UNIVERSIDAD PRIVADA DE TACNA

FACULTAD DE INGENIERIA

Escuela Profesional de Ingeniería de Sistemas

Proyecto RideUPT – Conecta tu camino universitario

Curso: Patrones de Software

Docente: Mag. Ing. 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

CaelTek

UNIVERSIDAD PRIVADA DE TACNA
FACULTAD DE INGENIERÍA
Esc. Prof. de Ingeniería de Sistemas

Proyecto RideUPT

Documento de Especificación de Requerimientos de Software

Versión 1.0

CONTROL DE VERSIONES					
Versión	Hecha por	Revisada por	Aprobada por	Fecha	Motivo
1.0	RMDC	MCG	JBD	27/11/2025	Versión Original

ÍNDICE

I. Introducción

1.1. Generalidades de la Empresa

1. Nombre de la Empresa

CaelTek

CaelTek es una empresa peruana enfocada en el diseño y desarrollo de soluciones tecnológicas para la movilidad inteligente en el entorno universitario. Su producto principal es **RideUPT – Conecta tu camino universitario**, una plataforma de carpooling orientada a estudiantes que busca reducir costos de transporte, optimizar el uso de vehículos particulares y mejorar la accesibilidad al campus.

2. Visión

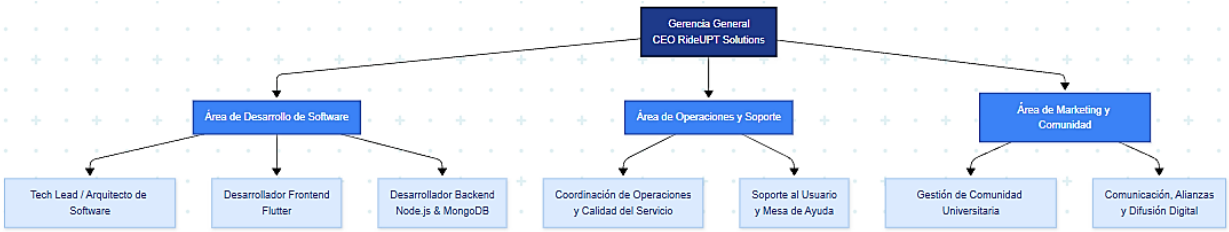
Ser la plataforma de referencia en el Perú para el transporte compartido universitario, ofreciendo una solución segura, económica y sostenible que conecte a estudiantes de distintas universidades y contribuya a reducir la congestión vehicular y el impacto ambiental asociado a la movilidad diaria.

3. Misión

El objetivo es crear, poner en marcha y mantener una aplicación móvil de carpooling pensada específicamente para estudiantes universitarios, que les permita reducir significativamente sus gastos de transporte—entre un 60 y 70%—mientras se garantiza su seguridad a través de validación institucional y funciones de control. Además, busca fortalecer la colaboración y el sentido de comunidad entre los estudiantes, utilizando tecnologías modernas como Flutter, Node.js, MongoDB, Firebase y Google Maps para asegurar un buen rendimiento, una plataforma escalable y una experiencia de uso intuitiva.

4. Organigrama

A continuación, se presenta el organigrama propuesto de CaelTek, modelado como una estructura ligera y enfocada en el desarrollo y operación de la plataforma RideUPT.



II. Visionamiento de la Empresa

2.1. Descripción del Problema

En el entorno universitario peruano, los estudiantes enfrentan diariamente **dificultades de transporte** que afectan directamente su rendimiento académico, puntualidad, seguridad y economía personal. Estas dificultades se manifiestan en:

- **Costos de transporte elevados**

Los estudiantes pueden gastar entre **S/. 8 y S/. 15 por día**, representando hasta un **30–40% de su presupuesto mensual**. Este impacto económico se agrava en aquellos que viven en distritos alejados o dependen de múltiples medios de transporte.

- **Insuficiencia de opciones accesibles**

El transporte público suele ser lento, impredecible y saturado, mientras que los servicios de taxi o movilidad por aplicaciones comerciales tienen tarifas que no son sostenibles para un estudiante promedio.

- **Problemas de congestión en el campus**

La cantidad creciente de vehículos particulares en los alrededores de la universidad genera congestión vehicular, conflictos con comercios cercanos y una necesidad creciente de espacios de estacionamiento que la institución no puede ampliar fácilmente.

- **Falta de alternativas seguras**

Los estudiantes no cuentan con una plataforma confiable para compartir viajes con personas verificadas. La falta de coordinación entre conductores y pasajeros dentro de la propia comunidad universitaria representa una oportunidad desaprovechada.

- **Impacto ambiental significativo**

La movilidad individual contribuye a mayores niveles de contaminación y a un consumo elevado de combustible, lo que se contrapone a los objetivos de sostenibilidad que muchas universidades están comenzando a promover.

Problema Central

No existe una plataforma institucional, segura y accesible que permita a los estudiantes universitarios compartir viajes de manera eficiente, reduciendo costos, congestionamiento vehicular y riesgos de seguridad.

2.2. Objetivos de Negocios

Los objetivos del negocio permiten alinear las metas de CaelTek con el impacto esperado en el mercado universitario. Entre los principales objetivos se encuentran:

Objetivo 1: Reducir los costos de movilidad estudiantil

Ofrecer una alternativa económica que permita un **ahorro mensual estimado entre S/. 150 y S/. 200 por estudiante**, promoviendo la equidad y accesibilidad a la educación.

Objetivo 2: Optimizar el uso de vehículos particulares

Fomentar que los estudiantes con vehículo compartan sus rutas, disminuyendo la saturación en las zonas cercanas al campus y reduciendo la demanda de estacionamientos.

Objetivo 3: Potenciar la seguridad en los desplazamientos

Garantizar que únicamente estudiantes verificados participen dentro de la plataforma mediante **autenticación institucional** y manejo responsable de datos personales.

Objetivo 4: Crear comunidad y promover interacción estudiantil

RideUPT también busca fortalecer el sentido de comunidad, permitiendo que los estudiantes interactúen de manera colaborativa mientras se apoyan mutuamente en sus desplazamientos diarios.

Objetivo 5: Establecer un modelo sostenible y escalable

Desarrollar una plataforma tecnológica robusta que pueda evolucionar e integrar nuevas universidades, funcionalidades adicionales y servicios complementarios.

2.3. Objetivos de Diseño

Los objetivos del diseño garantizan que la experiencia de usuario y la arquitectura del sistema sean consistentes, intuitivas y alineadas con las necesidades del mercado objetivo.

• Simplicidad en la experiencia de usuario (UX)

La aplicación debe permitir realizar tareas clave —crear un viaje, buscar un viaje, reservar un asiento— en **el menor número de pasos posibles**, evitando complejidad innecesaria.

• Diseño visual moderno y coherente

Basado en **Material Design 3**, utilizando colores institucionales, tipografías legibles y componentes visuales consistentes.

• Alto rendimiento y fluidez en dispositivos móviles

El sistema debe ser rápido, ligero y estable, incluso en dispositivos de gama media, garantizando tiempos mínimos de carga.

• Arquitectura escalable y mantenible

El backend en Node.js y la base de datos MongoDB deben permitir crecimiento a futuro, mientras que Flutter debe facilitar la incorporación de nuevas pantallas y componentes.

• Seguridad y privacidad como eje central

Los datos personales, ubicación y rutas deben protegerse siguiendo mejores prácticas, cifrado y control de acceso.

2.4. Alcance del Proyecto

El alcance del sistema RideUPT incluye:

- **Incluye (IN Scope)**
- Registro con cuenta institucional (Google Sign-In).
- Validación y creación de perfil del estudiante.
- Gestión de viajes: crear, buscar, filtrar y visualizar detalles.
- Sistema de reservas entre estudiantes.
- Notificaciones en tiempo real (FCM + Socket.IO).
- Historial de viajes realizados y ofertados.
- Gestión básica de documentos de conductor (licencia, SOAT).
- Panel administrativo básico para monitoreo.
- Cálculo automático de precios según distancia.
- **No incluye (OUT of Scope)**
- Pasarelas de pago o cobros automáticos.
- Integración con transporte público o servicios externos.
- Geofencing avanzado o rutas optimizadas por IA.
- Módulos de publicidad, gamificación o rankings (planes futuros).

Este alcance está en línea con los requerimientos académicos, técnicos y operativos establecidos para RideUPT.

2.5. Viabilidad del Sistema

La viabilidad del sistema ha sido confirmada mediante el análisis técnico, económico, operativo, legal, social y ambiental desarrollado previamente en el **FD01 – Informe de Factibilidad**.

1. **Viabilidad Técnica**

Las tecnologías seleccionadas (Flutter, Node.js, MongoDB, Firebase, Google Maps) son estándar en la industria, escalables y ampliamente soportadas. El sistema ya se encuentra implementado y funcional.

2. Viabilidad Económica

El costo total del proyecto asciende a **S/. 68,210**, con un retorno estimado menor a 12 meses y un VAN positivo.

3. Viabilidad Operativa

Los procesos de RideUPT están automatizados (notificaciones, expiración de viajes, cálculo de precios), por lo que el sistema requiere recursos mínimos para operar.

4. Viabilidad Legal

RideUPT cumple con la Ley 29733 y se encuentra categorizado como transporte privado compartido entre estudiantes, sin caer en regulación de transporte público.

5. Viabilidad Social y Ambiental

El proyecto tiene impacto positivo en la comunidad universitaria y contribuye a la reducción de emisiones y congestión vehicular.

- Técnica: viable con ASP.NET MVC 5, IIS y SQL Server 2016+; el equipo supera los mínimos de hardware/software.

- Operativa: alinea perfiles (Dev, DevOps, PM/SM, QA) y estandariza el flujo; uso vía navegador moderno.

