UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

DEPARTAMENTO DE ELECTROTECNIA E INFORMÁTICA
INGENIERÍA EN INFORMÁTICA

Desarrollo de Back-End para proyecto de Optimización de Solicitudes de Viaje en FlotUp, mediante Asignación Dinámica y Viajes Compartidos

Sebastian Antonio Albornoz Diaz

sebastian.albornozd@usm.cl

Dagoberto Cabrera
Profesor Guía

Diego Caceres
Profesor Correferente

Resumen: La Municipalidad de Viña del Mar presenta limitaciones en la gestión de su flota institucional debido a procesos manuales y falta de trazabilidad. Para abordar esta problemática se desarrolló FlotUp, un sistema orientado a digitalizar y optimizar la solicitud, asignación y supervisión de viajes municipales. El backend, implementado con Django, Django REST Framework, Django Channels y servicios de Google Maps, incorpora un motor de asignación dinámica capaz de seleccionar vehículos y conductores según disponibilidad, cercanía y compatibilidad de rutas, además de soportar viajes compartidos. Se integró una aplicación móvil en Flutter para los conductores y se diseñó una arquitectura modular con modelos de datos, casos de uso y diagramas UML. La validación evidenció funcionamiento consistente bajo diversas políticas de asignación y escenarios operativos. FlotUp mejora la trazabilidad, eficiencia y transparencia en la administración de la flota municipal.

Palabras Clave: Gestión de flotas, asignación dinámica, viajes compartidos, geolocalización, Google Maps.

1 Introducción

1.1 Contexto institucional.

La Municipalidad de Viña del Mar cuenta con una flota vehicular de aproximadamente 65 vehículos destinados a apoyar las labores administrativas de sus distintos departamentos. Estos vehículos permiten a los funcionarios asistir a reuniones, realizar coordinaciones interdepartamentales, trasladarse a actividades oficiales y ejecutar tareas que son esenciales para la continuidad del servicio público. Bajo este contexto, la flota representa un recurso estratégico dentro del funcionamiento municipal, pues facilita la movilidad necesaria para el cumplimiento eficiente de las responsabilidades institucionales.

La gestión de estos vehículos involucra a diversos actores internos, cada uno con un rol específico dentro del proceso. El funcionario que requiere un traslado comunica su necesidad a la secretaria de su departamento, quien se encarga de registrar la solicitud y transmitirla al director del departamento, quien es el que revisa la pertinencia del desplazamiento y otorga su aprobación. Una vez se valida la solicitud, esta se deriva al

CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título ☐ Tesis de Postgrado

Título del trabajo: Desarrollo de backend para proyecto de optimización de solicitudes de viaje en FlotUp, mediante asignación dinamica y viajes compartidos.

Nombre del candidato(a): Sebastian Antonio Albornoz Diaz

Carrera / Grado: Ingenieria en Informatica

Campus: Sede Viña del Mar **Departamento:** Departamento de electrónica e informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Dagoberto Cabrera Tapia, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución.

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL (marcar una opción)

☒ El trabajo **NO contiene** información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por **(marcar una opción)**:

☐ 6 meses ☐ 12 meses ☐ 2 años ☐ 3 años ☐ 5 años ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 24-02-2026 **Firma:** 

Estudiante o Candidato(a):

Fecha: 22/02/2026 **Firma:** Good

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

administrador de la flota, funcionario responsable de coordinar la disponibilidad de vehículos y conductores, así como de supervisar el correcto uso de los recursos institucionales.

En ausencia de un sistema digital formal, los colaboradores de la municipalidad interactúan a través de comunicaciones informales (principalmente llamados telefónicos o mensajes de texto). Cada viaje se anota en un cuaderno físico, permitiendo llevar un registro básico de las actividades, aunque con limitaciones naturales en cuanto a consulta, orden y trazabilidad. Este método refleja la estructura tradicional con la que se ha gestionado la flota durante años y que, pese a cumplir su función, no responde a las necesidades actuales de coordinación, supervisión y transparencia que demanda una institución de gran tamaño.

Sumado a esto, a lo largo del tiempo también han surgido prácticas irregulares dentro de algunos departamentos. En ciertos casos, algunos vehículos son utilizados como si estuvieran "designados" a áreas específicas, aun cuando la flota municipal es un recurso compartido para toda la institución. Asimismo, algunas secretarias han intentado sus propios métodos de coordinación (como calendarios compartidos) para suplir la falta de una plataforma oficial. También se han registrado situaciones donde ciertos conductores rechazan viajes sin justificación o extienden más de lo permitido sus tiempos de descanso, generando dificultades operativas para el administrador de la flota.

1.2 Problema actual en gestión de flota.

A pesar de la importancia estratégica que tiene la flota vehicular para el funcionamiento municipal, el proceso actual de gestión presenta diversas dificultades que afectan directamente la eficiencia, la coordinación interna y la capacidad de supervisión institucional. El principal problema radica en la falta de un sistema digital centralizado, lo que impide acceder a información consolidada sobre solicitudes, disponibilidad de vehículos y uso histórico de la flota. Esta ausencia de datos estructurados limita la toma de decisiones, dificulta la planificación y restringe la posibilidad de identificar patrones o mejoras operativas.

La dependencia de registros físicos genera un bajo nivel de trazabilidad, ya que la información no puede ser consultada en tiempo real ni comparada de forma sistemática. Esto provoca pérdida de antecedentes, imprecisiones en el seguimiento de los viajes y poca visibilidad sobre el estado actual de los vehículos, lo que reduce la capacidad de supervisión del administrador de flota.

De manera complementaria, se han observado prácticas irregulares que generan inequidades y sobrecarga operativa, como asignaciones preferenciales, rechazos injustificados por parte de algunos conductores o extensiones no autorizadas de tiempos de descanso. Sin instrumentos de control digital, resulta complejo detectar, registrar y corregir este tipo de situaciones.

1.3 Limitaciones del sistema manual.

El sistema manual utilizado para gestionar la flota presenta limitaciones propias de cualquier proceso que depende exclusivamente del trabajo humano y de la coordinación informal. Una de las principales dificultades es la falta de estandarización: cada departamento ejecuta las gestiones de manera ligeramente distinta, lo que genera diferencias en los tiempos de respuesta, en la calidad del registro y en la organización

interna. Esta variabilidad afecta la continuidad del proceso y dificulta mantener un criterio uniforme en la administración del recurso vehicular.

Otra limitación relevante es la incapacidad del modelo manual para generar información estructurada que permita evaluar el funcionamiento de la flota. Al no existir un soporte digital que consolide datos, resulta prácticamente imposible identificar patrones de uso, evaluar niveles de demanda o proyectar necesidades futuras. La ausencia de estos insumos restringe la capacidad de planificación y dificulta la adopción de mejoras basadas en evidencia.

El enfoque manual también limita la capacidad de respuesta ante situaciones dinámicas. Cuando se producen cambios de último minuto, solicitudes simultáneas o imprevistos operativos, el proceso carece de herramientas que permitan reorganizar rápidamente los recursos disponibles. Sin un sistema que centralice la información y facilite la toma de decisiones, la resolución de estos escenarios depende enteramente de la disponibilidad y experiencia del personal, aumentando la carga operativa y reduciendo la eficiencia del flujo de trabajo.

Finalmente, el modelo manual dificulta la continuidad operativa en situaciones donde se requiere trazabilidad o verificación posterior. Los registros dispersos, la falta de automatización y la dependencia de anotaciones aisladas hacen complejo reconstruir el historial de viajes o validar datos con precisión. Esto impacta tanto en la transparencia del proceso como en la capacidad institucional para realizar seguimiento y mejorar su gestión de manera sostenida.

1.4 Necesidad de una solución tecnológica.

Las limitaciones del modelo manual evidencian la necesidad de incorporar herramientas tecnológicas que permitan modernizar la gestión de la flota institucional. En un contexto donde los procesos administrativos requieren cada vez mayor eficiencia y capacidad de respuesta, depender exclusivamente de procedimientos informales y no sistematizados deja a la institución en desventaja frente a las demandas operativas actuales. La ausencia de digitalización impide contar con información confiable, ordenada y disponible en tiempo real, elementos fundamentales para coordinar adecuadamente los recursos vehiculares y asegurar su uso óptimo.

Una solución tecnológica permitiría centralizar los datos y automatizar etapas críticas del proceso, reduciendo la carga operativa asociada a la coordinación manual. Esto facilitaría la toma de decisiones, estandarizaría los flujos de trabajo y otorgaría un soporte más robusto para enfrentar situaciones dinámicas. Al disponer de información actualizada y accesible, el administrador de la flota podría supervisar de forma más efectiva el uso de los vehículos y anticipar necesidades futuras.

Además, la implementación de un sistema digital contribuye directamente a mejorar la transparencia y la trazabilidad del proceso. Al registrar cada solicitud, asignación y desplazamiento de manera estructurada, se fortalece la capacidad de seguimiento y fiscalización, asegurando un uso más equitativo y responsable del recurso municipal. Esto no solo beneficia la gestión interna, sino que también aporta valor institucional al promover prácticas más ordenadas, verificables y alineadas con los estándares modernos de administración pública.

1.5 Propuesta de solución general (SaaS para flota).

Frente a las limitaciones del modelo manual y la necesidad de modernizar la gestión de la flota institucional, se propone el desarrollo de un sistema informático integral orientado a digitalizar y optimizar el proceso de solicitud, asignación y seguimiento de los viajes municipales. La solución se concibe bajo un enfoque Software as a Service (SaaS), permitiendo su implementación inicial en la Municipalidad de Viña del Mar y su eventual escalabilidad hacia otras organizaciones que enfrenten dificultades similares en la administración de recursos vehiculares.

El sistema busca reemplazar los procedimientos informales por una plataforma centralizada en la que todas las etapas del proceso queden registradas de manera digital. A través de esta herramienta, los colaboradores pueden ingresar solicitudes de traslado, mientras que los administradores disponen de información consolidada que facilita la gestión de la flota, la evaluación de solicitudes y la toma de decisiones operativas. Por su parte, los conductores acceden a una interfaz adaptada a sus funciones, recibiendo asignaciones y reportando su avance de forma sencilla y directa.

La propuesta incorpora funcionalidades que permiten mejorar significativamente la organización del proceso, tales como el registro estructurado de solicitudes, la visualización del estado de los viajes, el monitoreo de vehículos mediante servicios de geolocalización y la sugerencia de asignaciones basadas en criterios operativos. Con ello, se busca aumentar la eficiencia del servicio, reducir la carga administrativa y asegurar un uso más transparente y equitativo del recurso vehicular.

1.6 Aporte específico del estudiante (backend, asignación dinámica, viajes compartidos).

Dentro del trabajo colaborativo realizado por el equipo de desarrollo, mi aporte se orienta específicamente al backend del sistema, construyendo la lógica central que permite gestionar las operaciones internas de la plataforma. Para ello, se utiliza el framework Django, mediante el cual se definen los modelos de datos, las reglas de negocio y los servicios que permiten administrar solicitudes de viaje, asignaciones, estados de los recorridos y comunicación entre los distintos perfiles de usuario.

