UNIVERSIDAD PRIVADA DE TACNA

FACULTAD DE INGENIERIA

Escuela Profesional de Ingeniería de Sistemas

Proyecto RideUPT

"Conecta tu camino"

Curso: *PATRONES DE SOFTWARE*

Docente: *Mag. Patrick Cuadros Quiroga*

Integrantes:

Jorge Luis BRICEÑO DIAZ (2017059611)
Mirian CUADROS GARCIA (2021071083)
Brayar LOPEZ CATUNTA (2020068946)
Ricardo DE LA CRUZ CHOQUE (2019063329)

Tacna – Perú
2025

CONTROL DE VERSIONES					
Versión	Hecha por	Revisada por	Aprobada por	Fecha	Motivo
1.0	JBD	MCG	JBD	22/10/2025	Versión Original
2.0	JBD	MCG	JBD	25/01/2025	Actualización con arquitectura implementada
3.0	MCG	BLC	JBD	25/01/2025	Actualización de arquitectura implementada

Sistema RideUPT-Conecta tu camino

Documento de Arquitectura de Software

Versión *{3.0}*

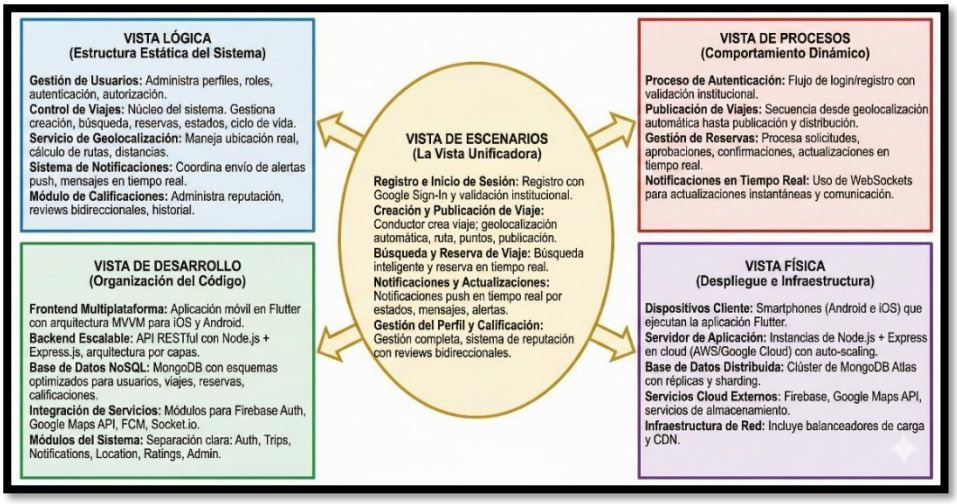
CONTROL DE VERSIONES					
Versión	Hecha por	Revisada por	Aprobada por	Fecha	Motivo
1.0	JBD	MCG	JBD	22/10/2025	Versión Original
2.0	JBD	MCG	JBD	25/01/2025	Actualización con arquitectura implementada
3.0	MCG	BLC	JBD	25/01/2025	Actualización de arquitectura implementada

Contenido

1. INTRODUCCIÓN

1.1. Propósito (Diagrama 4+1)

El presente documento describe de manera exhaustiva la arquitectura del sistema RideUPT, una plataforma innovadora de carpooling universitario desarrollada bajo el reconocido patrón arquitectónico 4+1 vistas. La arquitectura ha sido meticulosamente diseñada para satisfacer de manera integral los requerimientos funcionales de transporte compartido entre estudiantes, priorizando fundamentalmente tres pilares esenciales: la seguridad en las transacciones y datos, la escalabilidad para crecimiento futuro y la experiencia de usuario optimizada.



Entre las decisiones arquitectónicas más significativas se encuentra la selección de un frontend multiplataforma basado en Flutter, elección estratégica que maximiza el alcance potencial entre la comunidad estudiantil. Complementariamente, se optó por un backend basado en API REST construido con Node.js y Express, proporcionando la flexibilidad necesaria para adaptaciones futuras. La base de datos NoSQL con MongoDB responde a la necesidad de esquemas dinámicos que caracterizan las aplicaciones modernas, mientras que la comunicación en tiempo real se implementa mediante WebSockets para garantizar una experiencia fluida. Es importante destacar que, en la balanza de prioridades, se ha privilegiado la mantenibilidad y escalabilidad sobre optimizaciones extremas que pudieran comprometer la evolución futura del sistema.

1.2. Alcance

Este documento se centra específicamente en la vista lógica del framework RideUPT, detallando de manera minuciosa los subsistemas, componentes y sus interacciones fundamentales. Se incluyen aspectos esenciales de las vistas de implementación, procesos y despliegue que resultan relevantes para la comprensión global de la arquitectura. Sin embargo, se omiten deliberadamente aquellos detalles de bajo nivel que no impactan significativamente en la comprensión de la arquitectura general, manteniendo así el foco en los elementos estructurales que definen el sistema.

1.3. Definición, siglas y abreviaturas

Término	*Definición*
API	Application Programming Interface
FCM	Firebase Cloud Messaging
GPS	Global Positioning System
JWT	JSON Web Token
MVC	Model-View-Controller
MVP	Minimum Viable Product
NoSQL	Not Only SQL
REST	Representational State Transfer
UI/UX	User Interface/User Experience
UPT	Universidad Privada de Tacna

1.4. Organización del documento

Las referencias aplicables son:

- Documento de Visión de Proyecto.
- Documento de Factibilidad
- Documento de Especificación de Requerimientos de Software

2. OBJETIVOS Y RESTRICCIONES ARQUITECTONICAS

2.1. Priorización de requerimientos

Requerimientos funcionales — prioridad alta

ID	Descripción	Prioridad
RF001	El sistema debe permitir el registro e inicio de sesión de usuarios con credenciales válidas de estudiantes.	Alta
RF002	El sistema debe permitir la aceptación (habilitación) y edición de perfiles de conductor.	Alta
RF003	Los conductores deben poder crear viajes usando geolocalización automática para el origen.	Alta
RF004	Los pasajeros deben poder buscar viajes disponibles por origen, destino y hora.	Alta
RF008	El sistema debe permitir autenticación rápida y segura mediante cuentas de Google.	Alta

Requerimientos funcionales — prioridad media

ID	Descripción	Prioridad
RF005	El sistema debe enviar notificaciones en tiempo real ante cambios de estado relevantes.	Media
RF006	Los usuarios deben poder acceder a un historial de viajes pasados y próximos.	Media
RF007	Los viajes deben expirar automáticamente después de 10 minutos si no son tomados.	Media

Requerimientos no funcionales — prioridad alta

ID	Descripción	Prioridad
RNF001	El sistema debe garantizar la seguridad de las contraseñas mediante hash bcrypt antes de almacenarlas.	Alta

RNF002	El sistema debe implementar autenticación JWT con tokens que expiren en 30 días.	Alta
RNF003	El sistema debe validar que solo usuarios con email institucional (@upt.pe, @virtual.upt.pe) puedan registrarse.	Alta
RNF004	El sistema debe funcionar en múltiples plataformas: Android, iOS y Web.	Alta
RNF005	El sistema debe manejar errores de conexión y proporcionar mensajes claros al usuario.	Alta

2.1.1. Requerimientos Funcionales

Tabla 1: Cuadro de Requerimientos funcionales

ID	Nombre	Descripción	Prioridad
RF001	**Autenticar Usuario**	El sistema debe permitir el registro e inicio de sesión de usuarios con credenciales válidas de estudiantes.	Alta
RF002	**Gestionar Conductor**	El sistema debe permitir la aceptación (habilitación) y edición de perfiles de conductor.	Alta
RF003	**Crear Viaje**	Los conductores deben poder crear viajes usando geolocalización automática para el origen.	Alta
RF004	**Buscar Viaje**	Los pasajeros deben poder buscar viajes disponibles por origen, destino y hora.	Alta
RF005	**Enviar Notificación**	El sistema debe enviar notificaciones en tiempo real ante cambios de estado relevantes.	Media
RF006	**Consultar Historial**	Los usuarios deben poder acceder a un historial de viajes pasados y próximos.	Media

RF007	**Expirar Viaje**	Los viajes deben expirar automáticamente después de 10 minutos si no son tomados.	Media
RF008	**Autenticar con Google**	El sistema debe permitir autenticación rápida y segura mediante cuentas de Google.	Alta

Fuente: Elaboración propia del equipo de trabajo

En la Tabla 1 se detalla los Requerimientos Funcionales (RF) que definen las capacidades y comportamientos específicos del sistema RideUPT.