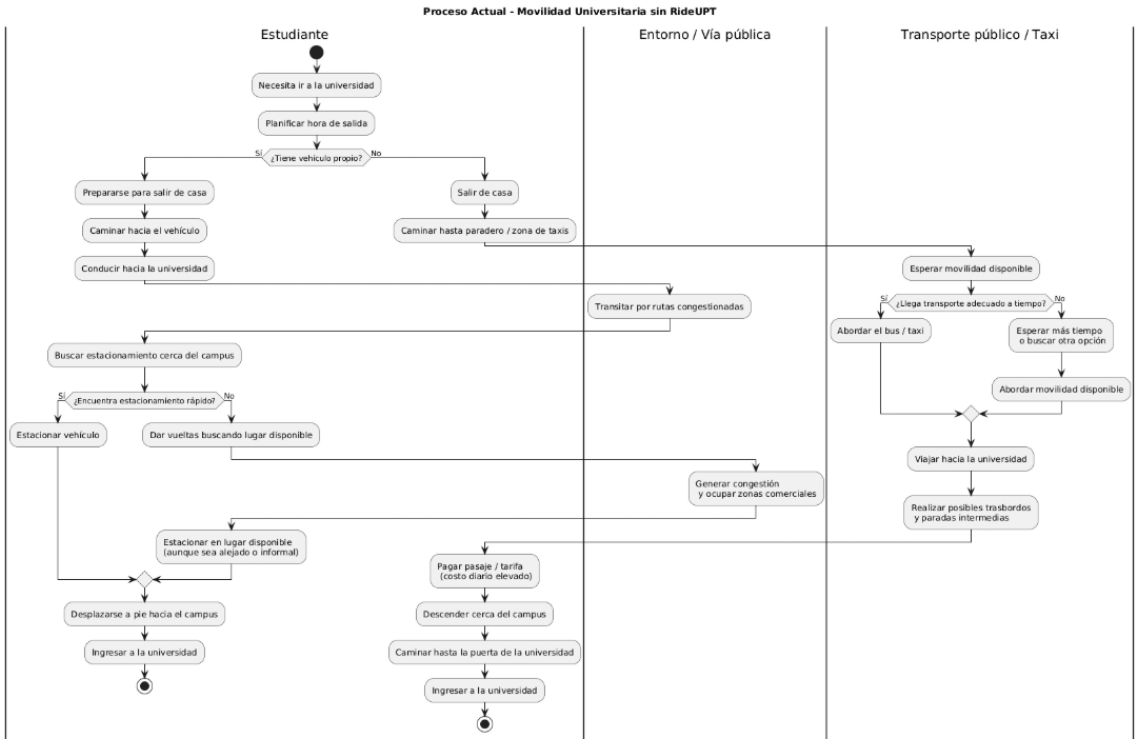
- Económica: costos totales estimados S/ 14,784.00; análisis previo con B/C ≈ 1.05, VAN positivo y TIR ≈ 16%.

- Legal: cumplimiento de protección de datos personales y propiedad intelectual, con licencias válidas para los componentes que lo requieran.

- Calidad/Servicio: el documento de visión define uptime > 95% como referencia de fiabilidad.

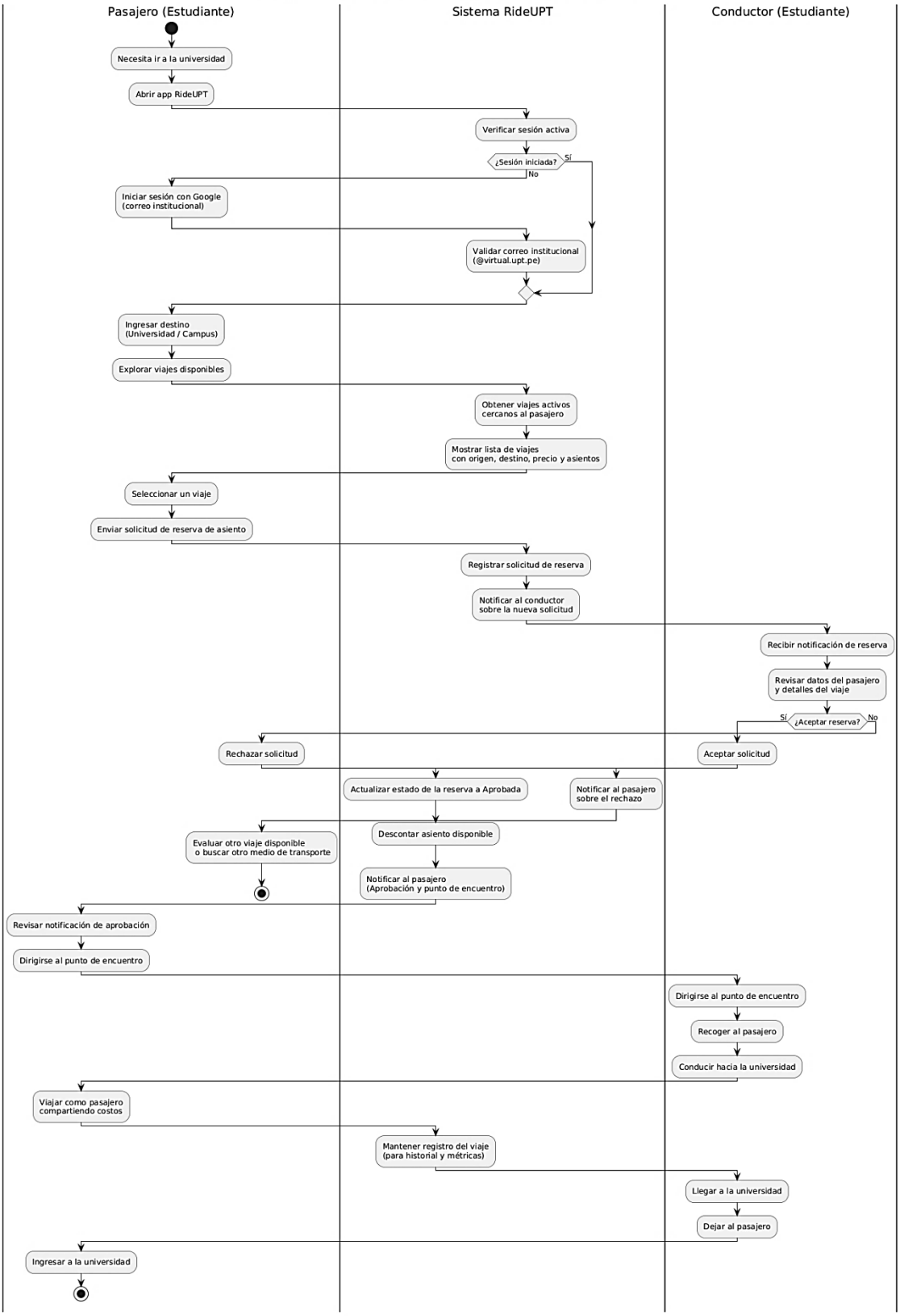
III. Análisis de Procesos

a) Diagrama del Proceso Actual – Diagrama de Actividades



b) Diagrama del Proceso Propuesto – Diagrama de Actividades Inicial

Proceso Propuesto - Movilidad Universitaria con RideUPT (Versión Inicial)



IV. Especificación de Requerimientos de Software

a) Cuadro de Requerimientos Funcionales Inicial

ID	Nombre	Descripción	Prioridad
RF001	Autenticar Usuarios	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 Viajes	Los conductores deben poder crear viajes usando geolocalización automática para el origen.	Alta
RF004	Buscar Viajes	Los pasajeros deben poder buscar viajes disponibles por origen, destino y hora.	Alta
RF005	Notificar Push	El sistema debe enviar notificaciones en tiempo real ante cambios de estado relevantes.	Media
RF006	Historial de Viajes	Los usuarios deben poder acceder a un historial de viajes pasados y próximos.	Media
RF007	Expirar Viajes	Los viajes deben expirar automáticamente después de 10 minutos si no son tomados.	Media
RF008	Autenticar desde Google Sign-In	El sistema debe permitir autenticación rápida y segura mediante cuentas de Google.	Alta

b) 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

c) Cuadro de Requerimientos Funcionales Final

ID	Nombre	Descripción	Prioridad
RF001	Autenticar Usuarios	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 Viajes	Los conductores deben poder crear viajes usando geolocalización automática para el origen.	Alta
RF004	Buscar Viajes	Los pasajeros deben poder buscar viajes disponibles por origen, destino y hora.	Alta
RF005	Notificar Push	El sistema debe enviar notificaciones en tiempo real ante cambios de estado relevantes.	Media
RF006	Historial de Viajes	Los usuarios deben poder acceder a un historial de viajes pasados y próximos.	Media
RF007	Expirar Viajes	Los viajes deben expirar automáticamente después de 10 minutos si no son tomados.	Media
RF008	Autenticar desde Google Sign-In	El sistema debe permitir autenticación rápida y segura mediante cuentas de Google.	Alta

d) Reglas de Negocio

ID	Regla	Descripción
RN001	Validación Estudiantil	Solo se permiten registros e inicios de sesión de usuarios con correo institucional válido (ej. @virtual.upt.pe).
RN002	Roles de Usuario	Un usuario puede actuar como pasajero, conductor o ambos, según la configuración de su perfil.
RN003	Precios de Viajes	El precio por asiento en un viaje debe encontrarse dentro del rango definido (por ejemplo, entre S/. 1.00 y S/. 3.00).
RN004	Expiración de Viajes	Todo viaje debe expirar automáticamente 10 minutos después de creado, si no se ha concretado o si no cumple condiciones para mantenerse activo.
RN005	Asientos Disponibles	El número máximo de asientos disponibles por viaje no puede exceder la capacidad del vehículo (por ejemplo, máximo 6 asientos).
RN006	Viajes Simultáneos	Un conductor solo puede tener un viaje activo a la vez; no puede publicar nuevos viajes mientras uno siga activo.
RN007	Reservas Múltiples	Un pasajero no puede reservar múltiples asientos en el mismo viaje ni tener reservas contradictorias en el mismo horario.
RN008	Validación de Edad	Solo usuarios mayores de 18 años pueden registrarse como conductores y ofrecer viajes.

V. Fase de Desarrollo

5.1. Perfiles de Usuario

En RideUPT se identifican tres perfiles principales de usuario que interactúan directamente con el sistema: **Pasajero**, **Conductor** y **Administrador**. Cada uno tiene características, necesidades y responsabilidades distintas dentro de la plataforma.

5.1.1. Perfil: Pasajero

- **Descripción:**
  Estudiante universitario que **no cuenta con vehículo propio** o prefiere no usarlo, y busca una opción de transporte más económica y confiable para desplazarse hacia y desde la universidad.
- **Características típicas:**
  - Edad aproximada: **18 a 28 años**.
  - Sin vehículo propio o con acceso limitado a uno.
  - **Presupuesto mensual de transporte limitado.**
  - Alta dependencia de transporte público o taxis.
  - Uso frecuente de smartphone y aplicaciones móviles.
- **Necesidades principales:**
  - Buscar viajes disponibles hacia el campus en horarios específicos.
  - Filtrar viajes por origen, destino y hora.
  - Solicitar reservas de asiento de forma rápida.
  - Recibir notificaciones sobre aprobación/rechazo de reservas y cambios de estado del viaje.
  - Consultar historial de viajes realizados.
- **Objetivo dentro de RideUPT:**
  Reducir sus costos de transporte y mejorar la puntualidad, sin sacrificar seguridad ni comodidad.

5.1.2. Perfil: Conductor

- **Descripción:**
  Estudiante universitario que **posee vehículo propio** y está dispuesto a compartir asientos disponibles con otros estudiantes, reduciendo sus costos de movilidad y ayudando a descongestionar el campus.
- **Características típicas:**
  - Edad aproximada: **20 a 25 años**.
  - Vehículo propio (auto) en buenas condiciones.

  - Disponibilidad principalmente en **horarios de entrada y salida** de clases (mañana y tarde).
  - Motivación tanto **económica** (compartir gastos de combustible) como **social** (colaborar con otros estudiantes).