Un componente fundamental del aporte es la implementación del mecanismo de asignación dinámica de vehículos, diseñado para evaluar en tiempo real variables como disponibilidad, cercanía geográfica y características del viaje. Este módulo genera propuestas automáticas que el administrador puede aceptar o ajustar, con el propósito de mejorar la eficiencia operativa y optimizar el uso de la flota institucional.

Asimismo, se desarrolla la lógica inicial para la funcionalidad de viajes compartidos, que permite identificar solicitudes con trayectorias compatibles y evaluar su consolidación en un mismo vehículo cuando ello resulta conveniente. Este enfoque busca reducir recorridos duplicados y promover un uso más eficiente del recurso vehicular.

Finalmente, se implementan las APIs necesarias para la interacción entre la plataforma web y la aplicación móvil utilizada por los conductores, habilitando funciones como el registro y seguimiento de viajes, la actualización de estados y la transmisión de información de geolocalización.

1.7 Objetivo general del proyecto.

Desarrollar e implementar un sistema informático que permita digitalizar, centralizar y optimizar la gestión de la flota vehicular de la Municipalidad de Viña del Mar, mejorando la eficiencia, trazabilidad y control del uso de los recursos institucionales.

1.8 Objetivo general del trabajo del estudiante.

Desarrollar el backend de un sistema de gestión de flotas enfocado en optimizar las solicitudes de viaje mediante asignación dinámica y viajes compartidos.

1.9 Objetivos específicos.

- Digitalizar el registro de solicitudes asegurando trazabilidad y acceso centralizado.
- Implementar un algoritmo de asignación dinámica de vehículos según disponibilidad y cercanía.
- Integrar monitoreo en tiempo real de viajes y vehículos mediante geolocalización.
- Incorporar la funcionalidad de viajes compartidos para reducir costos y recorridos duplicados.
- Establecer un sistema de validación sujeto a la aprobación del administrador de la flota.

1.10 Metodología de trabajo y estructura del documento.

Para el desarrollo del sistema se adopta la metodología ágil SCRUM, seleccionada por su capacidad de organizar el trabajo en iteraciones cortas, facilitar la adaptación a cambios y permitir una retroalimentación continua. La aplicación de este enfoque posibilita mantener un avance progresivo y controlado, dividiendo el proyecto en entregables parciales que se revisan al término de cada sprint.

El plan de trabajo se organiza en tres sprints principales, cada uno enfocado en un conjunto específico de funcionalidades. El primer sprint aborda el núcleo administrativo de la plataforma; el segundo incorpora el flujo de solicitudes desde la perspectiva del colaborador; y el tercero desarrolla los procesos asociados al conductor y el seguimiento operativo de los viajes. Durante todo el proceso se realizan actividades de depuración, pruebas internas y validaciones con los encargados de la gestión de la flota institucional, lo que permite ajustar y mejorar las funcionalidades implementadas.

En cuanto a la estructura del presente documento, el Capítulo 1 expone el contexto institucional, el problema, la propuesta de solución y la metodología empleada. El Capítulo 2 presenta el marco teórico que fundamenta los conceptos, tecnologías y enfoques utilizados en el desarrollo del sistema. El Capítulo 3 detalla el diseño y arquitectura del software, mientras que los capítulos posteriores abordan la implementación, la validación de la solución y las conclusiones del trabajo.

2 Marco teórico.

2.1 Arquitectura web y modelos de programación.

Una arquitectura cliente–servidor es un modelo ampliamente utilizado en el desarrollo de sistemas web. En este esquema existen dos actores principales: el cliente, que realiza solicitudes, y el servidor, que responde a ellas. Ambos se comunican a través de una red, habitualmente Internet. Esta división permite distribuir el trabajo de forma más eficiente: el cliente se encarga de la interacción con el usuario, mientras que el servidor concentra la lógica de negocio y el manejo de los datos. Gracias a esta separación, los sistemas se vuelven más ordenados, fáciles de mantener y capaces de crecer sin perder estabilidad.

En el desarrollo web moderno, esta arquitectura suele complementarse con patrones de diseño que organizan el código dentro del servidor. Django adopta el patrón Modelo–Vista–Template (MVT), una variación del conocido Modelo–Vista–Controlador (MVC). Aunque ambos patrones buscan separar responsabilidades, Django utiliza una nomenclatura particular:

- **Modelo:** representa los datos y cómo se accede a ellos.
- **Vista:** contiene la lógica que recibe la petición del usuario, consulta los modelos cuando es necesario y decide qué información mostrar.
- **Template:** define la presentación final, es decir, cómo se muestran esos datos en la interfaz.

En esencia, Django sigue los principios de MVC, pero renombra algunos elementos para evitar confusiones. Lo que en MVC sería el controlador, Django lo distribuye entre su sistema interno de enrutamiento y las vistas definidas por el desarrollador. Así, la vista en Django actúa como ese "controlador" que decide qué datos entregar, mientras que el template cumple el rol de la vista tradicional encargada de la presentación.

Esta forma de organizar el código tiene un objetivo claro, mantener cada parte del sistema separada pero bien conectada. La lógica de negocio no afecta la presentación y los cambios visuales no modifican el funcionamiento interno. Este desacoplamiento facilita que distintos equipos trabajen simultáneamente en modelos, vistas o plantillas sin interferir entre sí. También permite modificar o extender funcionalidades sin comprometer la estabilidad del sistema.

En contextos empresariales, donde los sistemas suelen mantenerse durante muchos años y se actualizan con frecuencia este tipo de arquitectura resulta especialmente útil. Un cambio en una política de negocio puede implementarse directamente en la vista o el modelo correspondiente, sin alterar la interfaz del sistema. De la misma manera, una optimización en el manejo de datos se realiza exclusivamente en el modelo, manteniendo intactas las capas superiores.

2.2 Frameworks backend.

Un framework backend es una herramienta que facilita el desarrollo del lado del servidor de una aplicación web. En lugar de que el desarrollador programe desde cero aspectos como el manejo de peticiones, la conexión a la base de datos o la seguridad, el framework ofrece una estructura ya preparada y soluciones comunes listas para usar. Esto permite concentrarse en la lógica del proyecto y avanzar más rápido con un código más ordenado.

Dentro de los frameworks backend existen dos enfoques principales:

- Monolíticos o full-stack, que incluyen casi todo lo necesario para construir una aplicación completa.
- Microframeworks, que ofrecen lo mínimo y permiten al desarrollador elegir qué componentes agregar.

Django es un framework monolítico escrito en Python. Nació con la idea de desarrollar aplicaciones rápidamente y con buenas prácticas incorporadas. Su filosofía "batteries included" significa que viene con muchas funcionalidades integradas: un ORM para manejar la base de datos, un sistema de plantillas, autenticación de usuarios, formularios, protecciones de seguridad y un panel de administración automático. Todos estos elementos siguen la misma lógica interna, por lo que encajan entre sí sin configuraciones complejas.

El valor de Django aparece especialmente en proyectos que requieren rapidez de desarrollo, estructura clara y seguridad. A diferencia de microframeworks como Flask o Express, que dan más libertad, pero requieren integrar manualmente muchas herramientas, Django ofrece un entorno cohesionado donde gran parte de las decisiones arquitectónicas ya están resueltas. Esto reduce errores, acelera la construcción del sistema y facilita su mantenimiento a largo plazo.

En términos teóricos, Django se justifica cuando se necesita una base sólida desde el principio: manejar usuarios, acceder a datos de forma segura, generar interfaces de administración o aplicar buenas prácticas sin añadir múltiples dependencias externas. Por esas razones, es común verlo en proyectos empresariales que buscan estabilidad y escalabilidad sin sacrificar el tiempo de desarrollo.

2.3 APIs y comunicación cliente – servidor.

En el desarrollo web, una API REST es una forma de organizar la comunicación entre cliente y servidor usando el protocolo HTTP. La información se presenta como recursos (por ejemplo, usuarios, vehículos o viajes), que se exponen a través de URLs y se manipulan mediante métodos estándar como GET, POST, PUT, PATCH y DELETE. Cada petición incluye todo lo necesario para procesarla (datos, credenciales, parámetros), por lo que el servidor no necesita "recordar" estados previos; esta característica *stateless* favorece la escalabilidad y la claridad del diseño.

En la práctica, las APIs REST suelen trabajar con datos en formato JSON y utilizan los códigos de estado HTTP (200, 201, 404, etc.) para indicar si una operación fue exitosa o no. De este modo, distintos clientes (una app móvil, una aplicación web o incluso otros sistemas) pueden interactuar con los mismos endpoints de forma consistente, sin depender de una interfaz gráfica específica.

En un sistema de gestión de flotas, la API REST cumple el rol de puente entre el backend y las aplicaciones cliente. A través de ella se exponen operaciones como registrar solicitudes de viaje, consultar vehículos disponibles, actualizar estados de recorridos o recuperar historiales, siempre siguiendo las mismas reglas de recursos y métodos HTTP.

Para las funcionalidades de geolocalización, este tipo de API se integra con servicios externos. En este contexto, la Google Maps Platform resulta especialmente relevante. Sus APIs, como Geocoding (para convertir direcciones en coordenadas), Directions (para calcular rutas y tiempos estimados) o Distance Matrix (para estimar múltiples distancias a la vez), permiten traducir datos ingresados por los usuarios en información útil para

la toma de decisiones: ubicación de vehículos, rutas sugeridas, tiempos de viaje, entre otros. Google Maps ofrece además SDKs y bibliotecas que facilitan su integración tanto en el backend como en aplicaciones móviles, lo que reduce la complejidad técnica y mejora la experiencia de uso.

Existen alternativas como OpenStreetMap o Mapbox, que también proveen datos cartográficos y APIs para trabajar con mapas. Sin embargo, en proyectos donde se priorizan la precisión de los datos, la cobertura global, el soporte maduro y una integración directa con herramientas habituales de desarrollo, la elección de Google Maps API se justifica teóricamente como opción principal. Su combinación de calidad cartográfica y servicios avanzados lo convierte en un componente clave para sistemas de gestión de flotas que dependen de rutas confiables y ubicaciones exactas.

2.4 Comunicación en tiempo real.

En muchas aplicaciones actuales, como plataformas de seguimiento, mensajería o paneles administrativos, es necesario mostrar cambios al instante sin que el usuario deba actualizar la página. Para esto se utilizan tecnologías de comunicación en tiempo real, siendo WebSocket la más común en entornos web.

A diferencia del modelo tradicional de HTTP, donde cada petición se cierra después de obtener una respuesta, un WebSocket abre una conexión permanente y bidireccional entre cliente y servidor. Esto permite que ambos intercambien información en cualquier momento, con muy baja latencia. En lugar de que el cliente pregunte continuamente "¿hay algo nuevo?", el servidor envía los datos en cuanto ocurren, lo que hace el uso de recursos más eficiente y la experiencia más fluida.