2.1.2. Requerimientos No Funcionales – Atributos de Calidad

Tabla 2: Cuadro de Requerimientos No funcionales

ID	Nombre	Descripción	Prioridad
RNF001	**Usabilidad**	La interfaz debe ser intuitiva, permitiendo que un usuario nuevo aprenda a usarla en < 3 minutos.	Alta
RNF002	**Rendimiento**	El tiempo de respuesta para operaciones principales (login, búsqueda, creación de viaje) debe ser < 2 segundos.	Alta
RNF003	**Disponibilidad**	El sistema debe aspirar a un uptime del 99.5% mensual.	Alta
RNF004	**Seguridad**	La información sensible debe ser protegida mediante cifrado y el sistema debe usar autenticación basada en tokens (JWT).	Alta
RNF005	**Escalabilidad**	La arquitectura debe permitir el crecimiento progresivo de usuarios sin degradar significativamente el rendimiento.	Media

Fuente: Elaboración propia del equipo de trabajo

En la Tabla N.º 2 tenemos la tabla de requerimientos no funcionales donde se toman atributos de calidad que es justificada con los requerimientos no funcionales.
Finalmente, se le asigna un código de identificación a cada requerimiento.

3. REPRESENTACIÓN DE LA ARQUITECTURA DEL SISTEMA

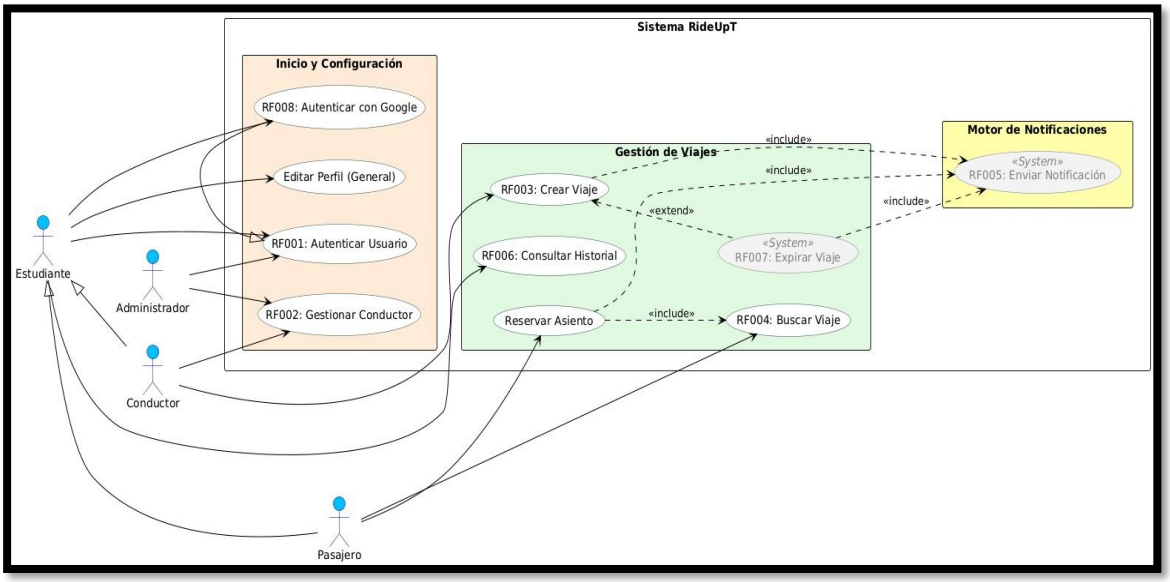
3.1. Vista de Caso de uso

En esta sección se muestran los Casos de Uso relevantes para la arquitectura, así como también a los principales Actores, su desarrollo del presente software implica que la arquitectura sea adecuada para poder suministrar esa funcionalidad.

3.1.1. Diagramas de Casos de uso

En este apartado se presenta el Diagrama de Casos de Uso del sistema, el cual describe las principales interacciones entre los actores (Conductor y Pasajero) y las funcionalidades del sistema.

Gráfico 01: Diagrama de Caso de Usos

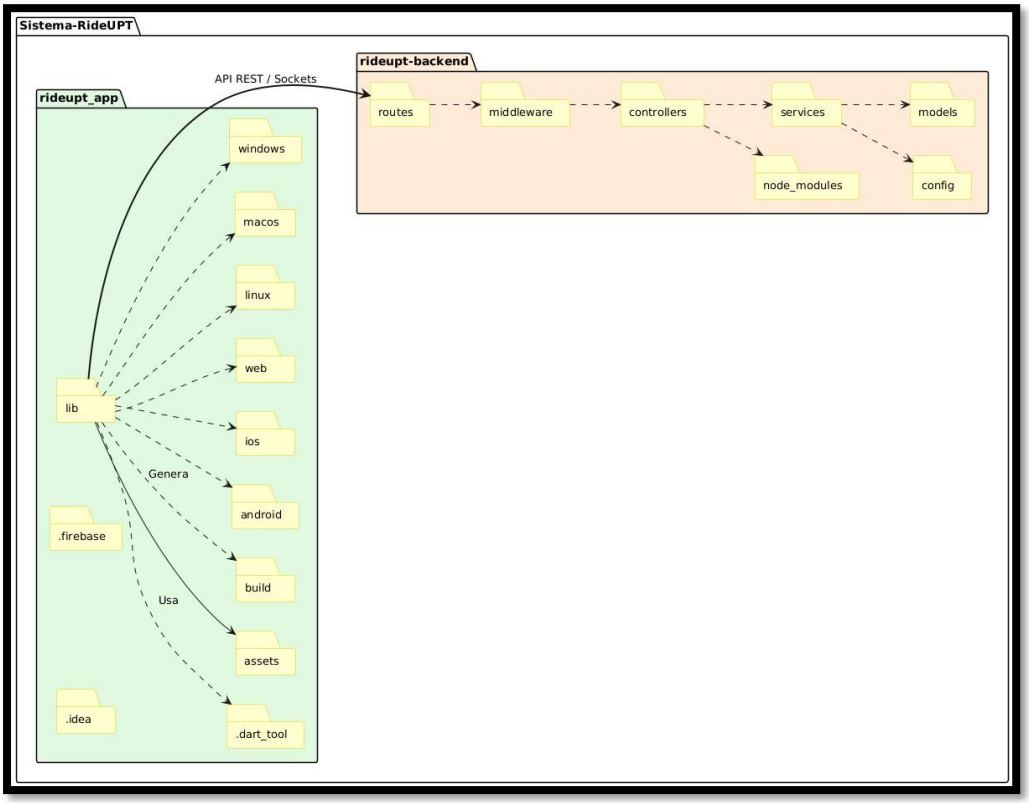


Fuente: Elaboración propia del equipo de trabajo

En el gráfico 01: El diagrama ilustra un sistema de viajes compartidos donde Pasajeros buscan transporte y Conductores publican rutas. Ambos acceden mediante credenciales o Google. El sistema automatiza procesos clave: envía notificaciones en tiempo real y expira viajes inactivos tras 10 minutos, garantizando eficiencia y seguridad en la conexión entre usuarios.