- **Necesidades principales:**
  - Configurar su perfil como conductor y registrar la información de su vehículo.
  - Crear viajes de manera sencilla, usando geolocalización para el origen y seleccionando el destino en el mapa.
  - Definir horarios, cantidad de asientos disponibles y precio por asiento (dentro del rango permitido).
  - Recibir y gestionar solicitudes de reserva (aprobar o rechazar).
  - Consultar historial de viajes ofrecidos y el comportamiento de sus pasajeros.
- **Objetivo dentro de RideUPT:**
  Optimizar el uso de su vehículo, compartir costos de transporte y contribuir a reducir la congestión vehicular alrededor de la universidad.

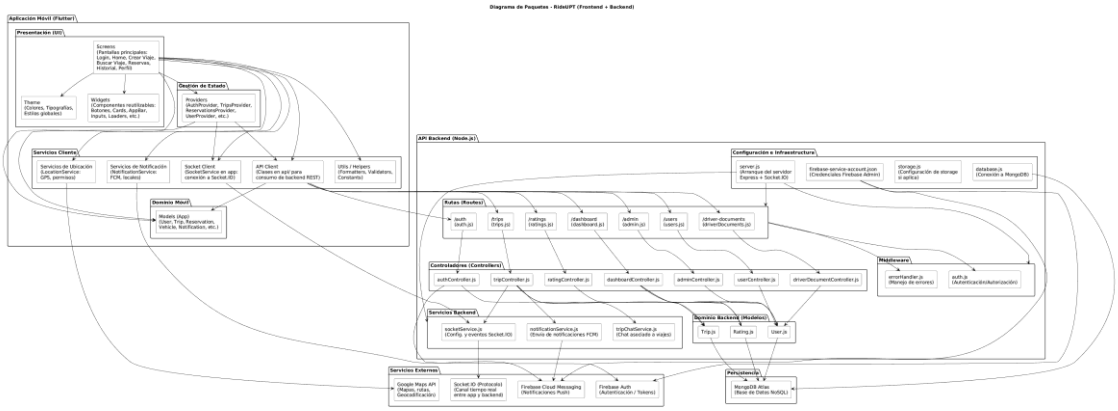
5.1.3. Perfil: Administrador

- **Descripción:**
  Usuario con privilegios especiales (por ejemplo, miembro del equipo técnico o responsable asignado por la universidad) encargado de **monitorear el uso del sistema**, revisar estadísticas y realizar acciones administrativas de control.
- **Características típicas:**
  - Conocimiento básico o intermedio de herramientas de monitoreo y paneles administrativos.
  - Relación directa con la **gestión y supervisión** de la plataforma.
- **Necesidades principales:**
  - Acceder a un **dashboard** con métricas de uso (cantidad de viajes, usuarios activos, etc.).
  - Revisar y, de ser necesario, intervenir en perfiles de usuario (bloqueo, desactivación).
  - Supervisar la correcta operación del sistema y recibir alertas ante situaciones anómalas.
- **Objetivo dentro de RideUPT:**
  Mantener el sistema **funcionando de forma estable, segura y alineada a las**

políticas de la institución, asegurando una buena experiencia general para pasajeros y conductores.

5.2. Modelo Conceptual

a) Diagrama de Paquetes



El diagrama de paquetes presentado representa la **arquitectura lógica y estructural** del sistema RideUPT, organizando sus componentes en módulos comprensibles y bien delimitados. El objetivo principal del diagrama es mostrar **cómo se agrupan las partes del sistema**, cómo interactúan entre sí y cómo se distribuyen las responsabilidades entre el **frontend (aplicación móvil)**, el **backend (API Node.js)**, la **persistencia de datos** y las **integraciones externas**.

A continuación, se describe cada uno de los paquetes y sus relaciones:

1. Aplicación Móvil (Flutter)

Este paquete representa la totalidad del código del cliente móvil, que es la interfaz mediante la cual interactúan los usuarios (pasajeros y conductores).
Se divide en subpaquetes:

Presentación (UI)

Abarca las pantallas principales de la app (screens), los widgets reutilizables y la capa visual común (theme).
Incluye pantallas como: Login, Home, Crear Viaje, Buscar Viaje, Reservas, Historial y Perfil del usuario.

Gestión de Estado

Contiene los *providers*, encargados de gestionar los estados esenciales del sistema:

- Sesión del usuario
- Viajes publicados

- Reservas
- Datos del conductor
- Notificaciones

Dominio Móvil (Models App)

Modelos principales utilizados por la app, como:
Usuario, Viaje, Reserva, Vehículo y Notificación.
Estas clases representan los datos que fluyen entre la interfaz y el backend.

Servicios Cliente

Incluye toda la lógica de infraestructura dentro del móvil:

- **API Client:** Comunicaciones con el backend vía REST.
- **LocationService:** GPS, permisos y geolocalización.
- **NotificationService:** Gestión y recepción de notificaciones (FCM).
- **SocketClient:** Conexión tiempo real con Socket.IO.
- **Utils/Helpers:** Validaciones y funciones generales.

Este módulo permite a la app comunicarse con el servidor, obtener datos, enviar solicitudes y recibir actualizaciones.

2. API Backend (Node.js)

Representa el servidor principal del sistema. Es la capa responsable de procesar la lógica del negocio, exponer datos mediante endpoints y coordinar servicios externos.

Se divide en submódulos:

Rutas (Routes)

Define los puntos de acceso HTTP para la aplicación móvil:
auth, users, trips, ratings, driver-documents, admin, dashboard.

- **• Controladores (Controllers)**

Contienen la lógica de negocio que responde a cada ruta.
Realizan validaciones, actualizan modelos, emiten notificaciones e interactúan con la base de datos.

Dominio Backend (Modelos)

Define las entidades persistidas en MongoDB:
User, Trip y Rating.

Estas entidades son el corazón del backend y representan las reglas del dominio real.

Middleware

Paquete encargado de tareas transversales:

- Autenticación JWT
- Manejo centralizado de errores

Servicios Backend

Servicios especializados que soportan operaciones de la API:

- Notificaciones push (Firebase Admin)
- Socket.IO (actualización en tiempo real)
- Chat entre pasajero y conductor en un viaje

Configuración e Infraestructura

Incluye archivos de configuración:
Conexión a la base de datos, credenciales Firebase, configuraciones de servidor (server.js).

3. Persistencia

MongoDB Atlas

Base de datos NoSQL en la nube donde se almacenan:

- Usuarios
- Viajes
- Reservas
- Calificaciones
- Token FCM

Todos los modelos del backend dependen de esta capa para persistir datos.

4. Servicios Externos

Este paquete representa las integraciones clave que el sistema necesita para funcionar adecuadamente:

- **Firebase Auth:** Autenticación segura y validación de identidad.
- **Firebase Cloud Messaging (FCM):** Envío de notificaciones instantáneas.
- **Google Maps API:** Geolocalización, mapas, rutas y cálculo de distancias.
- **Socket.IO:** Canal de comunicación bidireccional para eventos en tiempo real.

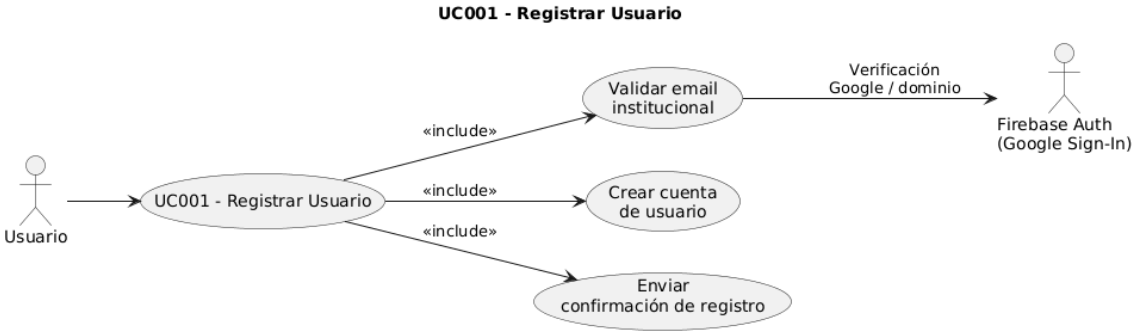
5. Relaciones entre paquetes

Se puede apreciar en el diagrama:

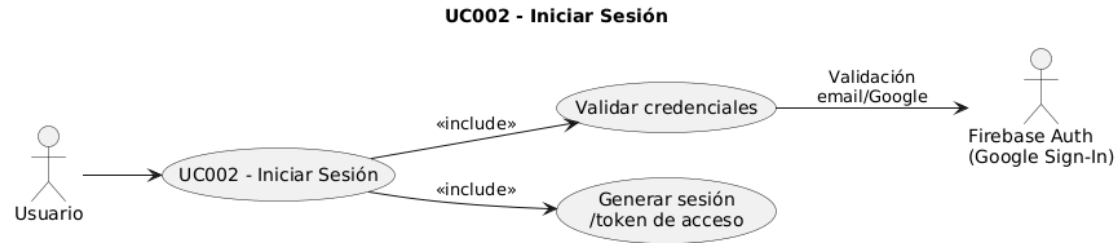
- **La app móvil consume las rutas del backend** mediante el API Client.
- Los **providers** del frontend dependen de los **modelos** del dominio y de los servicios API.
- El backend usa **controladores**, que a su vez dependen de **modelos**, **servicios** y **configuraciones**.
- MongoDB es la capa persistente usada por todos los modelos.
- Firebase y Google Maps funcionan como **servicios externos** que complementan las funciones principales.
- Socket.IO establece comunicación de tiempo real entre backend y frontend.