En un sistema de gestión de flotas, el uso de WebSockets resulta especialmente útil. Por ejemplo, permite actualizar la ubicación de un vehículo en tiempo real sobre un mapa, notificar cambios en el estado de un viaje o mostrar alertas instantáneas al operador. Sin esta tecnología, la única alternativa sería realizar solicitudes periódicas (polling), lo que aumenta la carga del servidor y genera retrasos visibles para el usuario.

En el ecosistema de Django, la comunicación en tiempo real se implementa mediante Django Channels, una extensión oficial del framework que agrega soporte para WebSockets y otras operaciones asíncronas. Channels introduce el concepto de *consumers*, que funcionan de manera similar a las vistas tradicionales, pero orientadas a manejar mensajes en tiempo real. Con esta herramienta, el sistema puede enviar actualizaciones a múltiples clientes de forma simultánea, manteniendo la integración con el ORM y las reglas de seguridad del backend.

2.5 Gestión de datos.

La base de cualquier sistema informático es su modelo de datos, ya que de él dependen la organización, la consistencia y la trazabilidad de la información. En aplicaciones empresariales lo habitual es utilizar una base de datos relacional, donde los datos se estructuran en tablas relacionadas entre sí. Cada tabla representa una entidad (vehículos, usuarios, rutas, etc.), cada fila es un registro concreto y cada columna corresponde a un atributo. Las relaciones entre tablas se establecen mediante claves foráneas, lo que permite evitar duplicación de información y mantener coherencia entre los datos.

El modelo relacional incorpora principios esenciales como la integridad de los datos (garantizar que las referencias sean válidas) y la normalización, que consiste en organizar la información para evitar redundancias y asegurar que cada dato exista en un único lugar. Esto facilita mantener la base de datos limpia, coherente y fácil de actualizar.

En sistemas de flotas también es fundamental la trazabilidad, es decir, la capacidad de seguir la historia de lo que ocurre: qué vehículo se movió, cuándo lo hizo, qué usuario registró un cambio, etc. Esto suele lograrse mediante campos de auditoría o tablas históricas que permiten reconstruir acciones pasadas, un requisito especialmente relevante en contextos institucionales o con responsabilidad administrativa.

Para interactuar con la base de datos desde el código, Django utiliza un ORM (Object-Relational Mapper). Esta herramienta permite trabajar con clases y objetos en lugar de escribir sentencias SQL manuales. El ORM traduce automáticamente las operaciones de alto nivel (crear, consultar, actualizar datos) en instrucciones SQL seguras y optimizadas. Esto simplifica el desarrollo, reduce errores comunes y hace que el código sea más legible y mantenible.

El uso de un ORM tiene varias ventajas, entre ellas, evita escribir SQL repetitivo, previene errores y vulnerabilidades como la inyección SQL, permite cambiar el motor de base de datos sin reescribir la aplicación, y mantiene sincronizados los modelos del código con la estructura real mediante migraciones.

2.6 Tecnología Frontend móvil.

La aplicación móvil destinada a los conductores fue desarrollada utilizando Flutter, un framework multiplataforma creado por Google que permite construir aplicaciones para Android e iOS a partir de un único código fuente. Flutter emplea el lenguaje Dart y un motor de renderizado propio, lo que posibilita interfaces fluidas, consistentes y con un rendimiento cercano al nativo.

Su enfoque basado en widgets y su modelo declarativo facilitan la creación y actualización de pantallas, especialmente en escenarios donde la información cambia con frecuencia, como el seguimiento de viajes o la recepción de nuevas asignaciones.

Flutter resulta adecuado para aplicaciones operativas porque ofrece una experiencia homogénea en distintos dispositivos y se integra con facilidad con APIs como Google Maps y servicios de tiempo real. Gracias a su arquitectura, es posible mantener una sola base de código y asegurar que la aplicación móvil evolucione de forma coherente en todas las plataformas.

2.7 Metodologías agiles.

Para gestionar el desarrollo del proyecto de forma ordenada y adaptable se utilizó Scrum, un marco de trabajo ágil que organiza el proceso en ciclos breves llamados sprints. Cada sprint produce un avance concreto del software y permite incorporar retroalimentación continua, algo especialmente útil cuando los requisitos pueden evolucionar durante el desarrollo.

Scrum define tres roles principales:

- Product Owner: prioriza las funcionalidades y gestiona el Product Backlog, asegurando que el equipo trabaje en lo que genera mayor valor.

- Scrum Master: facilita el proceso, elimina impedimentos y promueve el buen uso de la metodología.
- Equipo de Desarrollo: grupo multidisciplinario que implementa las funcionalidades comprometidas en cada sprint.

El marco establece también una serie de eventos que otorgan cadencia y transparencia al proyecto:
- Planning: se seleccionan los elementos que se desarrollarán durante el sprint y se define su objetivo.
- Daily: reunión breve para coordinar el trabajo y detectar obstáculos.
- Review: se presenta el incremento desarrollado y se recoge feedback de los interesados.
- Retrospective: el equipo analiza qué funcionó bien y qué mejorar de cara al siguiente ciclo.

Los artefactos principales son el Product Backlog (lista priorizada de requisitos), el Sprint Backlog (trabajo comprometido para el sprint) y el Incremento, que representa el resultado funcional alcanzado.

Scrum se fundamenta en tres pilares: transparencia, inspección y adaptación. Esto permite ajustar las prioridades de manera continua, entregar valor desde etapas tempranas y reducir el riesgo de desviaciones importantes. La entrega iterativa facilita mostrar avances frecuentes, incorporar comentarios oportunamente y mantener una visión compartida del progreso.

3 Estado del arte.

3.1 Soluciones de gestión de flota.

La gestión moderna de flotas ha evolucionado significativamente gracias a la telemática, un conjunto de tecnologías que combina GPS, sensores y diagnósticos a bordo para obtener información precisa y en tiempo real sobre cada vehículo. Hoy, estas soluciones permiten a las organizaciones saber dónde están sus unidades, cómo están siendo utilizadas, en qué condiciones operan y qué decisiones pueden tomarse para mejorar la eficiencia. Con ello se abordan desafíos clásicos del sector: optimización de rutas, reducción de costos de combustible, prevención de fallas, mejora en la seguridad de los conductores y cumplimiento de normativas operativas.

GEOTAB: Líder global en telemática y analítica de datos.

Geotab es uno de los nombres más consolidados en la industria. Fundada en Canadá en el año 2000, ha crecido hasta convertirse en uno de los proveedores de telemática más grandes del mundo, con millones de vehículos conectados y billones de puntos de datos procesados continuamente. Su ecosistema combina dispositivos instalados en los vehículos (como la serie GO) con la plataforma en la nube MyGeotab.7

La propuesta de Geotab se centra en un modelo abierto y escalable:
- Rastreo GPS en tiempo real.
- Diagnósticos del motor.
- Detección de hábitos de conducción.
- Analítica avanzada para productividad, seguridad y sostenibilidad.
- Marketplace que permite integrar soluciones de terceros.

Gracias a su enfoque en inteligencia de datos, Geotab ha sido reconocido de forma reiterada por consultoras del sector como uno de los proveedores más innovadores, especialmente por su apoyo a la gestión de flotas eléctricas y sus herramientas de análisis predictivo.

SAMSARA: IoT industrial centrado en seguridad y visibilidad total.

Samsara, fundada en 2015 en Estados Unidos, representa una nueva generación de plataformas de gestión de flotas basada en el Internet de las Cosas (IoT). Su propuesta consiste en integrar una amplia gama de sensores conectados a la nube: cámaras con inteligencia artificial, módulos GPS, sensores de temperatura, dispositivos para remolques y monitores de diagnóstico del motor.

El valor diferencial de Samsara radica en su enfoque unificado: hardware, software y analítica se diseñan como un solo ecosistema. Esto permite funcionalidades como:

- Alertas y reportes automáticos en tiempo real.
- Optimización operacional.
- Cumplimiento normativo.

En los últimos años, la plataforma ha crecido con fuerza en Norteamérica y Europa, posicionándose como una solución integral para organizaciones que buscan una visión completa y continua de sus operaciones.

Otros referentes en la industria.

El sector de gestión de flotas es amplio y competitivo. Entre los actores más relevantes destacan:

- Verizon Connect, su plataforma ofrece seguimiento casi en tiempo real, herramientas de mantenimiento, analítica de comportamiento del conductor y módulos para cumplimiento de normativas.
- Fleet Complete, orientada a empresas que requieren flexibilidad en hardware y visibilidad integral de activos móviles. Incorpora análisis avanzado, reportes configurables y funcionalidades para monitorear hábitos de conducción.
- Webfleet, destaca por su legado en navegación y datos de tráfico. Sus soluciones permiten mejorar la eficiencia del combustible, optimizar rutas y gestionar el rendimiento de los conductores mediante indicadores avanzados.

Otras compañías como Trimble o Teletrac Navman también ofrecen soluciones especializadas para industrias con requerimientos particulares, como transporte pesado, logística o gestión de maquinaria.

Comparación y tendencias actuales.

Al comparar las soluciones actuales de gestión de flotas, se observa que, aunque todas buscan mejorar la visibilidad y la eficiencia operativa, difieren principalmente en su enfoque tecnológico. Geotab prioriza la analítica avanzada y la integración con soluciones externas, mientras que Samsara concentra su propuesta en un ecosistema IoT unificado.

Proveedores como Verizon Connect, Fleet Complete y Webfleet aportan alternativas consolidadas que combinan escalabilidad, capacidades de navegación y distintos niveles de monitoreo según el tipo de flota y las necesidades del cliente.

En términos generales, estas plataformas comparten desafíos comunes, como los costos iniciales de implementación, especialmente cuando se requiere equipamiento especializado, y la curva de aprendizaje derivada de la cantidad de funciones disponibles. A esto se suma la dependencia de la conectividad móvil, que puede limitar el funcionamiento en tiempo real en zonas con cobertura inestable.

3.2 Algoritmos de asignación dinámica.

En transporte y logística, la asignación dinámica corresponde a la capacidad de decidir en tiempo real qué vehículo debe atender cada nueva solicitud, ajustándose continuamente a cambios en la demanda, el tráfico y la disponibilidad de recursos. A diferencia de la planificación estática, donde las rutas o asignaciones se determinan de antemano, este enfoque funciona como un sistema reactivo y adaptativo. Técnicamente, el problema suele representarse como un matching en tiempo real entre vehículos y solicitudes, modelado mediante grafos bipartitos donde cada posible emparejamiento se evalúa según criterios como cercanía, tiempo estimado de llegada o eficiencia global del sistema. Debido a que encontrar una solución óptima bajo estas condiciones es computacionalmente costoso, las plataformas reales utilizan heurísticas, algoritmos aproximados y modelos predictivos capaces de producir buenas soluciones en cuestión de milisegundos.