3.2. Vista Lógica

3.2.1. Diagrama de Subsistemas (paquetes)



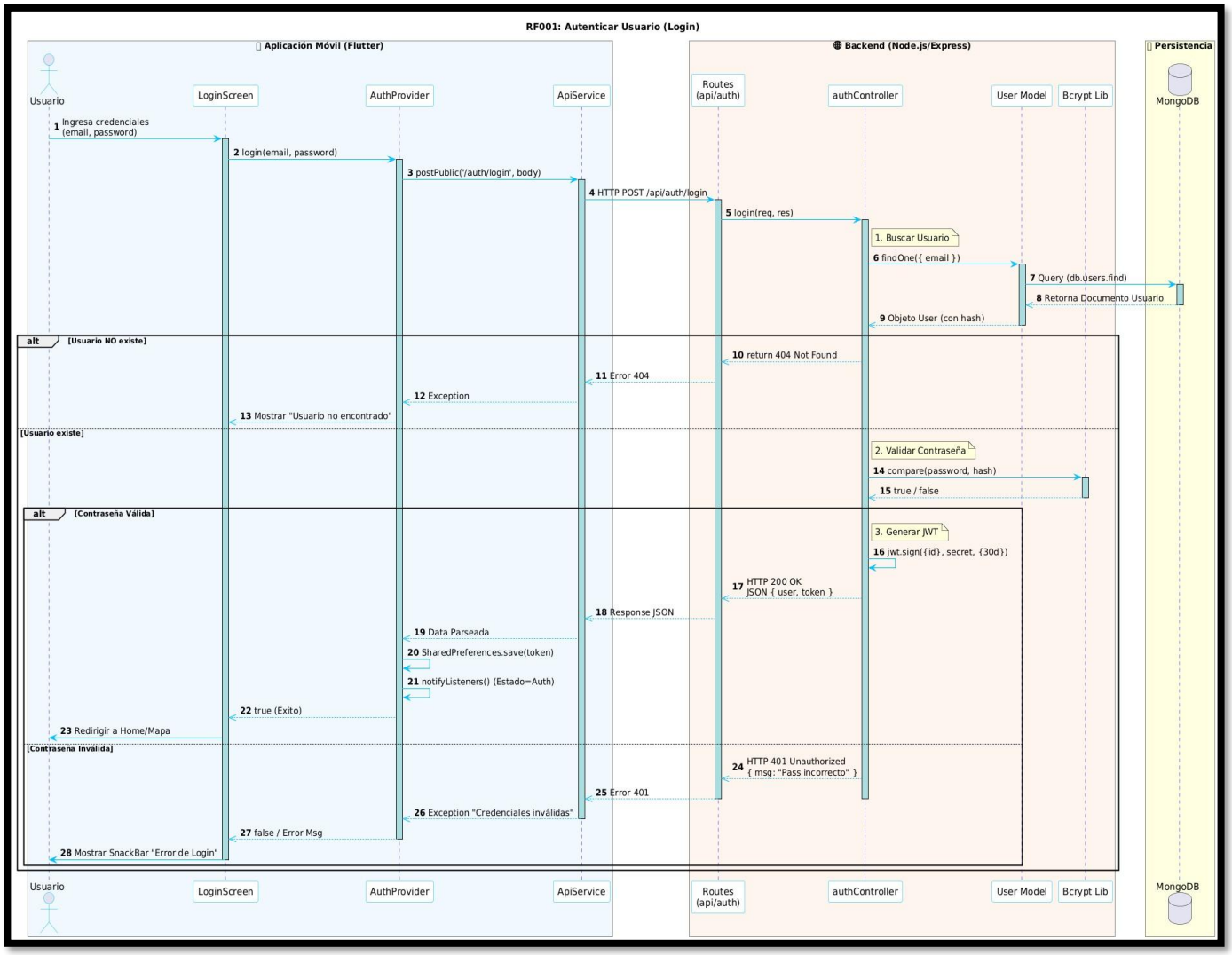
El Sistema RideUPT es una plataforma de transporte que funciona bajo una arquitectura cliente-servidor compuesta por dos elementos principales.
El frontend (Fideupt_app) es una aplicación multiplataforma desarrollada en Flutter que funciona en Windows, macOS, Linux y web, integrando tanto API REST como WebSockets para comunicación en tiempo real. Utiliza Firebase para servicios backend móviles y gestiona dependencias mediante una estructura modular.

El backend (Fideupt-backend) está construido en Node.js con una arquitectura por capas que incluye routes, middleware, controllers, services y models, permitiendo un procesamiento escalable de peticiones.

El sistema permite comunicación bidireccional en tiempo real, es completamente multiplataforma y sigue principios de desarrollo modular para facilitar el mantenimiento y escalabilidad.

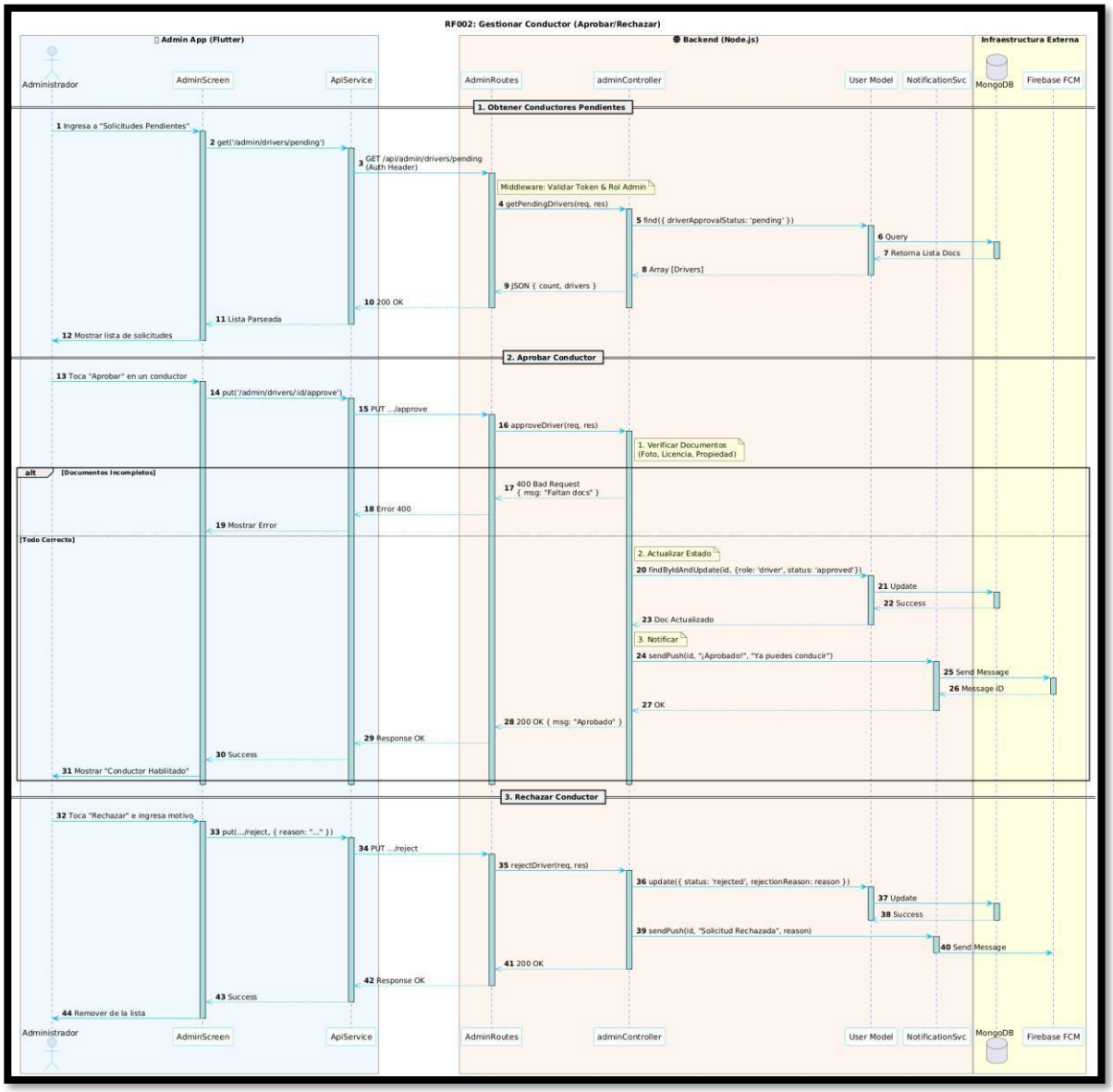
3.2.2. Diagrama de Secuencia (vista de diseño)