b) Diagrama de Casos de Uso

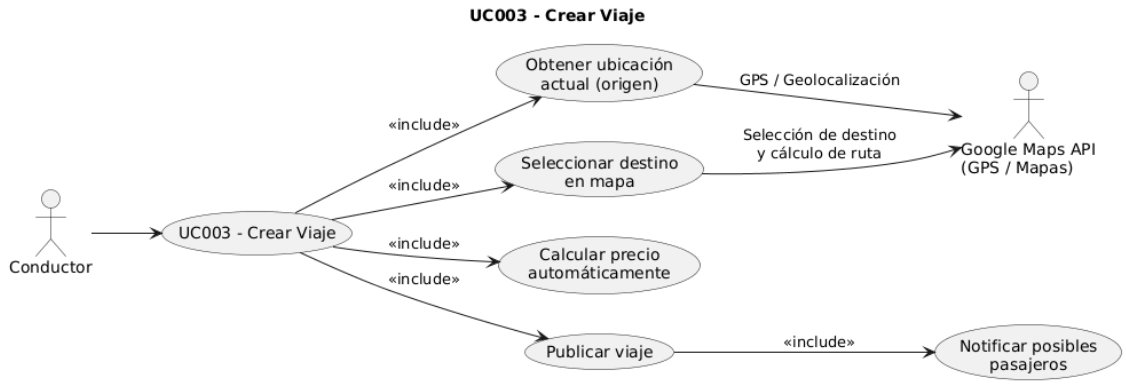
UC001 – Registrar Usuario



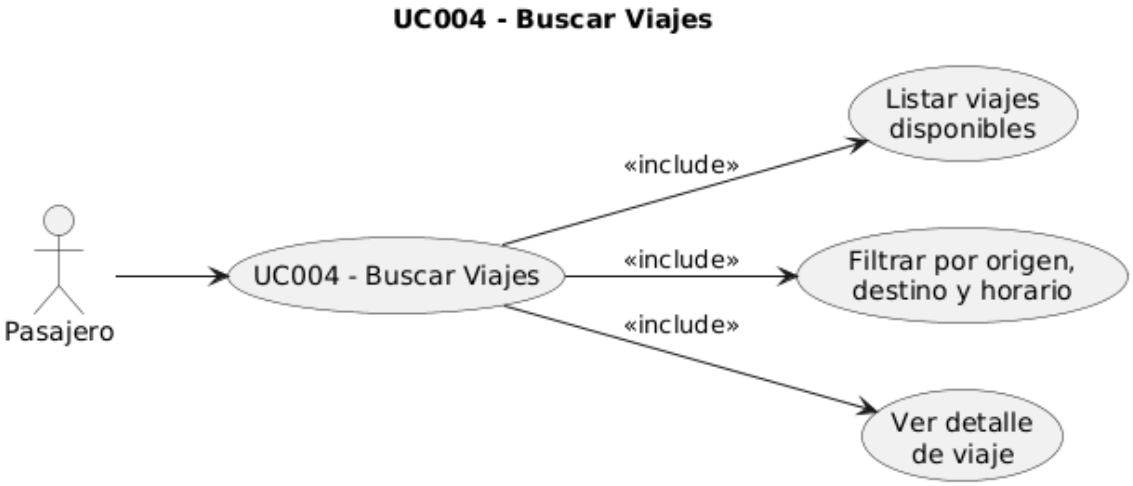
UC002 – Iniciar Sesión



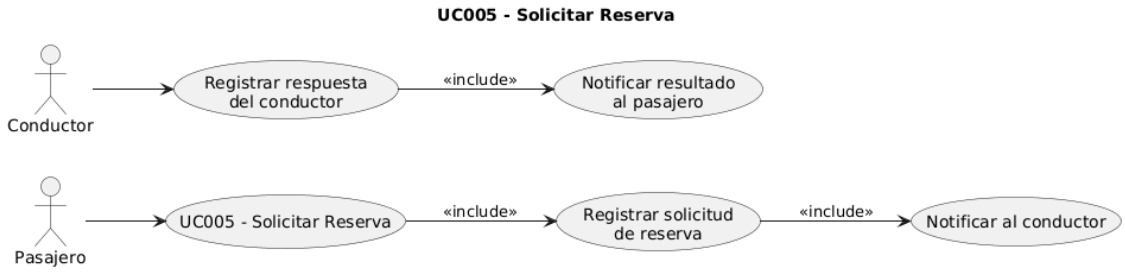
UC003 – Crear Viaje



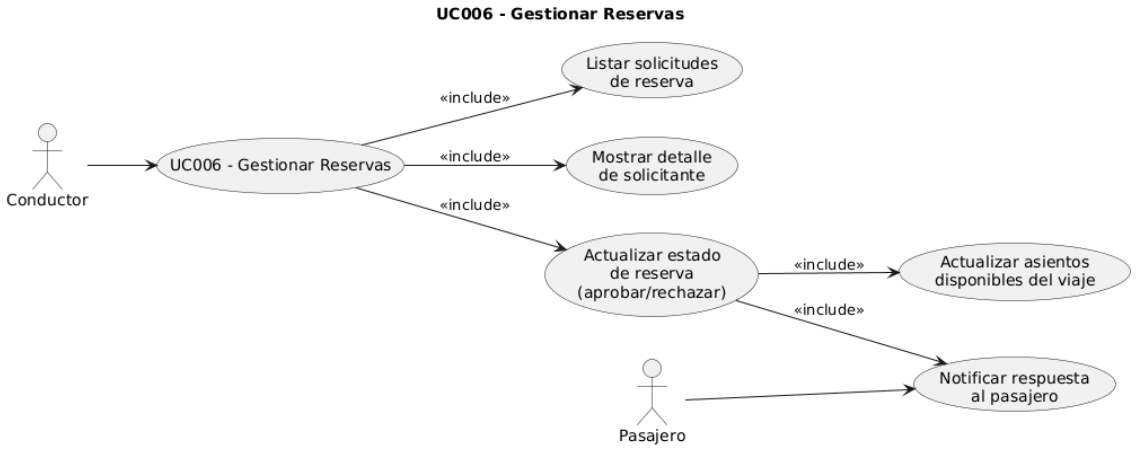
UC004 – Buscar Viajes



UC005 – Solicitar Reserva



UC006 – Gestionar Reservas



c) Escenarios de Caso de Uso (Narrativa)

UC001 – Registrar Usuario

Campo	Detalle
Actor principal	Usuario (Estudiante – Pasajero/Conductor potencial)
Propósito	Registrarse en RideUPT para poder utilizar la aplicación como pasajero y/o conductor.
Precondiciones	- El usuario posee un **correo institucional válido** (ej. @virtual.upt.pe).- La aplicación RideUPT está instalada y funcional.- El backend y los servicios de autenticación (Firebase/Google Sign-In) están operativos.
Flujo principal	1) El usuario abre la aplicación RideUPT.2) Selecciona la opción **"Registrarse"** o **"Continuar con Google"**.3) Completa los datos requeridos (si aplica) y autoriza Google Sign-In.4) El sistema valida que el correo pertenezca al dominio institucional permitido.5) El usuario confirma el registro.6) El sistema crea la cuenta de usuario y almacena la información básica (nombre, correo, código de estudiante, rol inicial).7) Opcionalmente, el sistema muestra un mensaje de bienvenida o un pequeño tutorial inicial.
Flujos alternos	**A1)** Correo no institucional → El sistema muestra un mensaje indicando que solo se aceptan correos institucionales y no crea la cuenta.**A2)** Correo ya registrado → El sistema indica que la cuenta ya existe y sugiere ir a **"Iniciar Sesión"**.**A3)** Error de conexión con el servidor o servicio de autenticación → El sistema muestra un mensaje de error y permite reintentar.
Postcondiciones	- El usuario queda registrado en RideUPT.- Sus datos básicos se almacenan en la base de datos.- El sistema puede dejar la sesión iniciada o redirigir a la pantalla de inicio de sesión, según el flujo implementado.
Controlador/Vistas	**Backend:** authController.js, rutas en routes/auth.js (endpoint de registro/Google Sign-In).**Frontend (App):** Pantalla de **Registro/Inicio de Sesión** (Login/SignUp Screen), integración con **Firebase Auth / Google Sign-In**.

UC002 – Iniciar Sesión

Campo	Detalle
Actor principal	Usuario (Estudiante registrado)
Propósito	Autenticarse en RideUPT para acceder a las funciones de pasajero o conductor.
Precondiciones	- El usuario tiene una cuenta registrada en RideUPT.- La aplicación RideUPT está instalada y operativa.- Los servicios de autenticación (Firebase/Google Sign-In) y el backend están disponibles.
Flujo principal	1) El usuario abre RideUPT.2) Selecciona **"Iniciar Sesión"** (email/contraseña o Google Sign-In, según la implementación final).3) El sistema valida las credenciales o el token devuelto por Google.4) Si las credenciales son válidas, el sistema genera una sesión/tokens de acceso (JWT).5) El sistema carga la información del perfil del usuario (rol, nombre, foto, etc.).6) El usuario es redirigido a la **pantalla principal** (Home) según su rol.
Flujos alternos	**A1)** Credenciales inválidas → El sistema muestra un mensaje claro indicándolo y permanece en la pantalla de login.**A2)** Cuenta no encontrada → El sistema sugiere registrarse primero.**A3)** Error con el servicio de autenticación → Mensaje de error y opción de reintentar.**A4)** Usuario inactivo/bloqueado por administrador → El sistema muestra un mensaje de restricción y no permite el acceso.
Postcondiciones	- Sesión activa con el rol del usuario cargado (pasajero, conductor o ambos).- Menús, pantallas y opciones visibles según el rol.
Controlador/Vistas	**Backend:** authController.js, rutas en routes/auth.js (login, validación de token).**Frontend (App):** Pantalla de **Login/SignIn**, lógica de navegación hacia **Home** / **Pantalla principal**.

UC003 – Crear Viaje

Campo	Detalle
Actor principal	Conductor (Estudiante con rol de conductor habilitado)
Propósito	Crear un nuevo viaje que pueda ser visto y reservado por pasajeros.
Precondiciones	- El usuario está **autenticado** en el sistema.- El usuario tiene rol de **conductor** configurado y vehículo registrado.- Servicios de geolocalización (GPS y Google Maps) están disponibles.- El backend y la base de datos están operativos.
Flujo principal	1) El conductor ingresa a la aplicación y accede a la opción **"Crear Viaje"**.2) El sistema detecta la ubicación actual del conductor como origen (mediante GPS).3) El conductor selecciona el destino en el mapa o lo busca por nombre/dirección.4) El conductor especifica: horario de salida, número de asientos disponibles y una breve descripción opcional.5) El sistema calcula automáticamente el **precio por asiento** en función de la distancia, dentro del rango permitido por las reglas de negocio.6) El conductor confirma la creación del viaje.7) El sistema registra el viaje en la base de datos con estado inicial (por ejemplo, "esperando") y establece la hora de expiración (10 minutos).8) El sistema puede notificar a posibles pasajeros o reflejar el nuevo viaje en la lista de viajes disponibles.

Flujos alternos	**A1)** Permisos de ubicación denegados → El sistema solicita permisos; si el usuario los rechaza, se le permite ingresar el origen manualmente o no continuar.**A2)** Destino no válido o no encontrado → El sistema muestra un mensaje y pide corregir o seleccionar de nuevo en el mapa.**A3)** Datos incompletos (horario, asientos, etc.) → El sistema no permite la creación y marca los campos obligatorios.**A4)** Error de conexión o base de datos → Mensaje de error y opción de reintentar guardar el viaje.
Postcondiciones	- Un nuevo registro de viaje se almacena en la base de datos con origen, destino, horario, precio, asientos disponibles y estado inicial.- El viaje aparece disponible para los pasajeros en la búsqueda (si está dentro de las condiciones).
Controlador/Vistas	**Backend:** tripController.js, rutas en routes/trips.js (creación de viaje).**Frontend (App):** Pantalla **Crear Viaje** (formulario + mapa) y uso de servicios de **Location** y **Google Maps API**.