Los servicios de movilidad bajo demanda como Uber, Lyft y DiDi han demostrado la sofisticación alcanzada por estos algoritmos. Uber, por ejemplo, procesa cada nueva solicitud como un evento que desencadena una reevaluación inmediata de los emparejamientos posibles, ponderando distancia, ETA (Tiempo estimado de llegada) y otros factores relevantes. Aunque no se publica la implementación exacta, se sabe que emplea variaciones del algoritmo húngaro y heurísticas en grafos ponderados para encontrar asignaciones casi óptimas a gran escala. Estas plataformas, además, reevalúan los emparejamientos cuando aparece un conductor más conveniente o cuando cambia el estado del sistema, manteniendo un equilibrio entre eficiencia operativa y estabilidad para el usuario.

La asignación dinámica también incorpora técnicas predictivas. Uber y Lyft emplean modelos de machine learning para estimar dónde y cuándo surgirán picos de demanda, permitiendo sugerencias de reposicionamiento de conductores antes de que ocurran. En China, DiDi ha explorado incluso enfoques de aprendizaje por refuerzo, formulando el problema como un proceso de decisión secuencial que busca maximizar beneficios a largo plazo, como la utilización de los vehículos y la reducción de viajes en vacío. Este tipo de modelos aprende políticas que no solo responden a la situación actual, sino que anticipan la distribución futura de la demanda.

3.3 Viajes compartidos (Carpooling).

El viaje compartido, o *carpooling*, consiste en agrupar a varios pasajeros cuyos trayectos son compatibles para que utilicen un mismo vehículo en lugar de desplazarse por separado. Este modelo busca elevar la ocupación promedio de los autos y reducir así la cantidad total de desplazamientos necesarios. Sus efectos positivos han sido ampliamente documentados: al disminuir el número de vehículos en circulación se reducen la congestión, los tiempos perdidos en tráfico, las emisiones contaminantes y el consumo de combustible. Desde el punto de vista económico, los pasajeros pagan solo una fracción del trayecto, mientras que los conductores distribuyen gastos como combustible o peajes, lo que convierte al carpooling en una alternativa accesible y eficiente.

En la industria han surgido distintos modelos de viajes compartidos. Servicios urbanos como UberPool o Lyft Shared aplican esta lógica en tiempo real, emparejando usuarios que se desplazan en direcciones similares y ofreciendo tarifas reducidas a cambio de leves desvíos en la ruta. Estos sistemas dependen de algoritmos que detectan solicitudes compatibles y evalúan si es posible consolidarlas sin afectar significativamente los tiempos de viaje. Aunque requieren una alta densidad de solicitudes para funcionar con regularidad, en las ciudades donde se han masificado han demostrado potencial para disminuir viajes unipersonales y contribuir a la movilidad sostenible.

Un enfoque distinto es el del carpooling interurbano, representado por plataformas como BlaBlaCar. En este modelo, los viajes se planifican con antelación: los conductores publican su ruta, los pasajeros reservan un asiento y se comparten los costos. La escala alcanzada por este tipo de plataformas ha mostrado impactos concretos, como reducciones masivas de emisiones y ahorros significativos para los usuarios, además de ampliar las oportunidades de movilidad para quienes no disponen de vehículo propio o viven en zonas con menor acceso a transporte público.

Detrás de estos servicios operan algoritmos de agrupación que resuelven versiones complejas del *Dial-a-Ride Problem*, donde deben combinarse múltiples pasajeros bajo restricciones de tiempo, distancia y comodidad. Los sistemas evalúan continuamente qué solicitudes pueden agruparse, considerando desvíos máximos aceptables, ventanas de llegada y la secuencia óptima de recogida y destino. Algunos trabajos en esta área han demostrado que, con modelos de optimización adecuados, es posible compartir una proporción significativa de los viajes urbanos sin perjudicar la calidad del servicio.

4 Diseño y desarrollo de la solución.

4.1 Arquitectura general del sistema.

La solución FlotUp se implementó bajo una arquitectura monolítica utilizando el framework web Django. Esto significa que todos los componentes del servidor (interfaz web, API y lógica de negocio) residen en una misma aplicación desplegable, favoreciendo la simplicidad de despliegue y la integración directa entre funcionalidades. En particular, el backend Django sigue el patrón Modelo-Vista-TEMPLATE (MVT), sirviendo tanto páginas web dinámicas (vistas con plantillas HTML para la interfaz de usuario web) como endpoints de API REST para ser consumidos por clientes externos. Además, el servidor incorpora comunicación en tiempo real mediante WebSockets a través de Django Channels, lo que permite enviar notificaciones instantáneas a los clientes (por ejemplo, actualizaciones de estado de viajes y ubicación de vehículos) sin requerir peticiones constantes desde el cliente. Esta combinación de interfaces (web tradicional, API REST y canales WebSocket) dentro del mismo sistema proporciona flexibilidad para soportar distintos tipos de clientes manteniendo una lógica de negocio unificada.

En cuanto a la estructura interna, la aplicación está organizada de forma modular en múltiples apps de Django que separan las distintas áreas funcionales. Entre las principales se encuentran las apps Accounts, Fleet y Trips, encargadas respectivamente de la gestión de usuarios/organizaciones, gestión de vehículos y conductores, y gestión de solicitudes/viajes (se detallan en la sección 4.3). Cada módulo encapsula modelos de datos, vistas y lógica específicos de su dominio, lo que aporta claridad arquitectónica y facilita la mantenibilidad.

Adicionalmente, FlotUp cuenta con una aplicación móvil desarrollada en Flutter, pensada principalmente para el rol de conductor. Esta app móvil complementa la plataforma web ofreciendo a los choferes una interfaz simplificada en sus dispositivos: muestra únicamente la información de su viaje asignado en curso (datos de ruta, pasajero, horarios) y envía periódicamente la ubicación GPS del vehículo al backend.

Gracias a la API REST del servidor, la aplicación Flutter puede autenticar conductores, obtener sus asignaciones de viaje y reportar eventos en tiempo real. Por su parte, los usuarios colaboradores (funcionarios que solicitan viajes) y administradores de la flota interactúan con el sistema principalmente a través de un frontend web accesible vía navegador, donde pueden registrar solicitudes, monitorear estados y gestionar asignaciones. De esta manera, coexisten dos frontends especializados (web y móvil) que consumen un mismo backend, alineados bajo la arquitectura lógica general del sistema. (Figura 4.1)

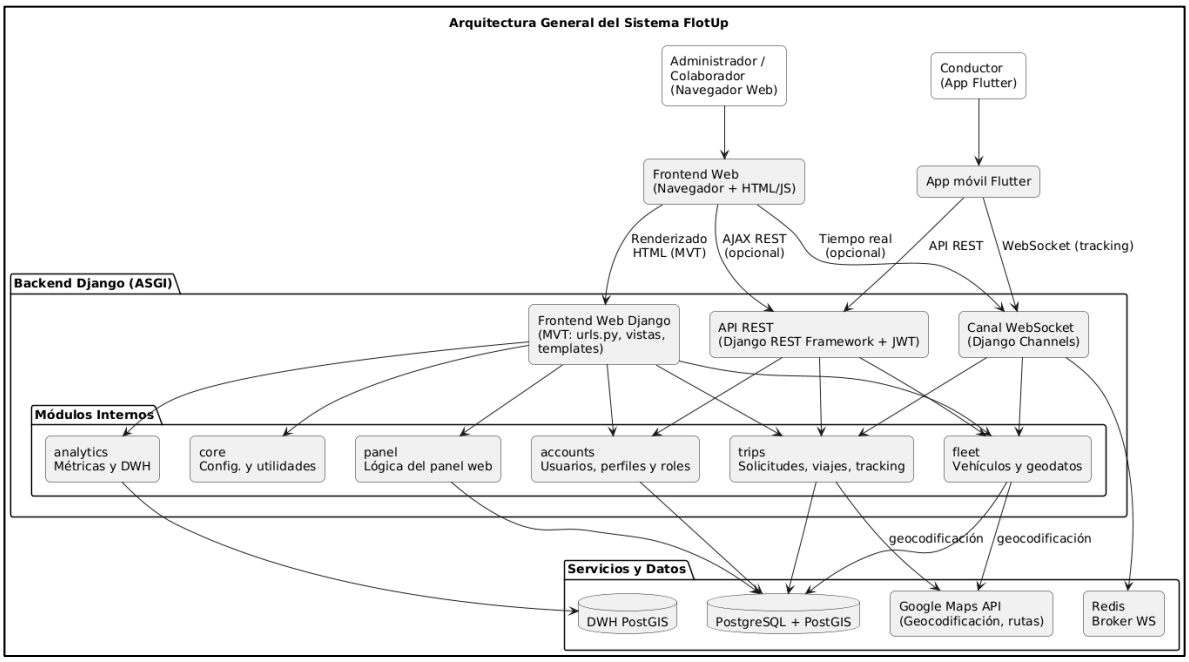

Figura 4.1 – Arquitectura general del sistema FlotUp

En la arquitectura general se aprovechan también diversos servicios externos para ampliar las capacidades del sistema. En particular, se integraron APIs de geolocalización y mapas (Google Maps, OpenStreetMap y Mapbox) para tareas como la obtención de coordenadas a partir de direcciones, el cálculo de distancias y tiempos estimados de recorrido, y la visualización de mapas con rutas. Estos servicios permiten, por ejemplo, determinar el vehículo municipal más cercano al punto de origen de una solicitud, trazar la ruta óptima para un viaje compartido o mostrar en el mapa la posición de los vehículos en tiempo real.

Todos estos cálculos se realizan desde el backend Django mediante peticiones a las APIs externas, cuyos resultados luego se almacenan o transmiten a los clientes según corresponda.

4.2 Diagrama de componentes.

El diseño de FlotUp puede entenderse también en términos de sus componentes principales y las interacciones entre ellos, tal como se representa en el diagrama de componentes (Figura X). Los componentes identificados incluyen: (1) Frontend web Django, (2) Backend/API Django, (3) Base de Datos, (4) Aplicación móvil Flutter y (5) Servicios externos de mapas/geolocalización. A continuación, se explica el rol de cada componente, sus responsabilidades y la forma en que se comunican para conformar el sistema completo.