RF001: Autenticar Usuario



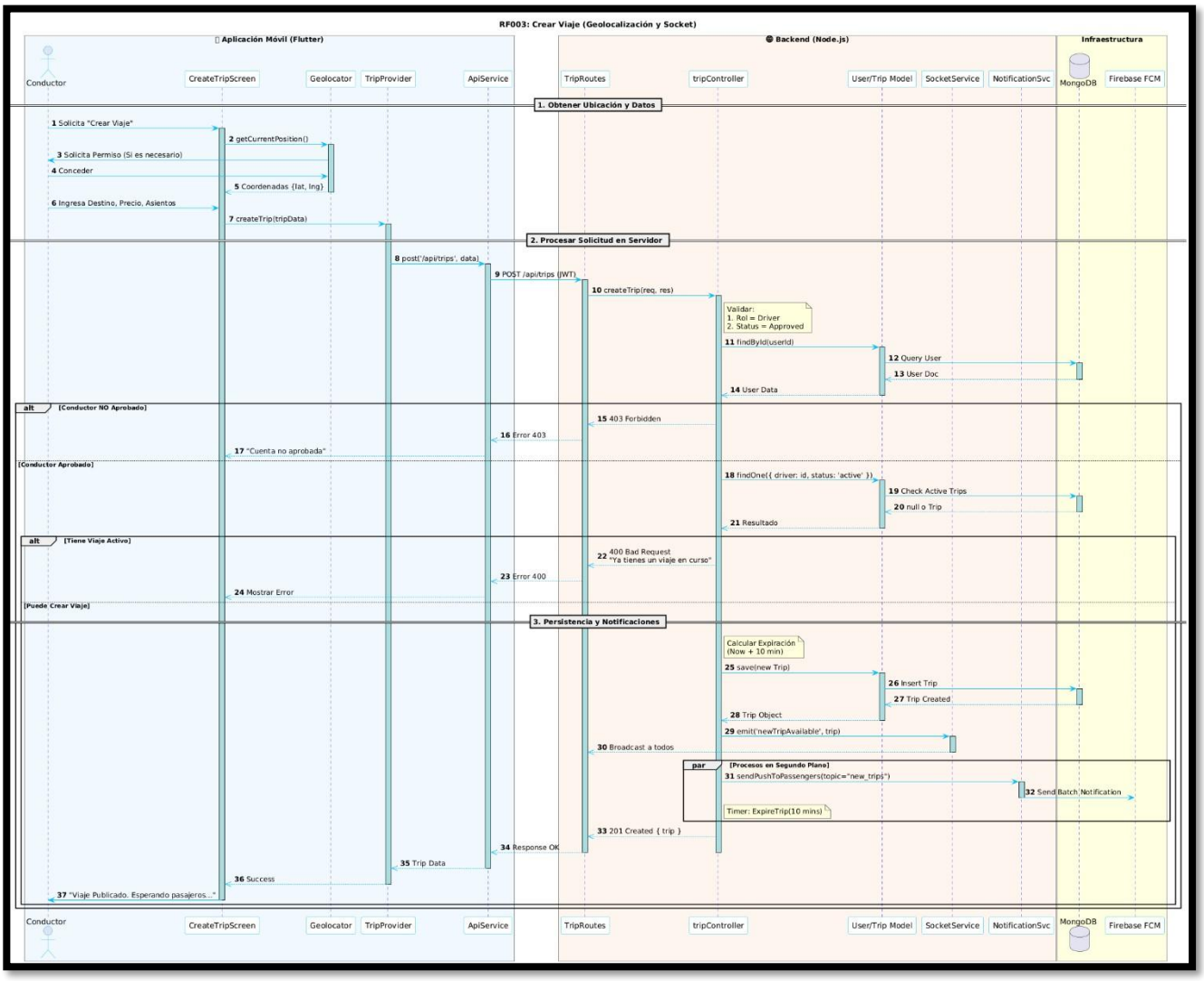
RF001: Autenticar Usuario (Login) Describe el flujo de seguridad donde la aplicación envía las credenciales (email/password) al backend. El controlador busca al usuario en MongoDB y utiliza bcrypt para validar la contraseña encriptada. Si la validación es exitosa, el servidor firma y retorna un Token JWT que la aplicación almacena localmente para autorizar futuras peticiones.

RF002: Gestionar Conductor



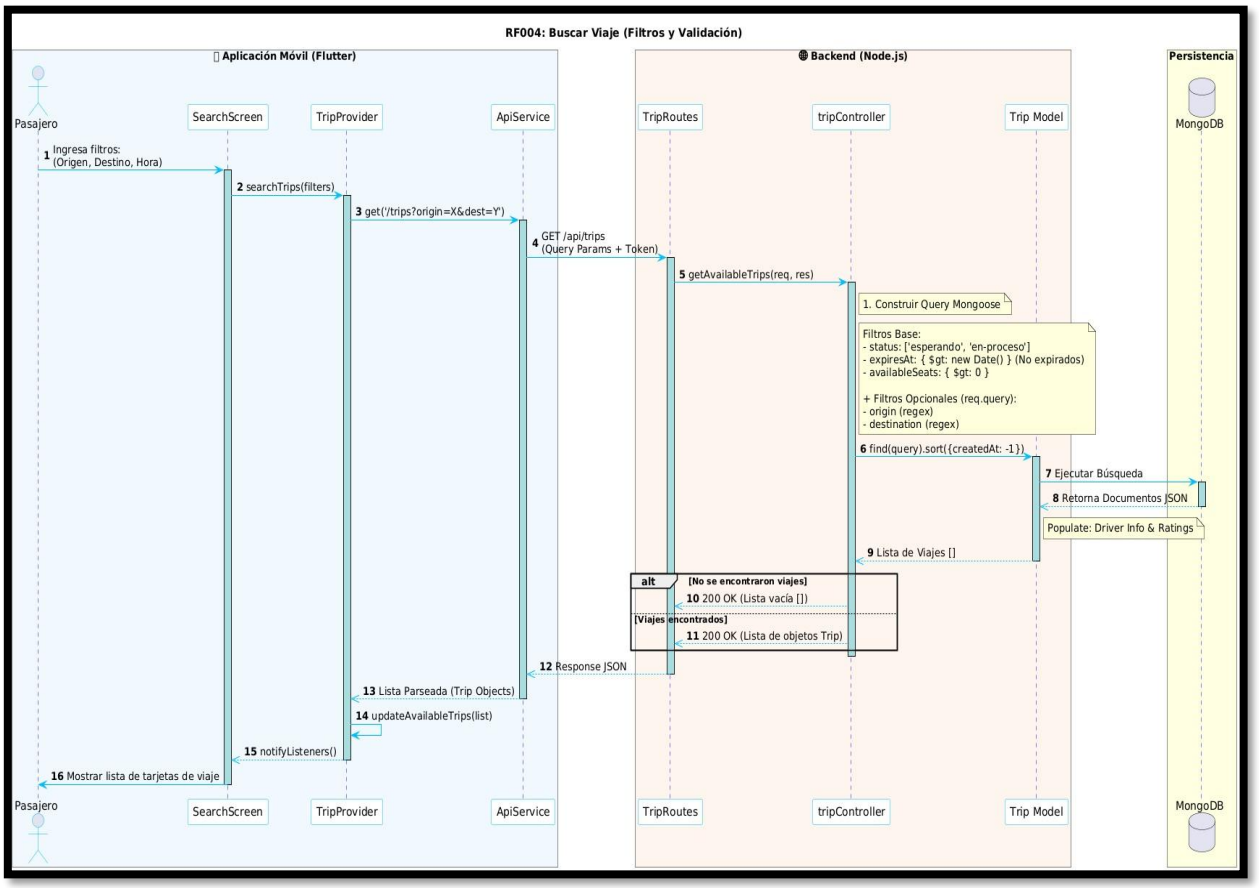
RF002: Gestionar Conductor (Admin) Detalla cómo el administrador revisa solicitudes pendientes. El backend valida los documentos requeridos y actualiza el estado del conductor (aprobado o rechazado) en la base de datos. Crucialmente, el sistema integra NotificationService para enviar una alerta Push vía Firebase (FCM) al dispositivo del conductor, informándole inmediatamente sobre el cambio de su estatus.