UC004 – Buscar Viajes

Campo	Detalle
Actor principal	Pasajero (Estudiante con rol de pasajero)
Propósito	Permitir al pasajero encontrar viajes disponibles que coincidan con su origen, destino y horario.
Precondiciones	- El usuario está **autenticado**.- Tiene rol de **pasajero** configurado (o al menos acceso a la funcionalidad de búsqueda).- Existen viajes publicados y vigentes en el sistema.
Flujo principal	1) El pasajero accede a la pantalla principal o módulo de **"Buscar Viajes"**.2) El sistema puede mostrar por defecto viajes cercanos al pasajero (según ubicación actual) o listados recientes.3) El pasajero aplica filtros por origen, destino y/o horario (ej. "hacia UPT", "8:00–9:00 AM").4) El sistema consulta la base de datos y obtiene los viajes que cumplen los criterios y aún no han expirado ni se encuentran llenos.5) El sistema muestra la lista de resultados con detalles básicos (conductor, precio, hora, asientos).6) El pasajero puede seleccionar un viaje para ver la información detallada (ruta, descripción, tal vez calificación del conductor, etc.).
Flujos alternos	**A1)** No hay viajes que coincidan con los filtros → El sistema muestra un mensaje indicando que no hay resultados y sugiere cambiar filtros u horarios.**A2)** Error de conexión → El sistema indica el problema y permite reintentar la búsqueda.**A3)** Viaje caducado entre consulta y selección → El sistema informa que el viaje ya no está disponible y actualiza la lista.
Postcondiciones	- El pasajero ha visualizado una lista de viajes vigentes compatibles con sus filtros.- El pasajero puede decidir avanzar a **solicitar una reserva (UC005)**.
Controlador/Vistas	**Backend:** tripController.js, rutas en routes/trips.js (búsqueda/listado de viajes).**Frontend (App):** Pantallas de **Home/Lista de Viajes** y **Detalle de Viaje**, integración con el **TripsProvider** y servicio API.

UC005 – Solicitar Reserva

Campo	Detalle
Actor principal	Pasajero (Estudiante)
Propósito	Permitir al pasajero reservar un asiento en un viaje específico.
Precondiciones	- El pasajero está autenticado y tiene perfil activo.- El pasajero ha encontrado un viaje disponible mediante UC004.- El viaje tiene al menos un asiento disponible y no está expirado o cancelado.
Flujo principal	1) El pasajero selecciona un viaje de la lista o desde el detalle de viaje.2) El pasajero pulsa la opción **"Solicitar Reserva"**.3) El sistema registra la solicitud de reserva asociándola al usuario y al viaje.4) El sistema envía una notificación al conductor indicando la nueva solicitud de pasajero (via FCM / Socket.IO).5) El conductor recibe la notificación y procederá a gestionar la solicitud en UC006.6) Una vez que el conductor responde, el sistema actualiza el estado de la solicitud (confirmada/rechazada) y notifica al pasajero.
Flujos alternos	**A1)** El viaje se llena justo antes de la solicitud → El sistema muestra que ya no hay asientos disponibles y no registra la reserva.**A2)** El viaje se marca como expirado → El sistema indica que el viaje ya no está disponible.**A3)** Error al registrar la reserva (DB o red) → Mensaje de error y posibilidad de reintentar.**A4)** El pasajero ya tiene una reserva activa en ese mismo viaje → El sistema impide la duplicidad y muestra un aviso.
Postcondiciones	- Una solicitud de reserva queda registrada con estado inicial (ej. "pendiente").- El conductor recibe la notificación para revisarla.- El pasajero queda a la espera de la decisión del conductor.
Controlador/Vistas	**Backend:** tripController.js (gestión de pasajeros/reservas dentro del viaje), posiblemente usando modelos Trip y estructuras internas de pasajeros.**Servicios:** notificationService.js, socketService.js para notificaciones en tiempo real.**Frontend (App):** Pantalla de **Detalle de Viaje** con botón **"Solicitar Reserva"**, integración con API y notificaciones.

UC006 – Gestionar Reservas

Campo	Detalle
Actor principal	Conductor (Estudiante)
Propósito	Permitir al conductor revisar y gestionar las solicitudes de reserva de su viaje (aprobar/rechazar).
Precondiciones	- El conductor está autenticado.- Tiene al menos un viaje activo con una o más solicitudes de reserva pendientes.- El backend y los servicios de notificaciones están operativos.
Flujo principal	1) El conductor abre la aplicación y accede a la sección de **Reservas** o al detalle de su viaje.2) El sistema muestra la lista de solicitudes de reserva asociadas al viaje (pendientes, confirmadas, rechazadas).3) El conductor selecciona una solicitud pendiente para ver detalles del pasajero (nombre, código, quizá historial/calificación, etc.).4) El conductor decide **aprobar** o **rechazar** la solicitud.5) El sistema actualiza el estado de la reserva en la base de datos (confirmada o rechazada).6) Si se aprueba, el sistema reduce el número de asientos disponibles del viaje y puede marcar el viaje como "completo" si ya no queda cupo.7) El sistema envía una notificación al pasajero con el resultado (aprobado/rechazado).

Flujos alternos	**A1)** Solicitud ya gestionada (otro dispositivo) → El sistema muestra que la solicitud ya fue atendida y actualiza la lista.**A2)** Sin solicitudes pendientes → El sistema puede mostrar la lista vacía o solo las reservas confirmadas.**A3)** Error al actualizar la reserva o los asientos → Mensaje de error y opción de reintentar la acción.**A4)** El viaje ya está marcado como cancelado o expirado → El sistema no permite cambios y muestra aviso al conductor.
Postcondiciones	- Las reservas quedan actualizadas con el estado correspondiente.- El número de asientos disponibles del viaje se mantiene consistente con las reservas aprobadas.- El pasajero es informado sobre la decisión del conductor.
Controlador/Vistas	**Backend:** tripController.js (actualización de reservas y asientos), notificationService.js y socketService.js para avisos en tiempo real.**Frontend (App):** Pantalla de **Gestión de Reservas** o **Detalle de Viaje (Conductor)** que permite aprobar/rechazar solicitudes.

5.3. Modelo Lógico

a) Análisis de Objetos

En esta sección se identifican y describen los **objetos principales del dominio** de RideUPT, basados en:

- Los casos de uso (UC001–UC006).
- El modelo conceptual que ya definiste (Usuario, Viaje, Reserva, Vehículo, Notificación).
- La estructura real del backend (User.js, Trip.js, Rating.js).

Estos objetos serán la base del **Diagrama de Clases** y de los **diagramas de interacción** (actividades y secuencia) que construiremos luego.

1. **Clase: Usuario**

Rol en el sistema:
Representa a cualquier estudiante que utiliza la aplicación, ya sea como **pasajero**, **conductor** o ambos. Es el objeto central para autenticación, perfil y relaciones con viajes y reservas.

Atributos principales (según tu definición):

- id: String
- firstName: String
- lastName: String

- email: String
- password: String (hash, si aplica)
- role: String (driver, passenger o ambos según implementación)
- phone: String
- university: String
- studentId: String (código de estudiante)
- profilePhoto: String
- age: Number
- gender: String
- bio: String
- fcmToken: String (para notificaciones push)
- createdAt: Date
- updatedAt: Date

Métodos (responsabilidades):

- getFullName(): String → Devuelve el nombre completo del usuario.
- getInitials(): String → Devuelve iniciales (ej. para avatar).
- isDriver(): Boolean → Indica si el usuario tiene rol de conductor.
- isPassenger(): Boolean → Indica si el usuario tiene rol de pasajero.
- isProfileComplete(): Boolean → Verifica si el perfil contiene todos los datos mínimos necesarios.

Relaciones:

- Un **Usuario** puede tener asociado **un Vehículo** (si es conductor).
- Un **Usuario** puede crear múltiples **Viajes** (como conductor).
- Un **Usuario** puede estar asociado a múltiples **Reservas/TripPassenger** (como pasajero).
- Un **Usuario** puede recibir múltiples **Notificaciones**.
- Un **Usuario** puede tener múltiples **Ratings** (ser calificado).

2. **Clase: Vehículo**

Rol en el sistema:
Representa el vehículo con el que un estudiante-conductor ofrece viajes.

Atributos principales:

- make: String (marca, ej. Toyota)
- model: String (modelo, ej. Yaris)
- year: Number
- color: String
- licensePlate: String (placa)
- totalSeats: Number (capacidad total del vehículo)

Métodos:

- getDisplayName(): String → Devuelve una cadena amigable, por ejemplo: "Toyota Yaris 2019".
- getAvailableSeats(): Number → Devuelve el número de asientos disponibles (en función de totalSeats y asientos usados en viajes activos).

Relaciones:

- Un **Vehículo** pertenece a un único **Usuario** (conductor).
- Sus datos se usan para calcular **asientos máximos** en los viajes.

3. **Clase: LocationPoint**

Rol en el sistema:
Representa un punto geográfico, utilizado para origen y destino de los viajes.

Atributos principales:

- name: String (nombre del lugar, referencia, dirección aproximada)
- type: String (normalmente "Point" para geometría GeoJSON)
- coordinates: [Number, Number] → [lng, lat]

Métodos:

- getLatitude(): Number
- getLongitude(): Number
- getDistanceTo(other: LocationPoint): Number → Devuelve la distancia entre dos puntos (usado para cálculo de precio).

Relaciones:

- Un **Viaje** tiene un origin: LocationPoint y un destination: LocationPoint.

4. **Clase: Viaje**

Rol en el sistema:
Es el objeto clave que representa un trayecto ofrecido por un conductor, con origen, destino, horario y asientos disponibles.

Atributos principales:

- id: String
- driver: ObjectId (ref: Usuario)
- origin: LocationPoint
- destination: LocationPoint
- departureTime: Date
- expiresAt: Date
- availableSeats: Number
- seatsBooked: Number
- pricePerSeat: Number
- description: String
- status: String (esperando, completo, en-proceso, expirado, cancelado)
- passengers: [TripPassenger]
- createdAt: Date
- updatedAt: Date

Métodos (lógica de negocio):

- hasTimeExpired(): Boolean
- minutesRemaining(): Number
- isInProgress(): Boolean
- isCompleted(): Boolean
- isActive(): Boolean
- isFull(): Boolean
- isExpired(): Boolean
- acceptsRequests(): Boolean (ej. viaje activo, no lleno, no expirado)
- isCancelled(): Boolean

Relaciones:

- Un **Viaje** es creado por un **Usuario** (conductor).
- Un **Viaje** tiene múltiples **TripPassenger/Reservas** asociadas.

- Un **Viaje** está relacionado con uno o varios **Ratings** (opiniones al finalizar, si se implementa).

5. **Clase: TripPassenger (Reserva)**

En tu modelo conceptual hablas de **Reserva** como entidad, y en análisis de objetos diste la clase **TripPassenger**. Aquí lo tratamos como la implementación concreta de una reserva de asiento.

Rol en el sistema:
Representa la participación de un pasajero en un viaje, con su estado de solicitud.

Atributos principales:

- user: ObjectId (ref: Usuario)
- status: String (pending, confirmed, rejected)
- bookedAt: Date

Métodos:

- isPending(): Boolean
- isConfirmed(): Boolean
- isRejected(): Boolean

Relaciones:

- Una instancia de **TripPassenger** representa la **reserva** de un usuario en un **Viaje**.
- Un **Viaje** tiene un arreglo de passengers: [TripPassenger].
- Un **Usuario** puede aparecer en varios TripPassenger (varias reservas en distintos viajes, en tiempos distintos).

6. **Clase: Reserva (Conceptual)**

Aunque en código se usa TripPassenger, a nivel de análisis es útil pensar en la entidad **Reserva** como tal.