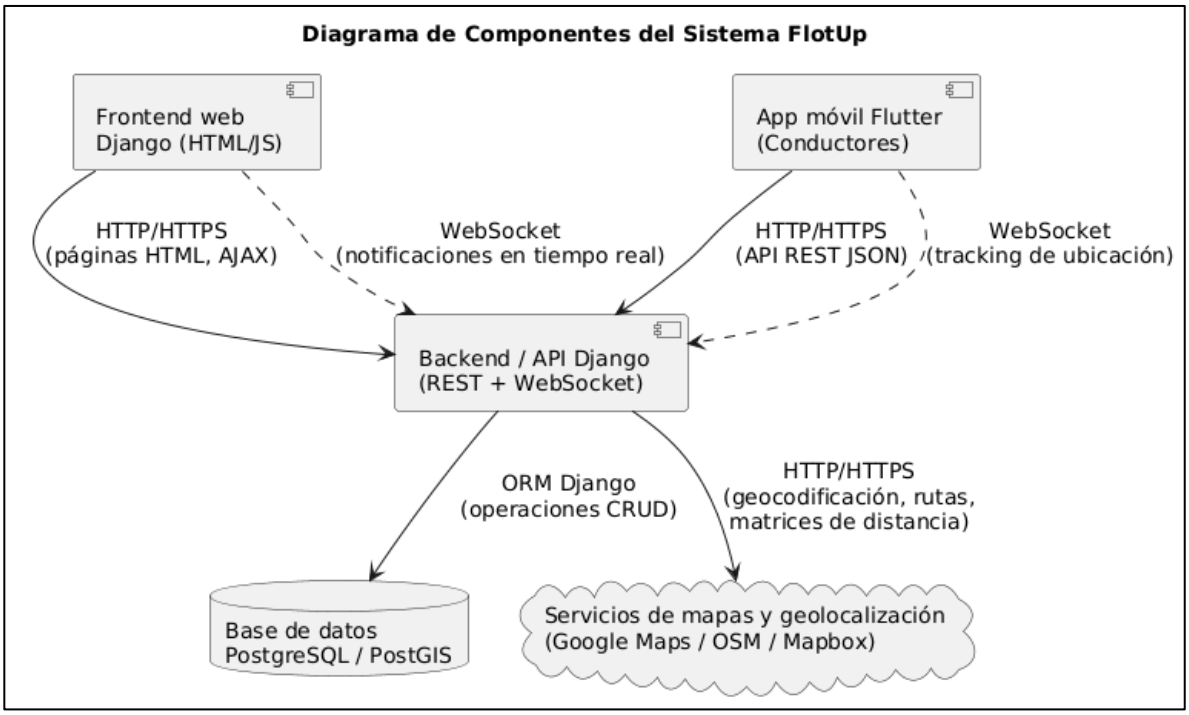

Figura 4.2 – Diagrama de componentes sistema FlotUp

Frontend web (Django): Corresponde a la interfaz de usuario accesible vía navegador para los roles de administrador de flota y colaborador solicitante. Este frontend está servido por el propio backend Django (usando sus vistas y plantillas) y ofrece funcionalidades como formularios para ingresar nuevas solicitudes de viaje, pantallas de consulta del estado de solicitudes, y un panel de administración de flota.

Su responsabilidad principal es presentar la información al usuario de forma amigable y permitirle interactuar con el sistema (por ejemplo, un colaborador puede ver si su solicitud fue aceptada o rechazada, y un administrador visualizar sugerencias de asignación de vehículos de manera gráfica).

Aunque forme parte de la misma aplicación Django, se le considera un componente separado lógicamente, ya que se encarga de la capa de presentación web. Este frontend web se comunica con el backend de dos formas: mediante solicitudes HTTP tradicionales (por ejemplo, cuando un usuario carga una página o envía un formulario) y mediante una conexión WebSocket para recibir notificaciones en vivo (por ejemplo, actualización instantánea en el navegador cuando un viaje es asignado o cuando un vehículo cambia de estado).

Backend / API (Django): Es el núcleo central del sistema, que expone tanto la lógica de negocio como los datos a los otros componentes. Por un lado, actúa como servidor web tradicional para atender las vistas del frontend web Django; por otro, expone una API RESTful que es consumida por la aplicación móvil Flutter.

Este backend implementa todos los procesos fundamentales: registro de solicitudes, algoritmo de asignación dinámica de vehículos, consolidación de viajes compartidos, gestión de usuarios y autenticación, etc. También mantiene la conexión con la base de datos para leer/guardar información y se integra con los servicios de mapas cuando necesita cálculos de rutas o geocodificación.

En la comunicación interna, el backend recibe peticiones del frontend web y de la app móvil, las procesa y retorna las respuestas o emisiones correspondientes. Adicionalmente, es el backend quien "publica" eventos en tiempo real: por ejemplo, al aprobar un viaje, envía mediante WebSocket una notificación al navegador del colaborador para indicarle que su solicitud fue aprobada, o difunde a los paneles de administrador la nueva ubicación de un vehículo cada vez que la recibe de la app móvil del conductor.

Base de datos: El almacenamiento persistente del sistema se maneja a través de una base de datos relacional central PostgreSQL. En ella residen todas las entidades definidas en el modelo de datos: usuarios, organizaciones, vehículos, solicitudes de viaje, viajes asignados, etc.

La base de datos es accedida únicamente por el backend Django, el cual mediante el ORM de Django realiza las operaciones CRUD necesarias. De esta manera, la base de datos no expone directamente información a ningún cliente, sino que el control de acceso se implementa a través del backend (siguiendo las reglas de roles y permisos definidas).

Esto garantiza una separación clara: el backend actúa como mediador y garante de seguridad entre los datos y los frontends. En el diagrama de componentes, la base de datos aparece como un componente aislado al cual solo el backend tiene acceso directo.

Cabe mencionar que, bajo el modelo SaaS multi-tenant adoptado, la base de datos aloja los datos de múltiples organizaciones cliente, pero gracias a la capa de acceso del backend, cada organización funciona como una partición lógica cuyos datos están aislados y solo son accesibles para usuarios pertenecientes a esa organización.

Aplicación móvil (Flutter): Es el cliente móvil utilizado principalmente por los conductores. Implementado en Flutter (lo que permite desplegarlo en dispositivos Android/iOS), este componente ofrece una interfaz simplificada adecuada para usar mientras se conduce.

A través de la app, el chofer recibe notificaciones de nuevos viajes asignados, puede visualizar detalles del viaje (punto de partida, destino, pasajeros, horario) y marcar cambios de estado (por ejemplo, iniciar o finalizar el viaje). La app móvil se comunica con el backend exclusivamente mediante la API REST (consumiendo endpoints para autenticarse, obtener sus viajes pendientes/en curso, enviar actualizaciones de estado, etc.).

En la práctica, el envío periódico de la posición GPS del vehículo se realiza desde la app móvil hacia el backend, que a su vez actualiza esta información en el sistema y la retransmite en tiempo real a los administradores a través del canal WebSocket.

Es importante destacar que la app móvil no accede directamente a la base de datos ni a servicios externos de mapas; solicita toda la información necesaria al backend, manteniendo así la coherencia y seguridad de los datos.

Servicios de mapas y geolocalización: Son componentes externos al sistema (proveídos por terceros) con los que el backend interactúa cuando requiere funcionalidades geoespaciales avanzadas.

FlotUp integra APIs de proveedores como Google Maps, OpenStreetMap y Mapbox para distintas finalidades: conversión de direcciones a coordenadas geográficas (geocodificación), estimación de distancias y tiempos entre ubicaciones, obtención de rutas óptimas, e incluso visualización de mapas (por medio de tiles o mapas embebidos en el frontend web).

Estos servicios se consumen mediante peticiones HTTP salientes desde el backend. Por ejemplo, al registrar una nueva solicitud de viaje, el sistema puede invocar la API de Google Maps para calcular la distancia desde el punto de origen del solicitante hasta todos los vehículos disponibles, determinando así cuál está más cercano.

Los resultados de estas llamadas se emplean en la lógica de negocio y también se almacenan o se muestran al usuario en la interfaz. Dado que estos servicios son externos, el diseño del sistema los encapsula en este componente separado, de modo que el resto del sistema simplemente solicita funcionalidades de mapas al backend y es este el que se encarga de obtenerlas del proveedor correspondiente.

En conjunto, la comunicación entre los componentes sigue una estructura cliente-servidor tradicional enriquecida con comunicación push desde el servidor: los frontends (web y móvil) actúan como clientes que envían solicitudes al backend y reciben respuestas; el backend actúa como servidor central que procesa lógica y consulta datos; y los servicios de mapas funcionan como APIs auxiliares a las que el servidor consulta según necesidad.

4.3 Diagrama de módulos y organización interna.

Internamente, el backend monolítico de Django se estructuró en varios módulos (apps Django) para dividir el dominio de la aplicación en sub-sistemas manejables. En la Figura 4.3 se presenta un diagrama de módulos donde se destacan las apps: Accounts, Fleet, Trips, Core, Analytics y Panel.

Accounts: Este módulo gestiona todo lo relativo a los usuarios, autenticación y organizaciones. Incluye modelos para representar a los usuarios del sistema (credenciales, datos personales básicos) y perfiles asociados con información adicional como el rol que cumplen (por ejemplo, administrador, conductor o colaborador).

También administra las organizaciones (por ejemplo, distintas municipalidades o departamentos), cada usuario pertenece a una organización y todos los datos que genera o visualiza están ligados a esta. Las funcionalidades clave de Accounts abarcan el registro/inicio de sesión de usuarios, la definición de roles y permisos y la administración de las relaciones entre usuarios y organizaciones.

Al encapsularse en su propio módulo, Accounts permite manejar fácilmente la expansión a nuevas organizaciones cliente y reforzar la seguridad, ya que las políticas de acceso por rol se implementan centralizadamente aquí. Por ejemplo, el módulo define que un

conductor solo puede acceder (vía API) a los endpoints de consultar sus viajes y actualizar su estado, mientras que un administrador tiene permisos para ver y modificar recursos de flota de su organización, etc.

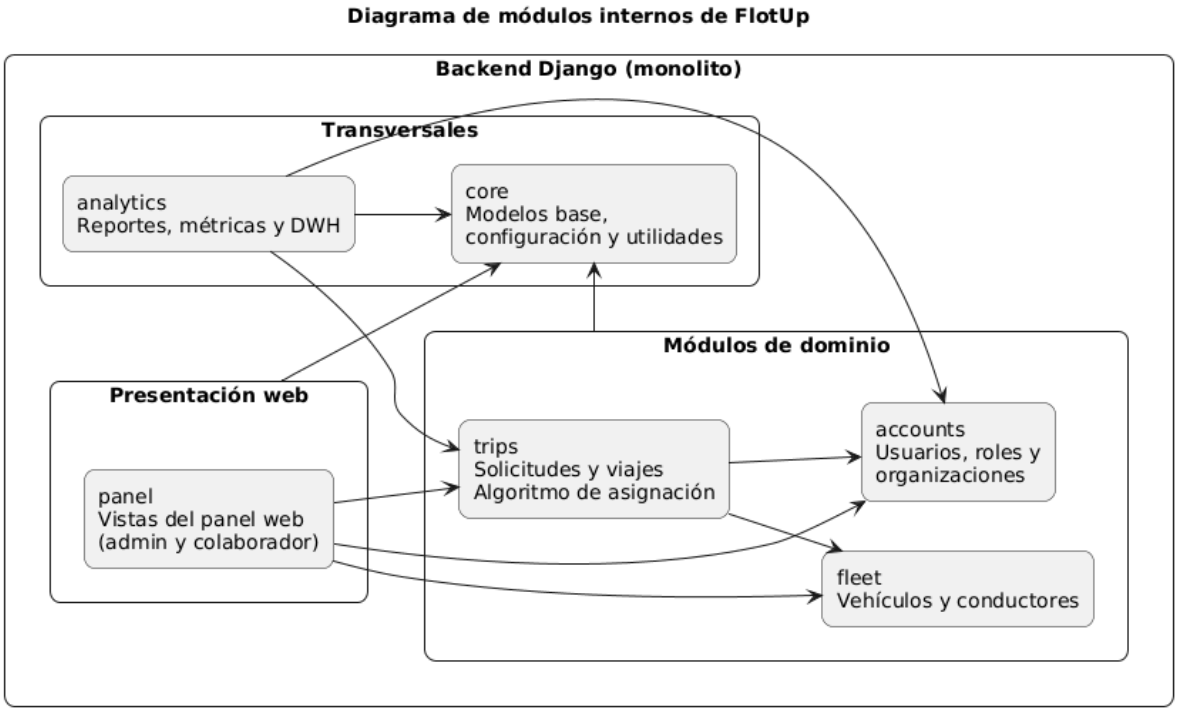

Figura 4.3 – Diagrama módulos internos de la solución.