RF003: Crear Viaje



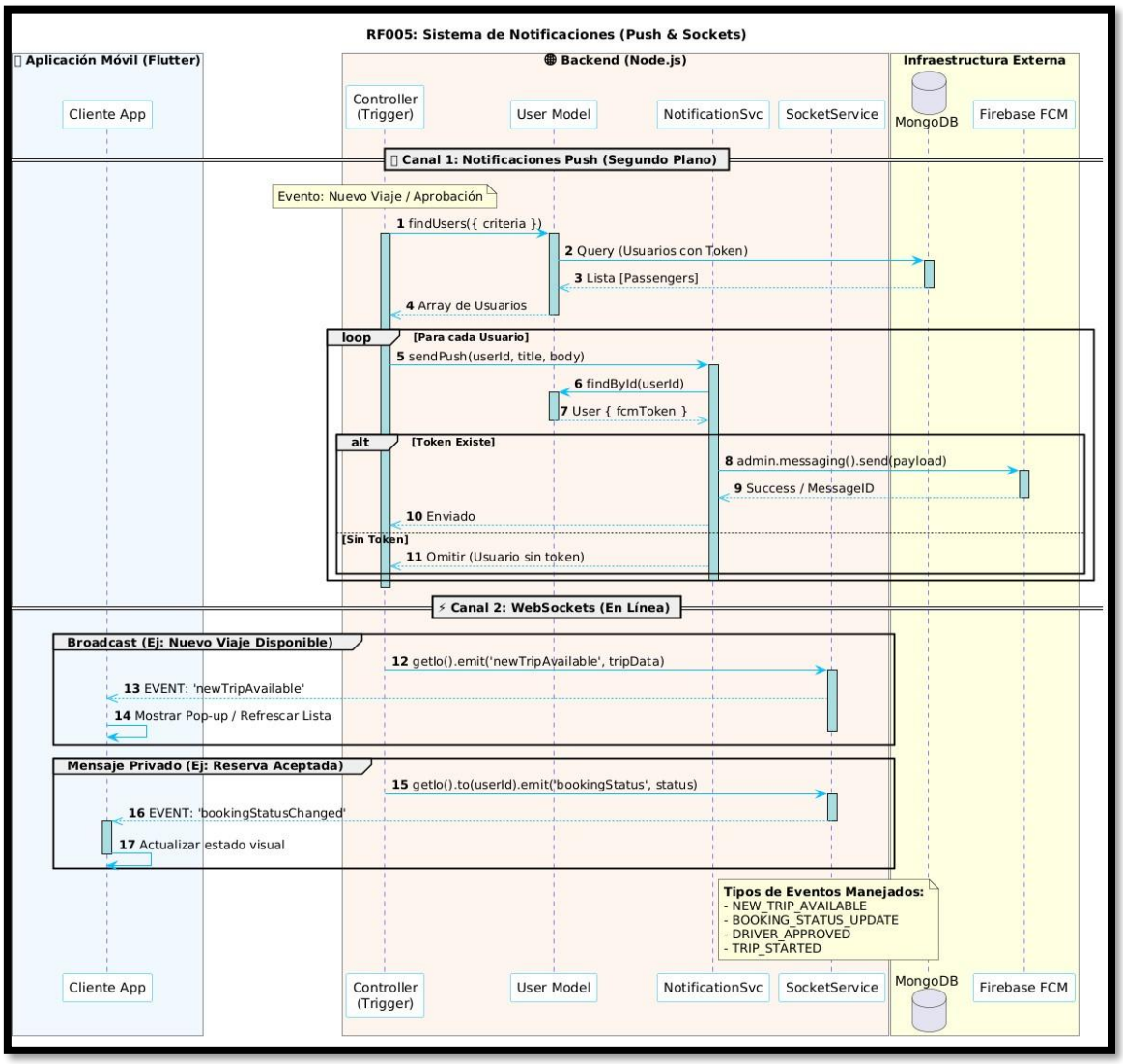
RF003: Crear Viaje (Conductor) Comienza con la obtención de coordenadas GPS en el dispositivo. El servidor valida que el conductor esté habilitado y sin viajes activos antes de guardar en MongoDB. Destaca por su concurrencia: al guardar, emite un evento Socket.io para visualización en tiempo real y dispara notificaciones Push en segundo plano para alertar a los pasajeros.

RF004: Buscar Viaje



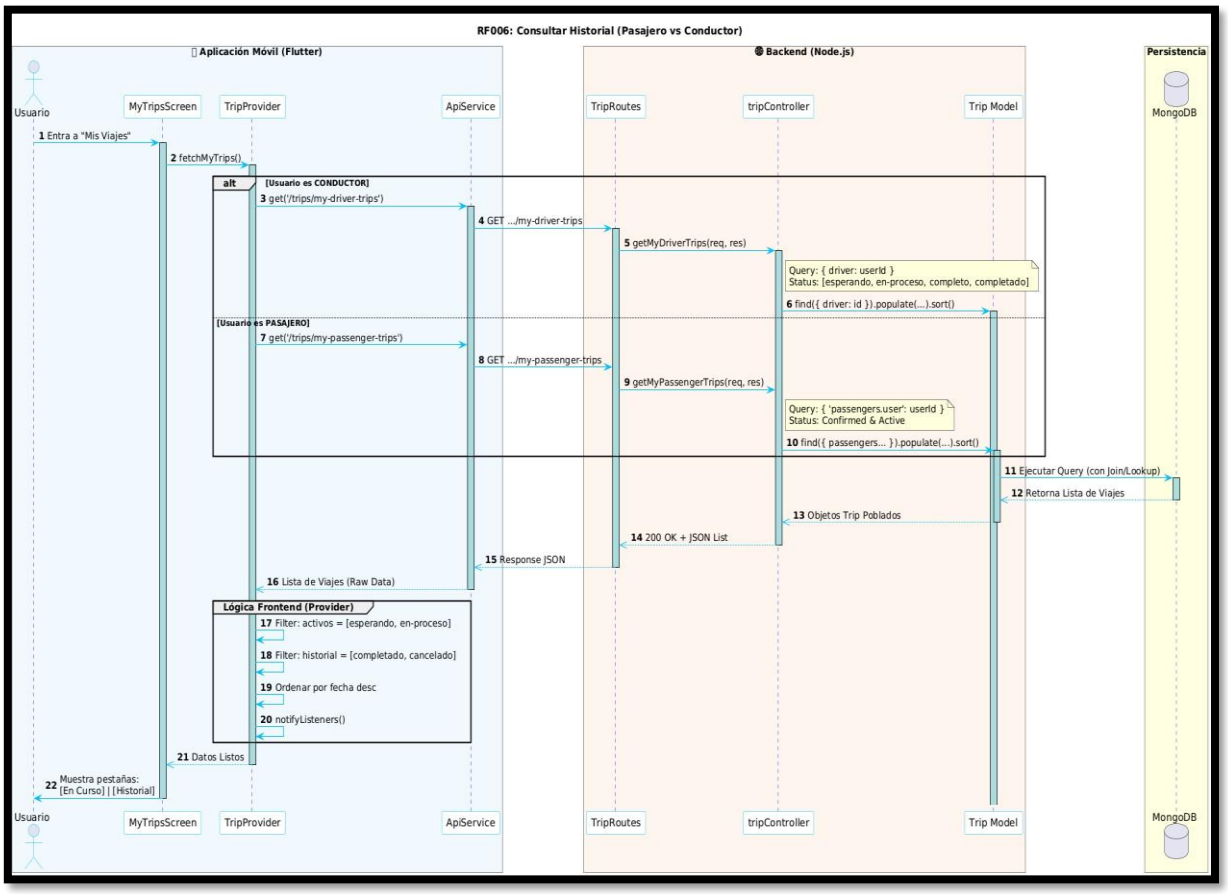
RF004: Buscar Viaje (Pasajero) Ilustra el proceso de filtrado de viajes. El controlador recibe parámetros (origen, destino) y ejecuta una consulta optimizada en MongoDB. Aplica filtros de seguridad (viajes no expirados, con asientos disponibles) y utiliza populate para inyectar los datos del perfil del conductor en la respuesta, entregando al pasajero una lista lista para visualizar.