Rol en el sistema:
Conceptualmente, representa la **solicitud de un asiento** hecha por un pasajero y su estado (pendiente, confirmada, rechazada).

Atributos (conceptuales, derivados de TripPassenger):

- id: String
- pasajero: Usuario
- viaje: Viaje

- status: String (pendiente, confirmada, rechazada)
- fechaSolicitud: Date

Relaciones:

- Una **Reserva** vincula un **Usuario (pasajero)** con un **Viaje (conductor)**.
- Un **Viaje** tiene N reservas, y un **Usuario** puede tener muchas reservas en distintos viajes (pero no múltiples activas en el mismo viaje, por regla de negocio).

7. **Clase: Notificación**

Rol en el sistema:
Representa mensajes importantes que el sistema envía a los usuarios: solicitudes, aprobaciones, rechazos, cambios de estado, etc.

Atributos típicos (conceptuales):

- id: String
- user: ObjectId (ref: Usuario) (destinatario)
- title: String
- message: String
- type: String (ej. reserva_pendiente, reserva_aprobada, reserva_rechazada, viaje_expirado)
- createdAt: Date
- read: Boolean

Relaciones:

- Una **Notificación** está asociada a un **Usuario**.
- Generalmente se dispara en eventos relacionados con **Viajes** o **Reservas**.

8. **Clase: Rating (Calificación)**

Rol en el sistema:
Permite que, al finalizar un viaje, se registren calificaciones/opiniones entre usuarios (generalmente pasajero → conductor).

Atributos (a partir del modelo Rating.js, que no detallamos aún):

Mínimos esperables, sin inventar demasiado:

- id: String
- fromUser: ObjectId (ref: Usuario)

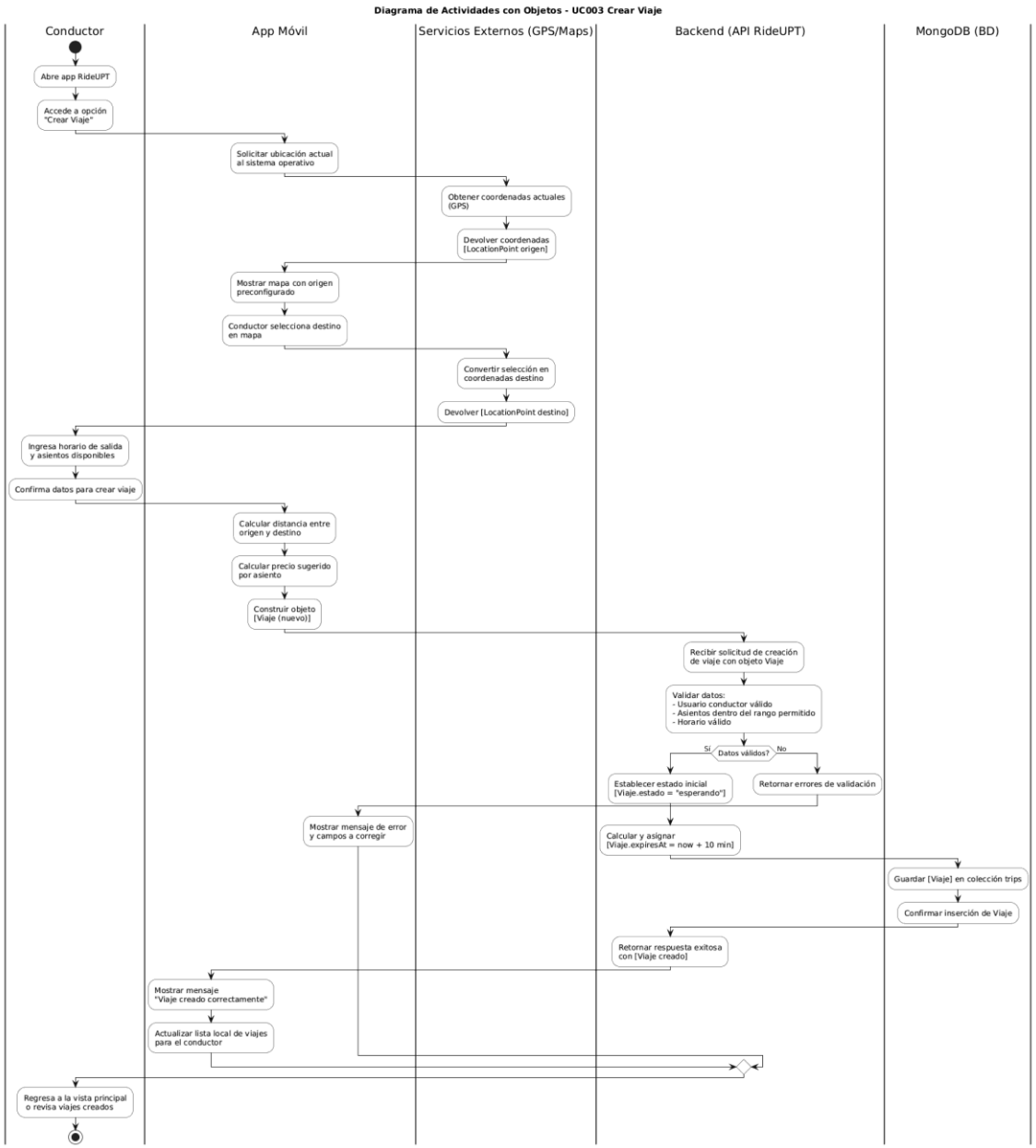
- toUser: ObjectId (ref: Usuario)
- trip: ObjectId (ref: Viaje)
- score: Number (por ejemplo, 1–5)
- comment: String (opcional)
- createdAt: Date

Relaciones:

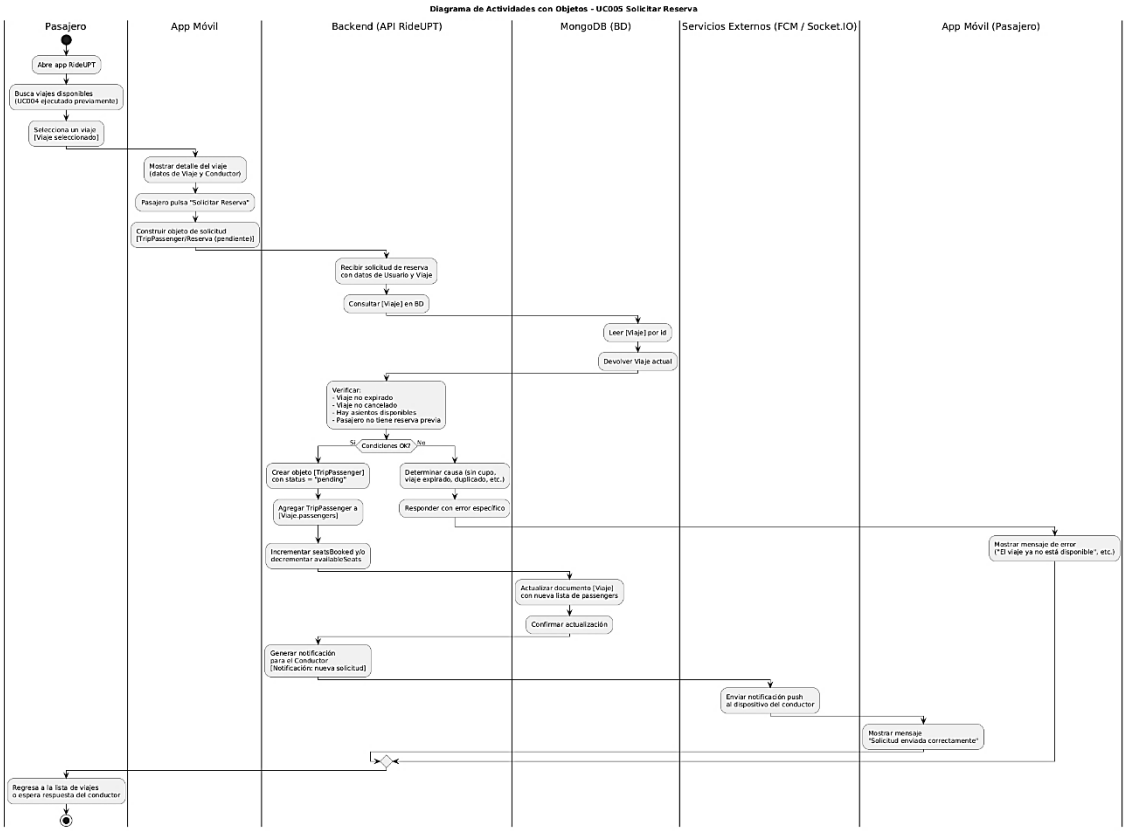
- Un **Rating** está asociado a un **Viaje** y a dos usuarios: el que califica y el calificado.
- Un **Usuario** puede tener múltiples ratings recibidos y emitidos.