Fleet: Este módulo se encarga de la gestión de la flota vehicular y los conductores. Define el modelo Vehículo con sus atributos y relaciona cada vehículo con su organización propietaria. También maneja la información de conductores asignados a vehículos: en algunas organizaciones un vehículo puede tener un chofer designado permanente, mientras que en otras los conductores rotan; para soportar ambas modalidades, el sistema incluye entidades que vinculan vehículos con conductores (por ejemplo, un modelo de asignación o binding vehículo-conductor con fechas de vigencia, permitiendo saber qué chofer está asignado a cuál vehículo en cada momento).

El módulo Fleet también gestiona la disponibilidad de los vehículos mediante atributos o submodelos: por ejemplo, se contemplan estados como disponible, en viaje, fuera de servicio, etc., e incluso se pueden registrar bloqueos de flota (mantenimiento, reservas especiales) para que ciertos vehículos no se consideren disponibles en el algoritmo de asignación durante determinados periodos.

Trips: Es el módulo central para la gestión de solicitudes de viaje y viajes asignados. Contiene el modelo TravelRequest (solicitud de viaje) que registra cada pedido de transporte realizado por un colaborador. Asociado a este, el módulo maneja el flujo de aprobación y asignación: cuando una solicitud es aprobada por un administrador, se crea un Trip que representa el traslado programado, vinculando la solicitud con un vehículo y un conductor asignados.

El modelo Trip almacena información como el vehículo utilizado, el chofer que realizará el servicio (referenciando al perfil del conductor), la hora de salida y llegada estimadas

y finalmente los tiempos reales registrados, así como el estado actual del viaje (asignado, en curso, finalizado, cancelado).

Dentro de Trips reside la implementación del algoritmo de asignación dinámica, el cual al aprobar una solicitud puede sugerir automáticamente el vehículo óptimo considerando criterios en tiempo real (por ejemplo, el más cercano al origen y que esté disponible).

También en este módulo se soporta la lógica de viajes compartidos: el sistema es capaz de detectar cuando dos o más solicitudes tienen rutas similares o destinos cercanos y ventanas de tiempo compatibles, para proponer su combinación en un solo viaje compartido. Para ello, el modelo de datos de Trips incluye la posibilidad de definir múltiples paradas o puntos de destino en un mismo viaje. De este modo, un Trip compartido contendría las referencias a todas las solicitudes atendidas en ese recorrido, garantizando que se mantenga la trazabilidad de cuál funcionario fue transportado y a dónde.

El módulo Trips también registra el historial de cambios de estado de cada solicitud/viaje, de forma que existe trazabilidad completa desde que el funcionario crea la solicitud, pasando por la aprobación/asignación, hasta la finalización del viaje.

Core: Además de los módulos de dominio mencionados, el sistema incluye un módulo Core que agrupa componentes compartidos y utilitarios transversales a la aplicación. Al centralizar estos elementos en Core, se evita la duplicación de código y se asegura consistencia en funcionalidades usadas por múltiples módulos. Este módulo actúa como el fundamento común sobre el cual los módulos Accounts, Fleet y Trips construyen sus funcionalidades específicas.

Analytics: Si bien en la primera versión de FlotUp la analítica podría ser básica, se prevee un módulo Analytics destinado a manejar la generación de reportes y estadísticas sobre el uso de la flota. Este módulo se encarga de procesar los datos históricos de viajes, calculando indicadores clave como número de viajes por mes, kilómetros recorridos, tasa de ocupación de vehículos, tiempos promedio de respuesta a solicitudes, entre otros.

Panel: Finalmente, el módulo Panel hace referencia a la capa de presentación web personalizada de la aplicación (diferente del panel de administración genérico de Django). Aquí residen las vistas, plantillas y archivos estáticos que conforman la experiencia de usuario para colaboradores y administradores cuando interactúan vía navegador.

Básicamente, Panel orquesta las páginas web del frontend, componiendo información proveniente de Accounts, Fleet y Trips. También posibilita, en caso de ser requerido, sustituir o complementar la interfaz web en el futuro (por ejemplo, migrar a un frontend JavaScript separado) manteniendo de referencia esta implementación base.

4.4 Modelo de datos.

El modelo de datos de FlotUp se estructuró en tres dominios centrales, directamente involucrados en los procesos esenciales del sistema: la gestión de usuarios y organizaciones (accounts), la administración vehicular (fleet) y la gestión de solicitudes y viajes (trips). Estos módulos conforman el núcleo funcional de la aplicación y permiten implementar las capacidades clave del sistema, tales como la asignación automática de vehículos, el seguimiento operativo de viajes y la trazabilidad de la flota municipal.

Si bien el sistema incluye otras aplicaciones internas, sus modelos cumplen funciones de apoyo (como utilidades, vistas administrativas o exportación de datos) y no inciden directamente en la lógica operativa ni en los algoritmos de asignación. Por ello, la descripción detallada de esta sección se concentra exclusivamente en los tres módulos destacados.

Modelo Accounts: usuarios, perfiles y organizaciones.

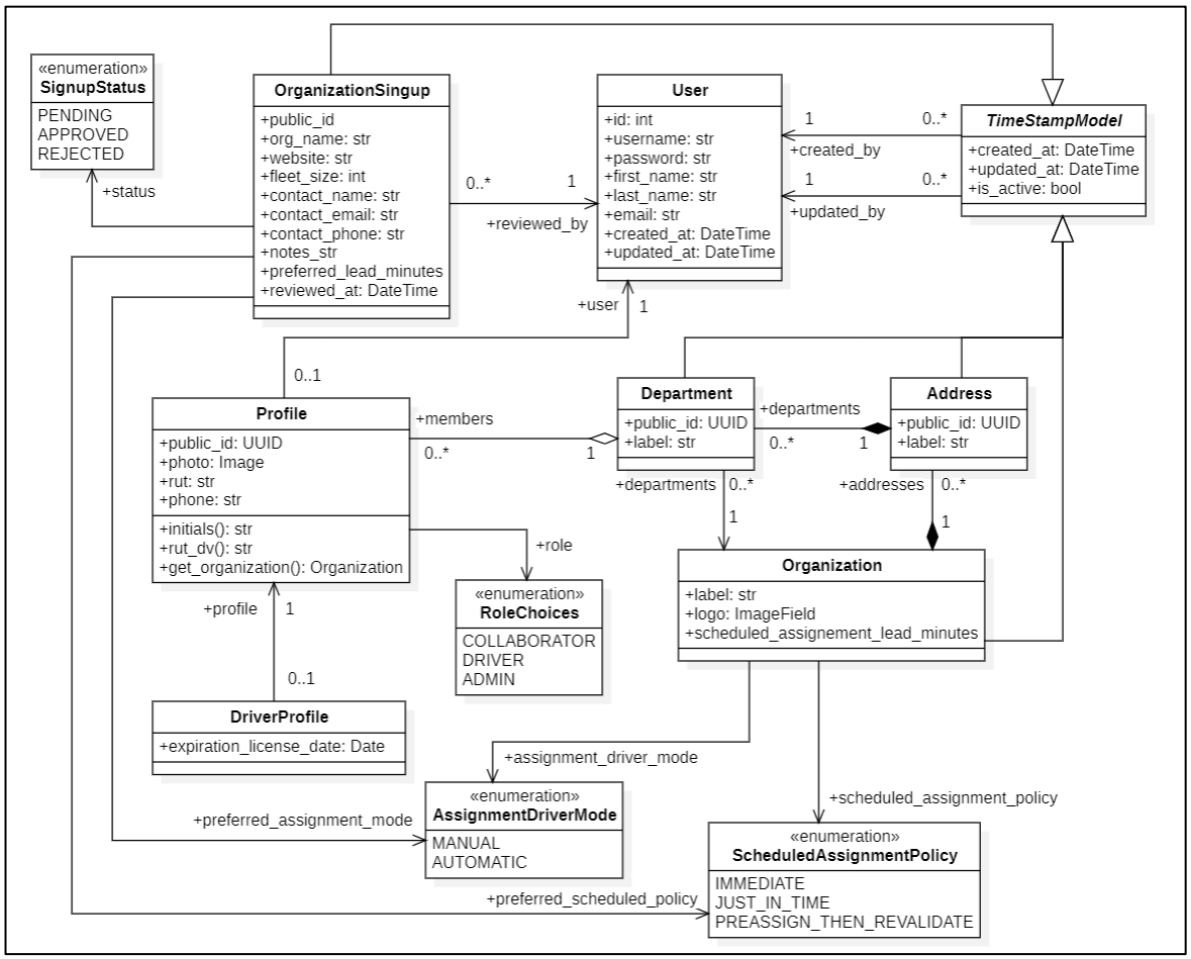

Figura 4.4 – Diagrama ER modulo 'accounts'.

El módulo accounts representa la capa organizacional y de identificación de usuarios. En el centro se encuentra la entidad User, que almacena credenciales y metadatos básicos, y se apoya en el modelo común TimeStampModel para manejo de auditoría. Cada usuario pertenece a una Organization, la cual constituye el eje del modelo multi-tenant del sistema.

La organización incluye atributos como nombre, imagen corporativa y parámetros operativos (por ejemplo, scheduled_assignment_lead_minutes), que luego influyen directamente en el comportamiento del algoritmo de asignación automática. Una organización puede tener además múltiples Department y Address, utilizados para jerarquizar y clasificar usuarios de una organización.

El usuario se complementa con un Profile, en relación uno-a-uno. Este contiene información personal adicional y define explícitamente el rol del usuario, representado

mediante la enumeración RoleChoices (ADMIN, COLLABORATOR, DRIVER). En caso de ser conductor, el perfil se especializa a través del modelo DriverProfile, que agrega atributos específicos.