RF005: Enviar Notificación



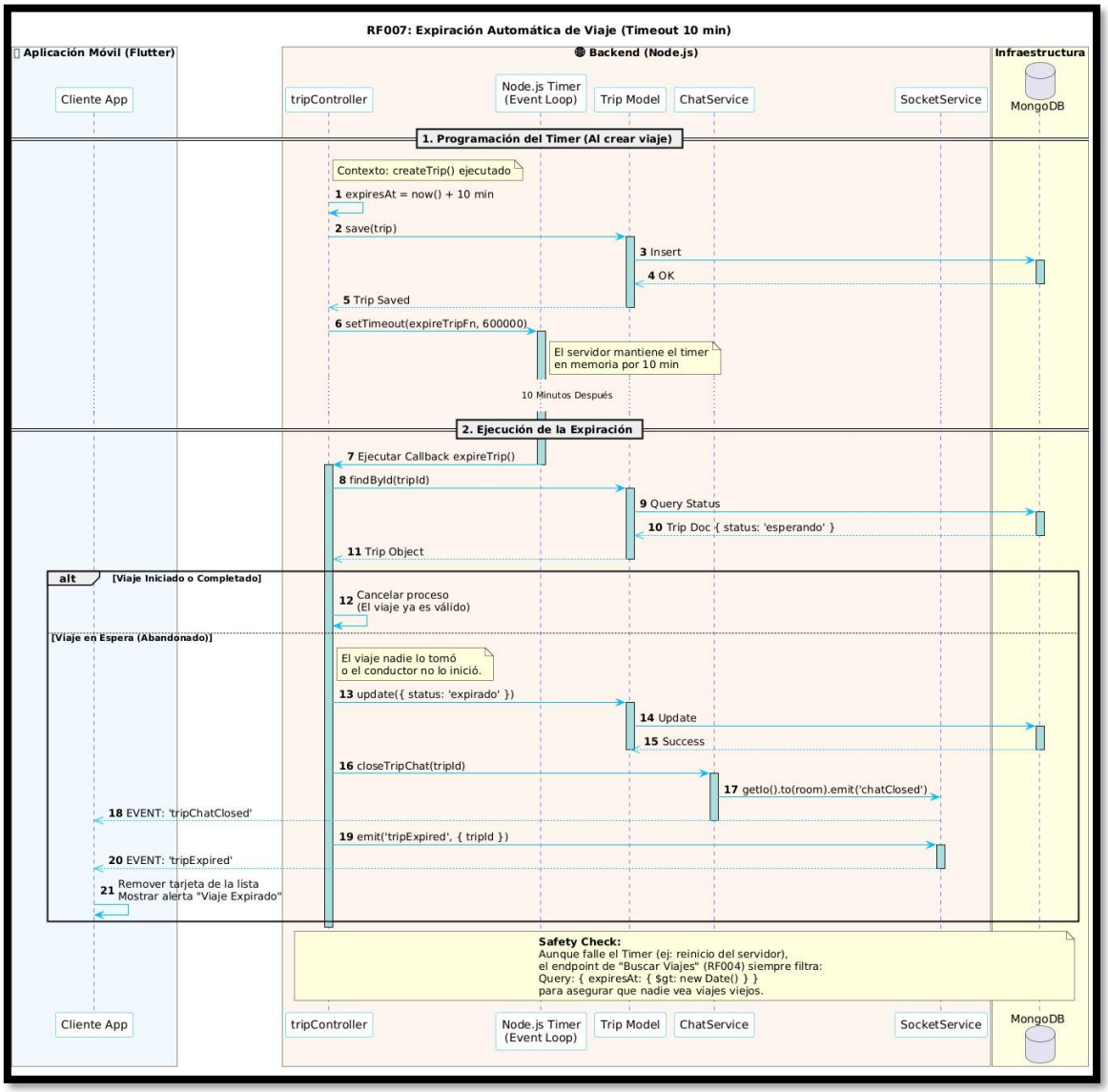
RF005: Enviar Notificación (Sistema) Define la arquitectura híbrida de comunicaciones. Diferencia claramente dos canales: Firebase FCM para notificaciones Push (cuando la app está cerrada) y WebSockets para eventos en tiempo real (cuando la app está abierta). Maneja tanto mensajes de difusión (Broadcast) para nuevos viajes, como mensajes privados para cambios de estado en reservas específicas.

RF006: Consultar Historial



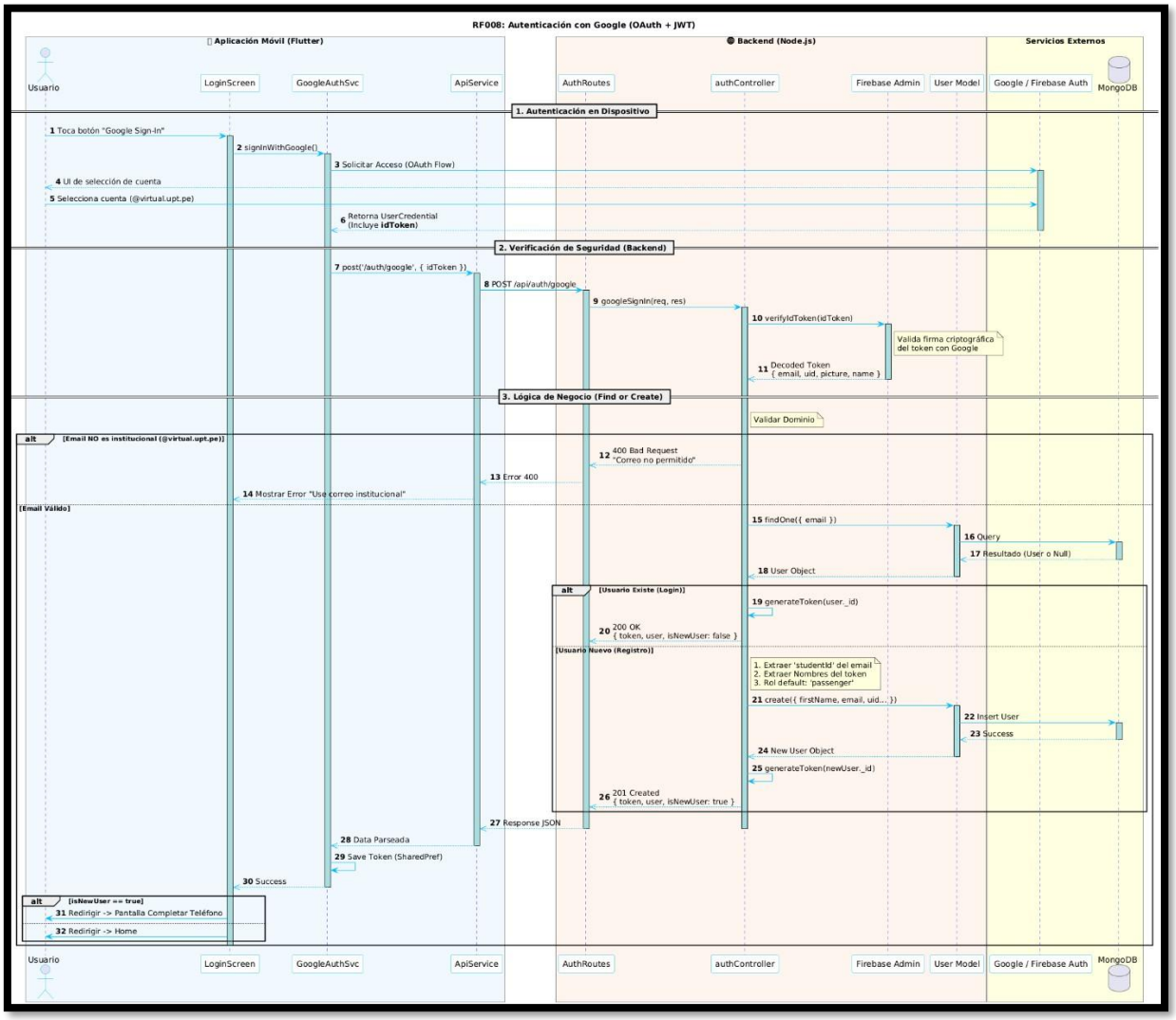
RF006: Consultar Historial (Usuario) Muestra una lógica condicional basada en el rol. El backend ejecuta consultas distintas dependiendo de si el usuario solicita datos como conductor (viajes creados) o pasajero (viajes tomados). Devuelve la data cruda ordenada cronológicamente, dejando que el Frontend (Flutter) procese y separe visualmente los viajes en pestañas de "Activos" e "Historial".