b) Diagrama de Actividades con Objetos

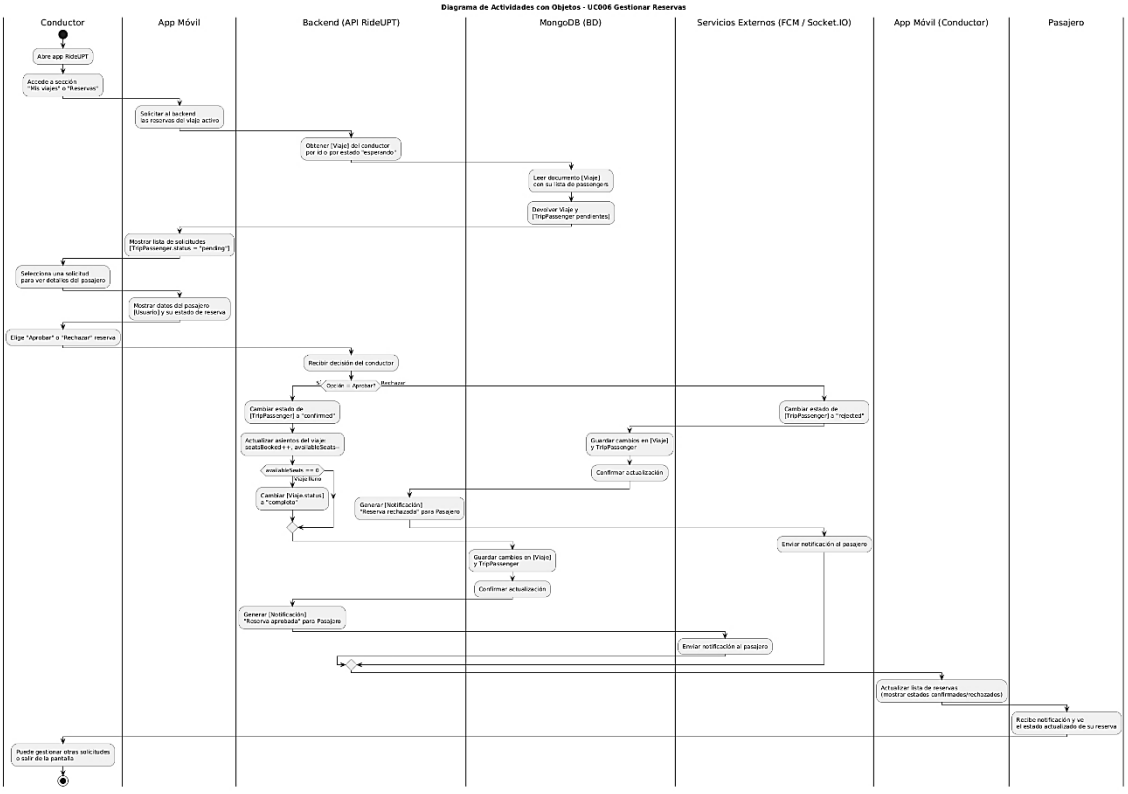
1. Crear Viaje

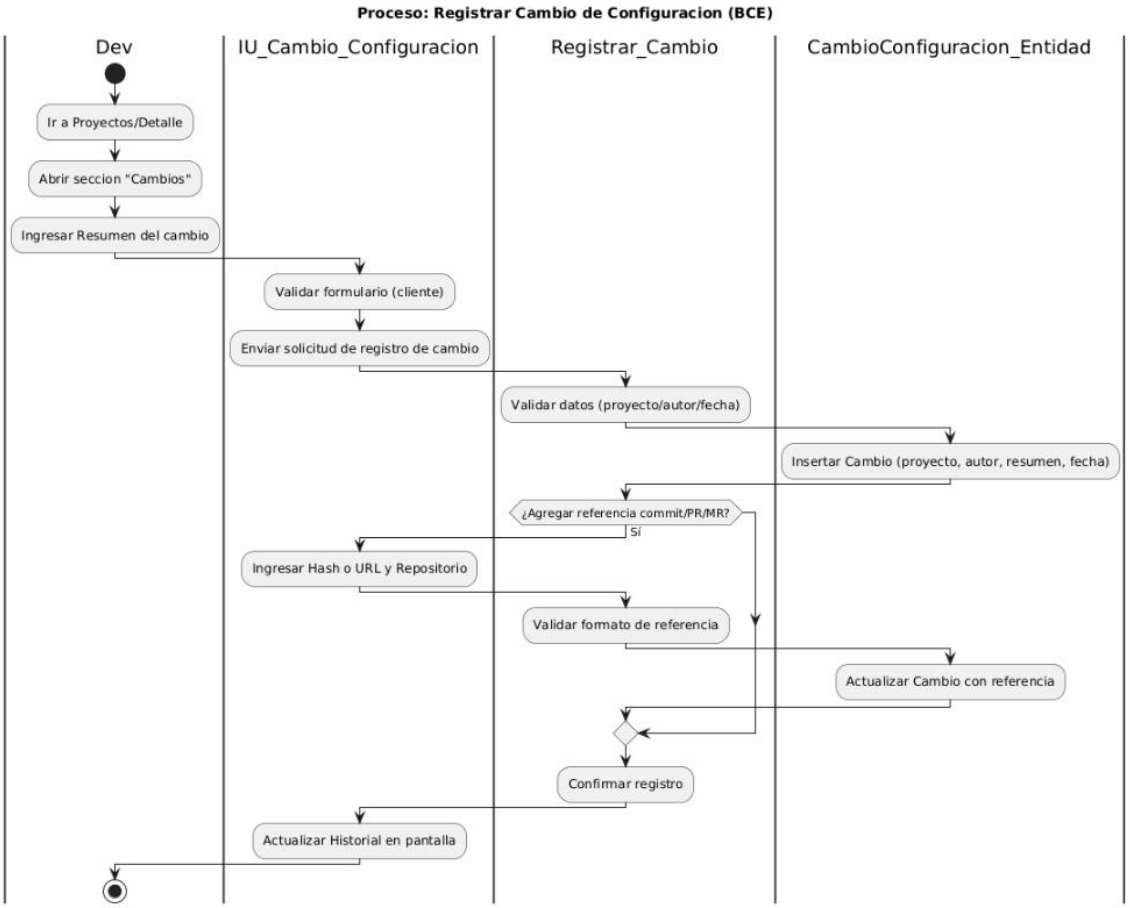


2. Solicitar Reserva



3. Gestionar Reservas





c) Diagrama de Secuencia

Iniciar Sesión

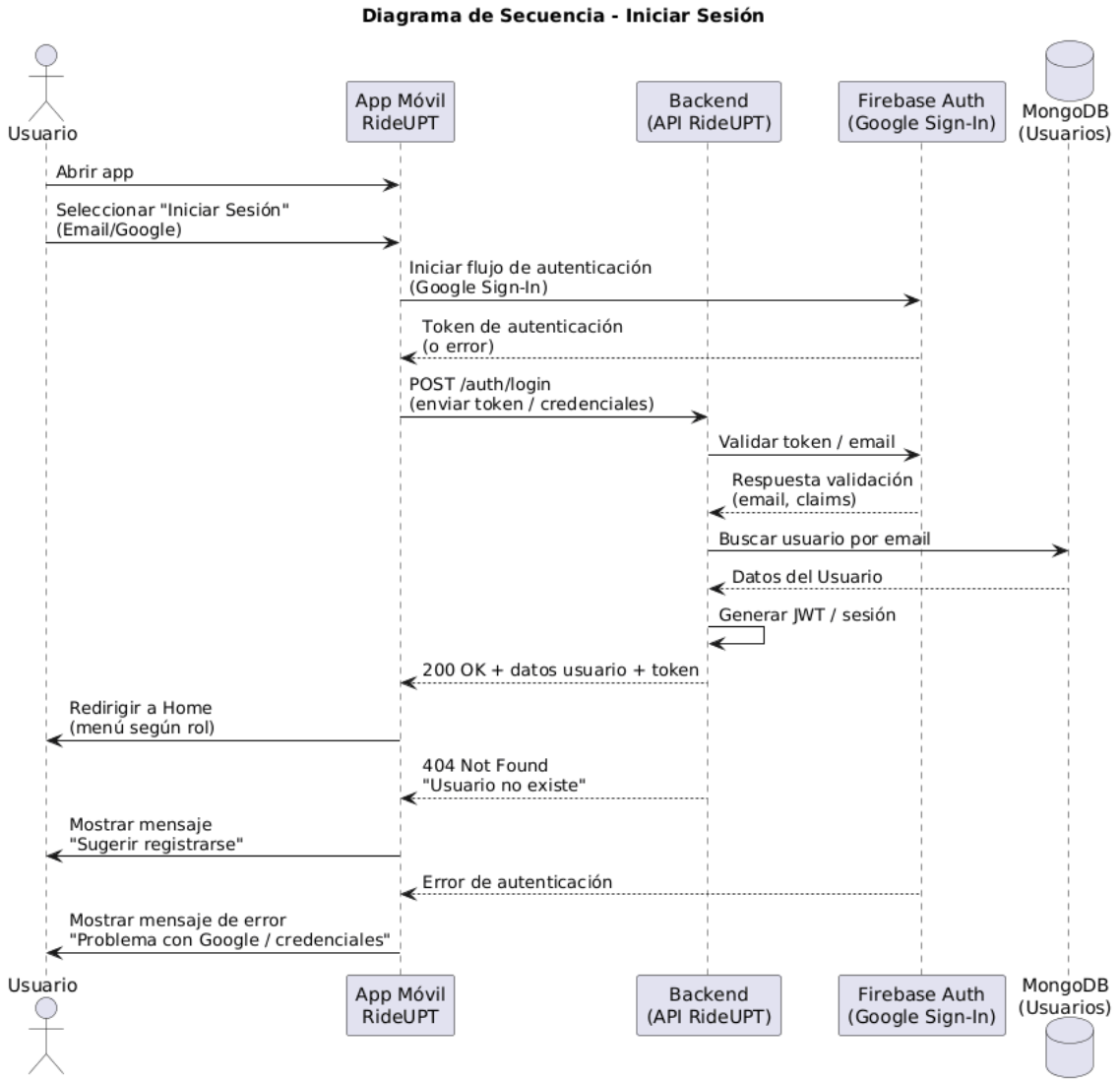
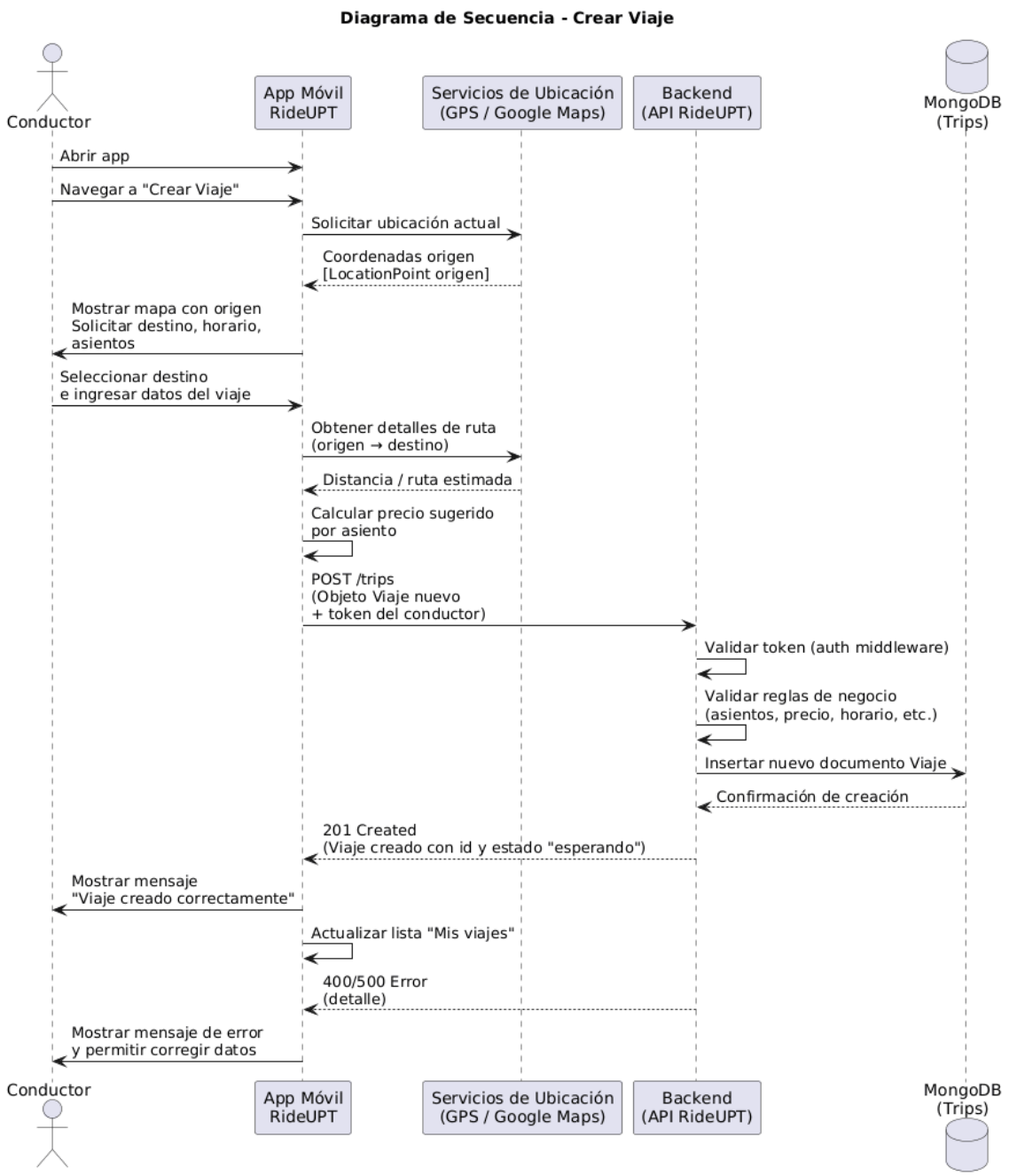