La figura de OrganizationSignup modela el proceso de registro de nuevas organizaciones, incluyendo su estado (SignupStatus) y los metadatos del proceso de revisión.

Finalmente, las políticas de asignación preferidas por cada conductor se modelan mediante los campos preferred_assignment_mode y preferred_scheduled_policy, que referencian directamente las enumeraciones AssignmentDriverMode y ScheduledAssignmentPolicy. Este vínculo permite que el sistema considere preferencias individuales al momento de realizar asignaciones automáticas o manuales.

Modelo fleet: vehículos y asignaciones.

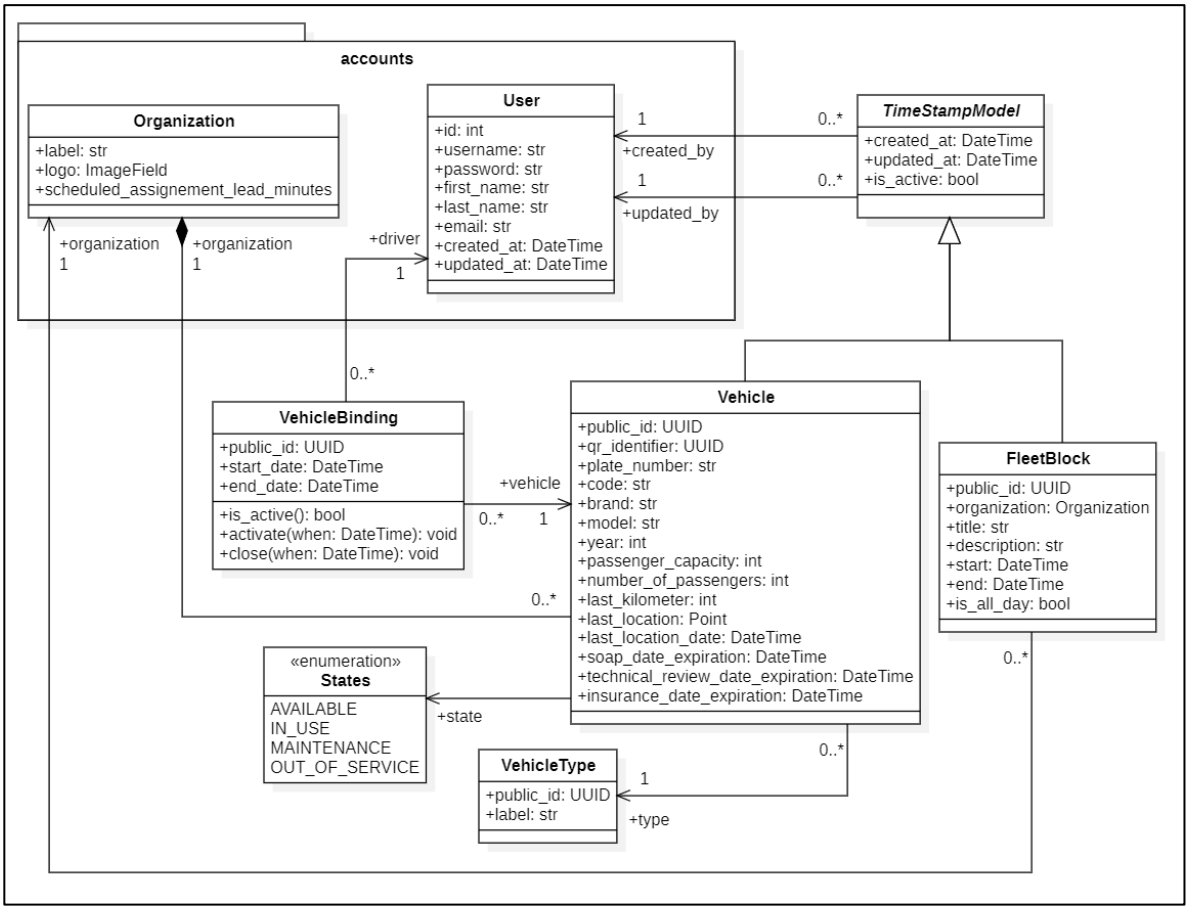

Figura 4.5 – Diagrama ER modulo 'fleet'.

El dominio fleet almacena todo lo relativo a la flota vehicular, su estado, disponibilidad y relación con los conductores.

Su entidad principal es Vehicle, que contiene información técnica (marca, modelo, año, identificadores), legal (fechas de SOAP, revisión técnica, seguro) y operacional (capacidad, ubicación actual, kilometraje). Cada vehículo tiene un VehicleType, lo que permite clasificar la flota en categorías.

La disponibilidad del conductor respecto del vehículo se modela mediante VehicleBinding, que representa asignaciones temporales o permanentes. Esta entidad incluye fechas de vigencia y un método de activación/cierre, permitiendo mantener un historial de qué conductor estuvo autorizado para cada vehículo en un momento dado.

El estado operacional del vehículo se representa mediante la enumeración States, que permite filtrar rápidamente si un vehículo está AVAILABLE, IN_USE, en MANTENANCE o OUT_OF_SERVICE.

El módulo incorpora además la entidad FleetBlock, que registra bloqueos específicos en rangos de tiempo, utilizados para impedir la asignación de vehículos que, aunque no estén en viaje, se encuentran reservados o en mantenimiento.

Modelo Trips: solicitudes, viajes, paradas y asignación dinámica.

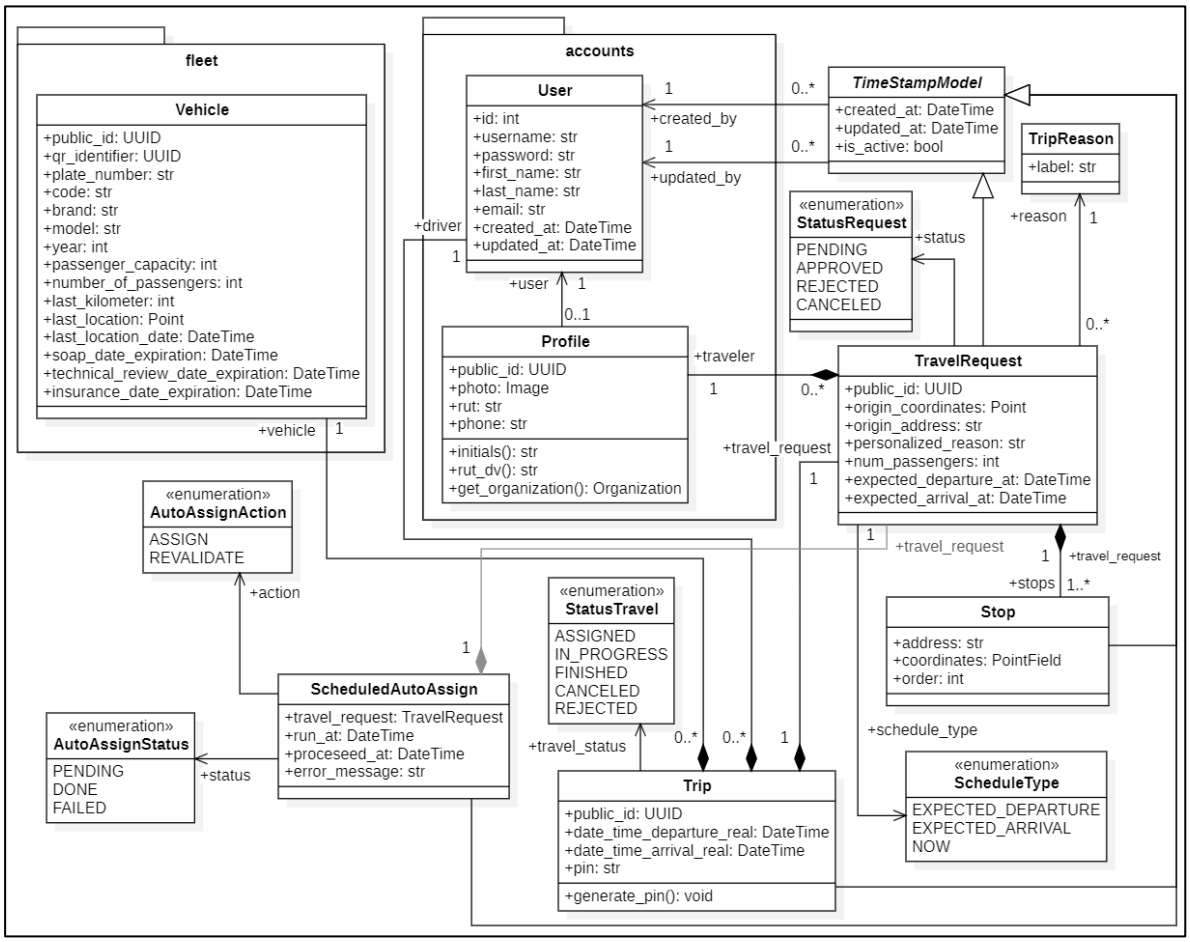

Figura 4.6 – Diagrama ER modulo 'trips'.

El módulo trips modela todo el ciclo operacional de los viajes.

La entidad TravelRequest representa la solicitud creada por un colaborador. Incluye: origen/destino tanto en texto como en coordenadas (permitiendo integración con Google Maps), motivo del viaje (referencia a TripReason), número de pasajeros, tipo de programación (ScheduleType), estado (StatusRequest).

Una TravelRequest puede incluir múltiples Stop, permitiendo la construcción de rutas con varias paradas (clave para viajes compartidos).

Una vez aprobada, la solicitud se transforma en un Trip, que registra: el vehículo asignado, el conductor, horarios estimados y reales, PIN de viaje para validación, estado del viaje (StatusTravel).

Cada Trip proviene de una TravelRequest, pero puede asociarse a varias mediante sus paradas, permitiendo carpooling.

Para soportar la asignación automática, el dominio incorpora dos elementos: AutoAssignAction, el cual indica la acción ejecutada por el algoritmo (ASSIGN o REVALIDATE). ScheduledAutoAssign, el cual permite programar asignaciones futuras según la política de la organización (IMMEDIATE, JUST_IN_TIME, PREASSIGN_THEN_REVALIDATE), registrando estado (AutoAssignStatus) y resultado.

Las relaciones del dominio muestran claramente cómo los diferentes elementos se entrelazan: TravelRequest se vincula con su solicitante (Profile), Trip se vincula con Vehicle y con DriverProfile a través de VehicleBinding, y todas estas entidades heredan trazabilidad mediante TimeStampModel.

4.5 Casos de uso.

En el diseño de la solución FlotUp se definió un conjunto de casos de uso para capturar las interacciones clave entre los tres roles principales del sistema, el Funcionario solicitante, el Administrador de flota y Conductor, y así delinear las funcionalidades requeridas. Cada caso de uso describe un escenario concreto de cómo los usuarios y el sistema interactúan para lograr un objetivo específico, sirviendo como puente entre los requisitos funcionales y la implementación.