RF007: Expirar Viaje



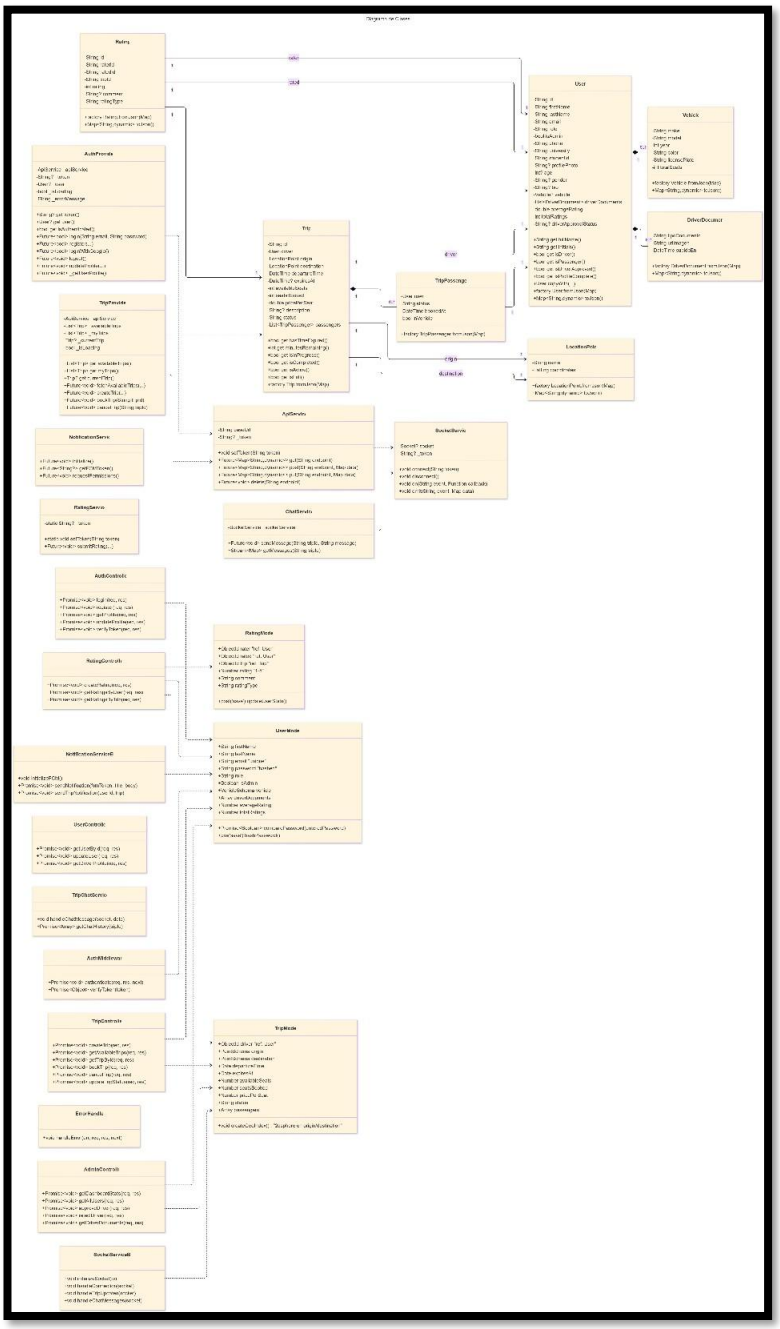
RF007: Expirar Viaje (Automático) Modela el ciclo de vida del viaje mediante un temporizador en el servidor. Si un viaje permanece en estado "esperando" por 10 minutos, el sistema ejecuta un callback que actualiza el estado a "expirado" en la BD, fuerza el cierre de la sala de chat asociada y emite un evento Socket para remover el viaje de las búsquedas.

RF008: Autenticar con Google



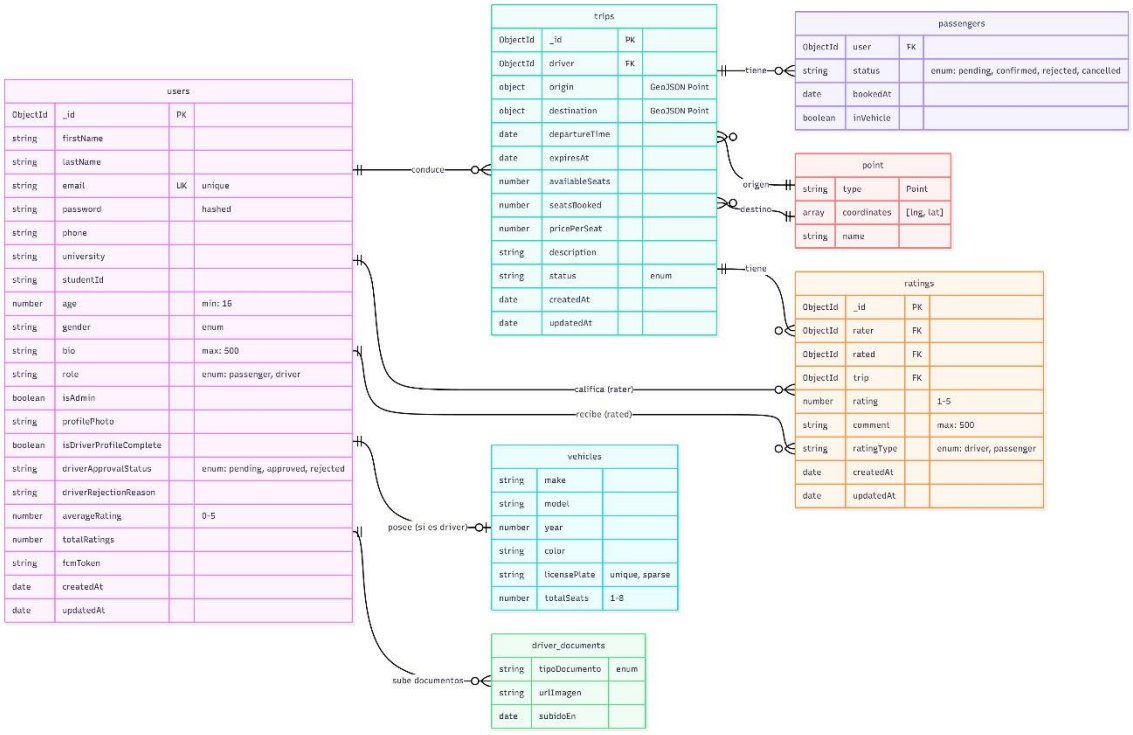
RF008: Autenticar con Google (OAuth) Presenta un flujo de autenticación federada seguro. La app obtiene un token de Google, pero el backend lo verifica rigurosamente usando Firebase Admin SDK. Implementa la regla de negocio del correo institucional (@virtual.upt.pe) y una lógica inteligente "Find or Create", que registra automáticamente al usuario extrayendo sus datos si es su primer acceso.

