/la/documents/gb/commons/RAW14302-ESES-00.pdf>.
- [3] L. Vespa, «Patrones Arquitectónicos en Android,» Mayo 2019. [En línea]. Available: <https://medium.com/@vespasoft/patrones-arquitect%C3%B3nicos-en-android-ded39f7a2c10>.
- [4] C. Toombs, «Meet ART,» 6 Nov 2013. [En línea]. Available: <https://www.androidpolice.com/2013/11/06/meet-art-part-1-the-new-super-fast-android-runtime-google-has-been-working-on-in-secret-for-over-2-years-debuts-in-kitkat/>.
- [5] P. Márquez, «Lenguajes de programación: qué son y para qué se utilizan,» Junio 2024. [En línea]. Available: <https://areaf5.es/blog/lenguajes-de-programacion-que-son-y-para-que-se-utilizan>.
- [6] Google, «Lenguaje de programación Dart,» [En línea]. Available: <https://dart-es.com/>.
- [7] AWS, «¿What is a framework?,» 2024. [En línea]. Available: <https://aws.amazon.com/what-is/framework/>.
- [8] Microsoft, «Introducción a la creación de una aplicación de escritorio con React Native para Windows,» 2024. [En línea]. Available: <https://learn.microsoft.com/es-es/windows/dev-environment/javascript/react-native-for-windows>.



- [9] AWS, «¿Qué es Flutter?,» 2024. [En línea]. Available: <https://aws.amazon.com/es/what-is/flutter/>.
- [10] Google, «Flutter architectural overview,» 2024. [En línea]. Available: <https://docs.flutter.dev/resources/architectural-overview>.
- [11] AWS, «¿Qué es una interfaz de programación de aplicaciones (API)?,» 2024. [En línea]. Available: <https://aws.amazon.com/es/what-is/api/>.
- [12] K. Schwaber y J. Sutherland, «Guía de Scrum 2020,» Noviembre 2020. [En línea]. Available: https://objetivoscrum.com/wp-content/uploads/2021/01/2020-Scrum-Guide-Spanish-European-2.0_objetivoScrum.pdf.
- [13] Refactoring.guru, «Facade Pattern,» 2024. [En línea]. Available: <https://refactoring.guru/es/design-patterns/facade>.
- [14] Refactoring.guru, «Factory Method Pattern,» 2024. [En línea]. Available: <https://refactoring.guru/es/design-patterns/factory-method>.
- [15] Flutter, «Flutter DevTools,» 2024. [En línea]. Available: <https://docs.flutter.dev/tools/devtools>.
- [16] Flutter, «Build Modes,» [En línea]. Available: <https://docs.flutter.dev/testing/build-modes>.
- [17] Flutter, «Use the performance view,» 2024. [En línea]. Available: <https://docs.flutter.dev/tools/devtools/performance>.
- [18] Flutter, «Preguntas frecuentes,» [En línea]. Available: <https://docs.flutter.dev/resources/faq>.