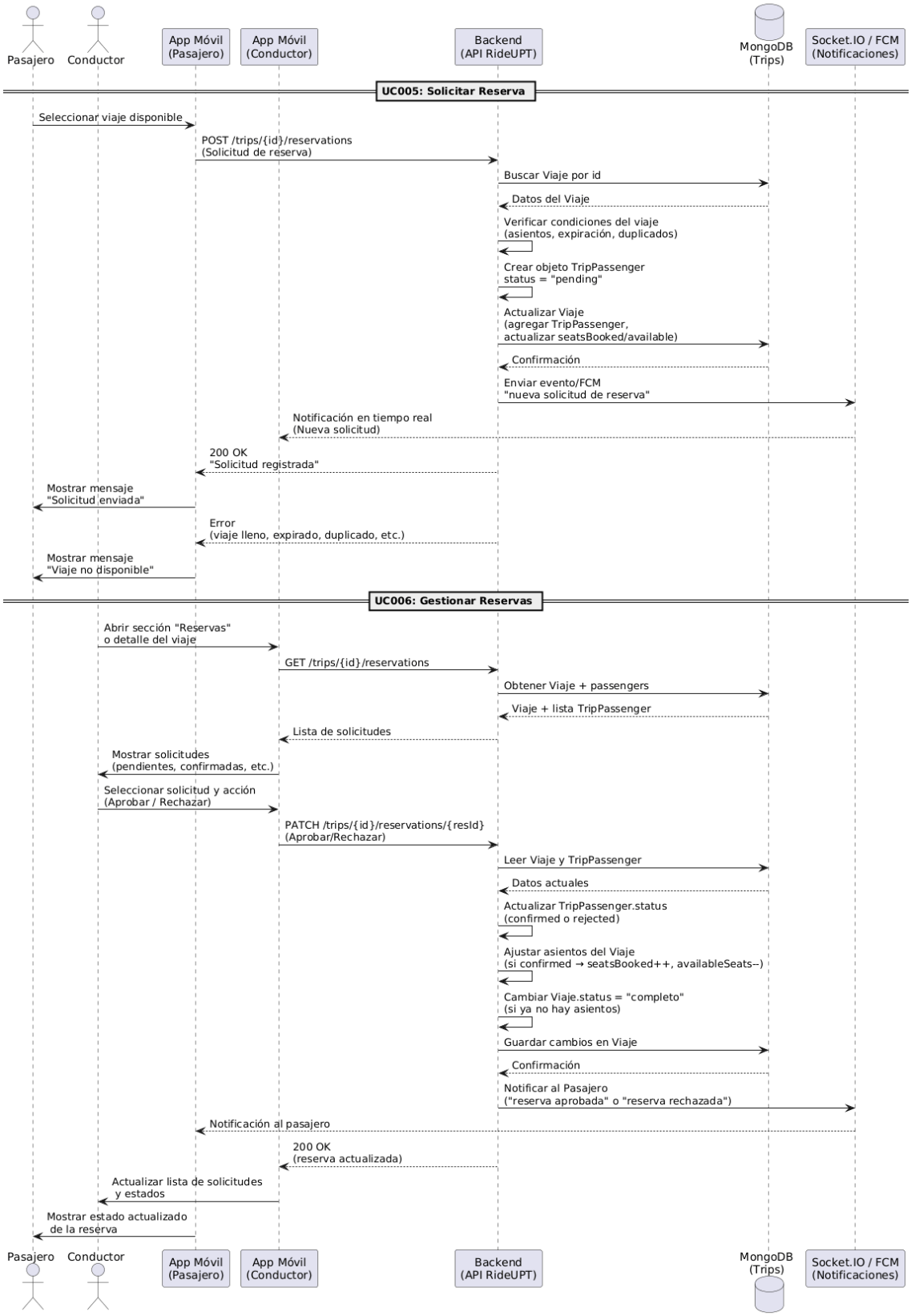
Diagrama de Secuencia - Iniciar Sesión
Usuario
App Móvil RideUPT
Backend (API RideUPT)
Firebase Auth (Google Sign-In)
MongoDB (Usuarios)
Abrir app
Seleccionar "Iniciar Sesión" (Email/Google)
Iniciar flujo de autenticación (Google Sign-In)
Token de autenticación (o error)
POST /auth/login (enviar token / credenciales)
Validar token / email
Respuesta validación (email, claims)
Buscar usuario por email
Datos del Usuario
Generar JWT / sesión
200 OK + datos usuario + token
Redirigir a Home (menú según rol)
404 Not Found "Usuario no existe"
Mostrar mensaje "Sugerir registrarse"
Error de autenticación
Mostrar mensaje de error "Problema con Google / credenciales"
Usuario
App Móvil RideUPT
Backend (API RideUPT)
Firebase Auth (Google Sign-In)
MongoDB (Usuarios)

Crear Viaje

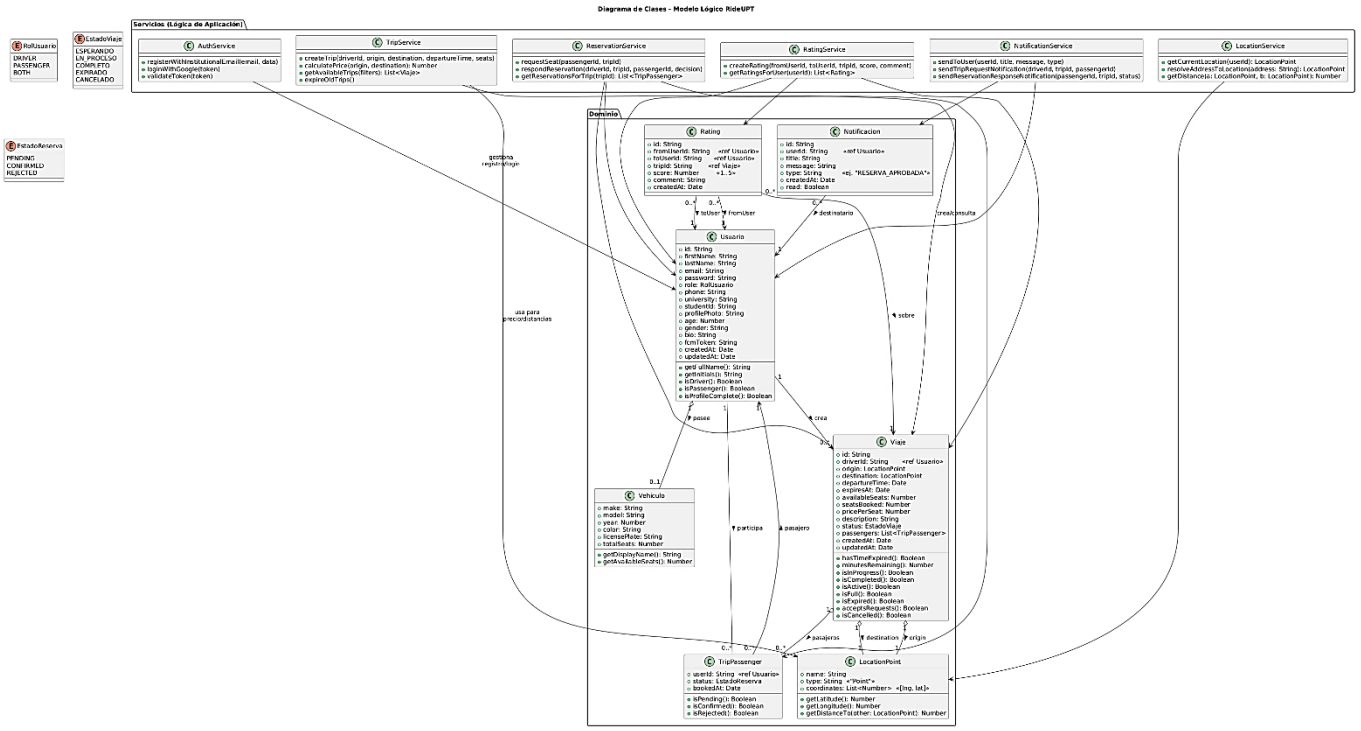


Solicitar y Gestionar Reservas

Diagrama de Secuencia - Solicitar y Gestionar Reservas



d) Diagrama de Clases



CONCLUSIONES

El análisis realizado permitió comprender de manera profunda el funcionamiento interno de RideUPT, identificando los objetos esenciales que conforman el sistema y las relaciones que mantienen entre sí. A través del Modelo Lógico se logró representar, con precisión, los elementos del dominio como usuarios, viajes, reservas, ubicaciones y calificaciones, reflejando fielmente la estructura real implementada en el backend y el comportamiento esperado en la aplicación móvil. Esta alineación entre la teoría y la implementación garantiza que la documentación sea coherente, útil y aplicable directamente al desarrollo del proyecto.

Los diagramas de actividades y de secuencia permitieron visualizar cómo fluye la información entre los distintos componentes del sistema, desde la interfaz móvil hasta los servicios backend y la base de datos, pasando también por integraciones externas como Google Maps y Firebase. Esta representación gráfica evidenció que RideUPT cuenta con procesos bien definidos y consistentes, donde cada actor —pasajero, conductor o administrador— interactúa con objetos específicos y recibe respuestas claras dentro de flujos controlados y seguros.

El diagrama de clases consolidó la estructura lógica del sistema, mostrando que RideUPT posee un dominio robusto y organizado, con entidades que encapsulan responsabilidades específicas y mantienen relaciones coherentes entre sí. La

incorporación de enumeraciones para estados y roles permitió reforzar la claridad del modelo y asegurar que los comportamientos del sistema estén correctamente acotados por reglas de negocio formales.

En conjunto, los análisis y diagramas producidos en esta sección permiten afirmar que RideUPT se apoya en una arquitectura bien sustentada, modular y con una separación clara entre responsabilidades. Esto facilita no solo el mantenimiento y la escalabilidad futura, sino también la comprensión del sistema por parte de otros desarrolladores, docentes y evaluadores. Con una base conceptual sólida y una representación gráfica completa, RideUPT se posiciona como un proyecto técnicamente maduro y preparado para su despliegue y evolución.