3.2.3. Diagrama de Clases

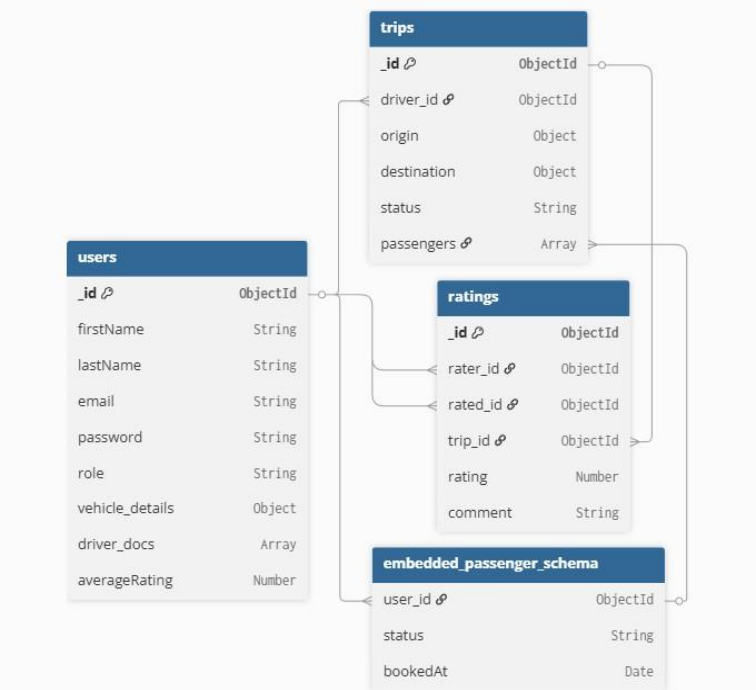


*Este Diagrama de Clases modela la arquitectura integral del sistema **RideUpT**. Define las entidades de datos principales (**User**, **Trip**, **Vehicle**, **Rating**) y sus relaciones. Estructura la lógica separando los componentes del Cliente (**Providers**, **Services**) de los del Backend (**Controllers**, **Models**), organizando modularmente la autenticación, la gestión de viajes en tiempo real y el manejo de datos.*

3.2.4. Diagrama de Base de datos (relacional o no relacional)



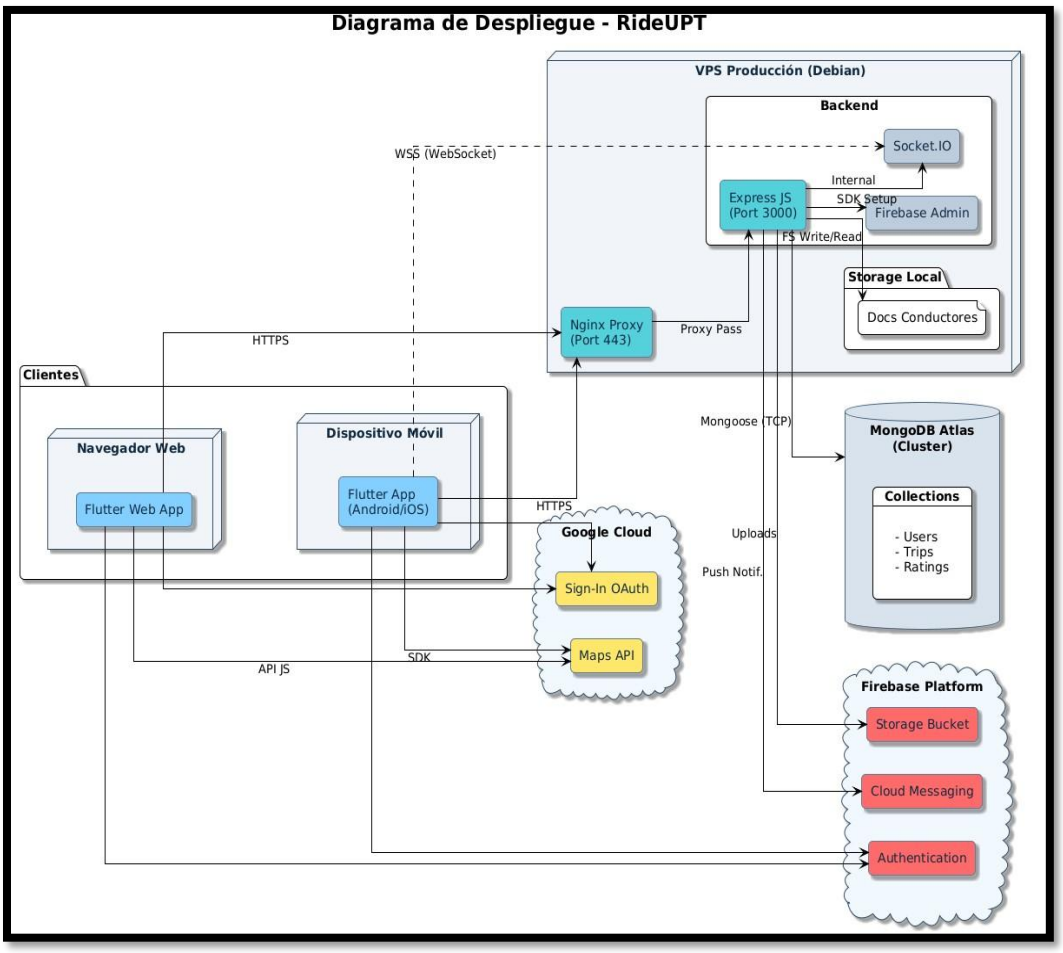
Este Diagrama Entidad-Relación (ERD) define el esquema de datos del sistema. Centrado en la colección Users, gestiona roles, Vehículos y Documentos de validación. La entidad Trips controla la logística operativa utilizando coordenadas GeoJSON y listas de Pasajeros, mientras que la colección Ratings permite calificaciones cruzadas entre usuarios, asegurando la reputación y confianza dentro de la plataforma.



3.3. Vista de Despliegue (vista física)

Se despliega uno o más escenarios de distribución física del sistema sobre los cuales se ejecutará y hará el despliegue del mismo. Muestra la comunicación entre los diferentes nodos que componen los escenarios antes mencionados, así como el mapeo de los elementos de la Vista de Procesos en dichos nodos

3.3.1. Diagrama de despliegue



4. Conclusiones

- Arquitectura Modular y Escalable El sistema implementa una arquitectura de software moderna y desacoplada, separando claramente el Frontend (Flutter) del Backend (Node.js/Express). La estructura en capas (Controladores, Servicios, Modelos) facilita el mantenimiento y la escalabilidad, permitiendo que la lógica de negocio, como la gestión de viajes y usuarios, evolucione independientemente de la interfaz de usuario.

- Seguridad y Confianza Institucional La seguridad es un pilar fundamental del diseño, garantizada mediante un doble mecanismo: la autenticación federada restringida al dominio institucional (@virtual.upt.pe) y un flujo de validación administrativa riguroso para los conductores. Esto asegura un entorno de confianza cerrado, donde la identidad de los usuarios y la legalidad de los vehículos están verificadas antes de permitir cualquier interacción operativa.

- Sincronización Híbrida en Tiempo Real La experiencia de usuario se optimiza mediante una estrategia de comunicación híbrida. El uso de WebSockets (Socket.io) permite actualizaciones instantáneas dentro de la aplicación (como el chat y cambios de estado de viaje), mientras que la integración con Firebase Cloud Messaging (FCM) asegura la entrega de notificaciones críticas en segundo plano, garantizando que conductores y pasajeros estén siempre coordinados.

- Automatización de la Lógica de Negocio El sistema reduce la fricción operativa mediante procesos automatizados inteligentes, como el algoritmo de expiración de viajes (10 minutos de inactividad) y el filtrado automático de rutas por geolocalización y tiempo. Esto asegura que la base de datos (MongoDB) se mantenga limpia con información relevante y vigente, mejorando la eficiencia en la conexión entre la oferta de transporte y la demanda estudiantil.