Para facilitar su análisis, los 23 casos de uso identificados se han agrupado en cinco módulos funcionales: Gestión de Viajes, Gestión de Flota, Reportes y Supervisión, Priorización Inteligente, y Evaluación e Incidencias. Esta organización modular refleja las áreas funcionales del sistema y permite estructurar el diseño en subsistemas lógicos, manteniendo la cohesión interna de cada conjunto de funcionalidades.

Módulos funcionales y Diagramas de casos de uso.

Gestión de Viajes: Este módulo agrupa los casos de uso relacionados con el ciclo de vida completo de una solicitud de traslado, desde su creación hasta la asignación inicial del viaje. En el diagrama de casos de uso de Gestión de Viajes interactúan principalmente el Funcionario solicitante y el Administrador.

El Funcionario inicia el proceso al registrar una solicitud de viaje en la plataforma. El Administrador, por su parte, revisa las solicitudes pendientes y decide su curso de acción: aprobar aquellas procedentes y rechazar o reprogramar las que no cumplen los criterios institucionales. Una vez aprobada una solicitud, el Administrador procede a asignar un vehículo y un chofer para el traslado, actividad en la que el sistema puede asistir sugiriendo opciones óptimas.

De esta manera, el módulo de Gestión de Viajes encapsula la interacción central entre quien solicita el viaje y quien administra los recursos para concretarlo, estableciendo el punto de partida del flujo operativo.

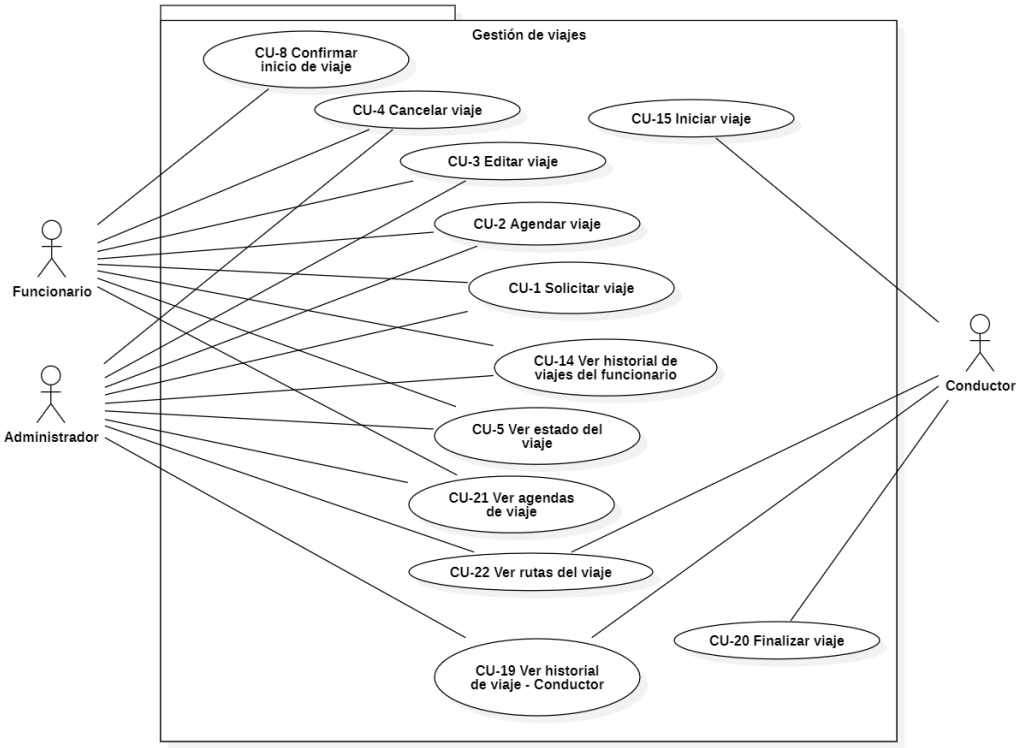

Figura 4.7 – Diagrama casos de uso "Gestión de viajes"

Gestión de Flota: En este módulo se concentran los casos de uso orientados a la administración de los vehículos y recursos asociados a la flota municipal. El rol protagonista aquí es el Administrador de flota, encargado de mantener actualizado el inventario de vehículos y sus características.

Las funcionalidades incluyen registrar nuevos vehículos en el sistema, editar su información relevante y gestionar la disponibilidad de cada unidad. Esto último implica marcar vehículos como disponibles, en mantenimiento o fuera de servicio según corresponda.

Aunque el Funcionario y el Conductor no participan directamente en los casos de uso de Gestión de Flota, este módulo impacta indirectamente a todos los roles: una administración adecuada de la flota garantiza que, al generarse una solicitud de viaje, haya vehículos idóneos para asignar, y que los conductores dispongan de unidades en buen estado.

El diagrama de casos de uso de este módulo muestra exclusivamente al Administrador interactuando con las funciones de alta, baja y actualización de vehículos, reflejando cómo se cuida la infraestructura base sobre la cual opera el sistema de viajes.

Evaluación e Incidencias: Este módulo agrupa las funcionalidades posteriores a la culminación de cada viaje, asegurando el feedback y el registro de eventos inesperados. Intervienen aquí tanto el Funcionario como el Administrador (incluso el Conductor, en ciertos casos), dado que abarca la evaluación del servicio y la gestión de incidencias ocurridas.

Una vez que un viaje finaliza, el Funcionario que solicitó el traslado puede calificar la experiencia mediante una breve evaluación en la plataforma. Esta retroalimentación se

registra como parte de los casos de uso de evaluación y puede ser consultada por el Administrador para fines de mejora continua.
Por otro lado, si durante el viaje ocurrió alguna anomalía (por ejemplo, un accidente menor, un retraso significativo, o un incumplimiento de protocolo) entra en juego la funcionalidad de incidencias: tanto el Conductor como el Funcionario tienen la posibilidad de reportar una incidencia, la cual genera una alerta para el Administrador.

El Administrador entonces registra y atiende la incidencia, documentando su resolución (por ejemplo, coordinar asistencia mecánica, reprogramar el viaje, o tomar medidas disciplinarias si corresponde).

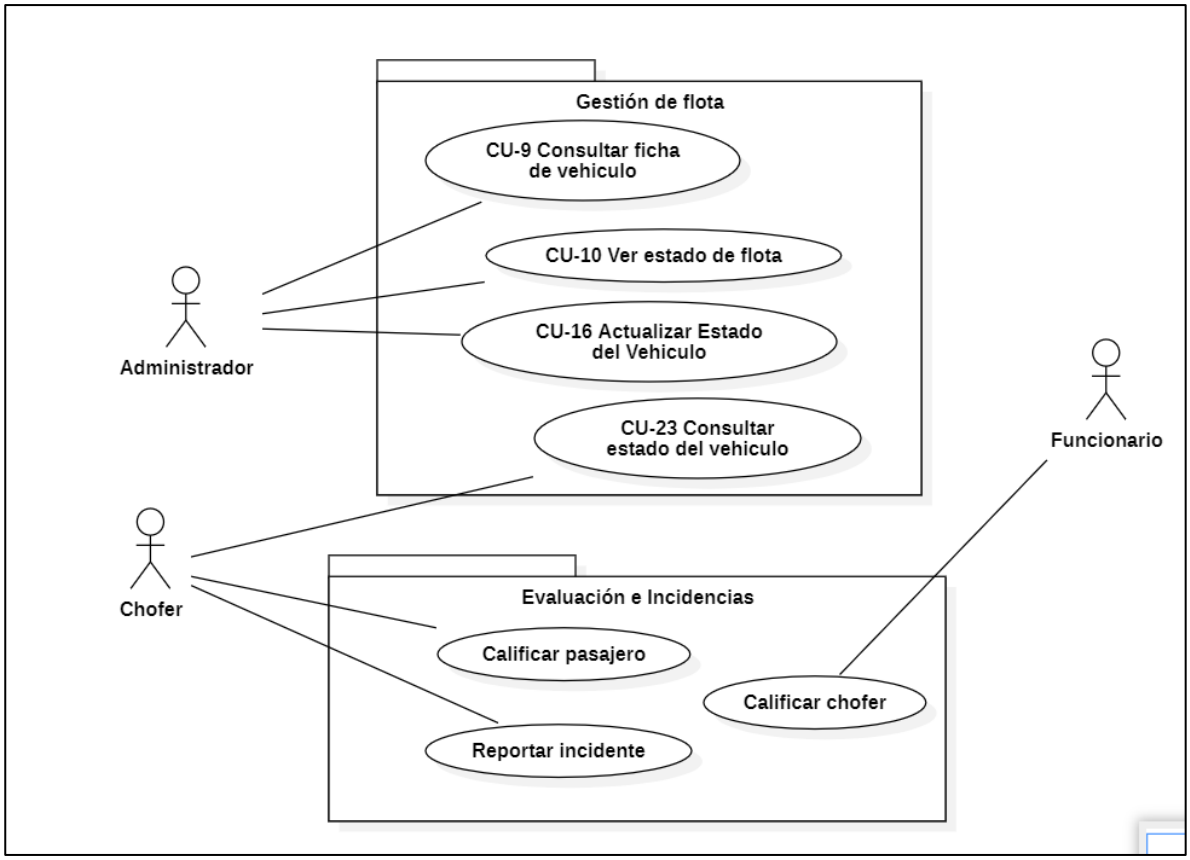

Figura 4.8 – Diagrama casos de uso "Gestión de flota" y "Evaluación e incidencias"

Reportes y Supervisión: El cuarto módulo funcional engloba las capacidades de monitoreo en tiempo real y de análisis posterior de la operación. En su diagrama de caso de uso, el Administrador vuelve a ser el actor principal, pues es quien necesita supervisar los viajes en curso y generar reportes sobre la utilización de la flota.

A través de estas funcionalidades, el Administrador puede visualizar en un mapa los vehículos activos, con sus ubicaciones actualizadas constantemente, así como el estado de cada viaje. Esta supervisión en tiempo real permite detectar desviaciones o eventualidades durante la ejecución de los traslados y tomar decisiones informadas al instante.

Adicionalmente, el módulo incluye casos de uso para emitir reportes estadísticos o informes periódicos sobre la gestión de la flota. Estas capacidades de reporte apoyan la

fiscalización y la toma de decisiones estratégicas, proporcionando datos objetivos para evaluar la eficiencia del sistema.
Priorización Inteligente: Este módulo funcional se distingue por incorporar la lógica de asignación dinámica y las sugerencias automatizadas de viajes compartidos. Aquí el sistema en sí actúa como protagonista, en apoyo del Administrador.

Los casos de uso de Priorización Inteligente se activan típicamente cuando el Administrador va a asignar un vehículo a una solicitud, el sistema calcula la mejor asignación considerando múltiples variables en tiempo real.

Aunque la decisión final recae siempre en el Administrador (quien puede aceptar o ajustar la sugerencia, la Priorización Inteligente agiliza la toma de decisiones y promueve un uso más eficiente de la flota.

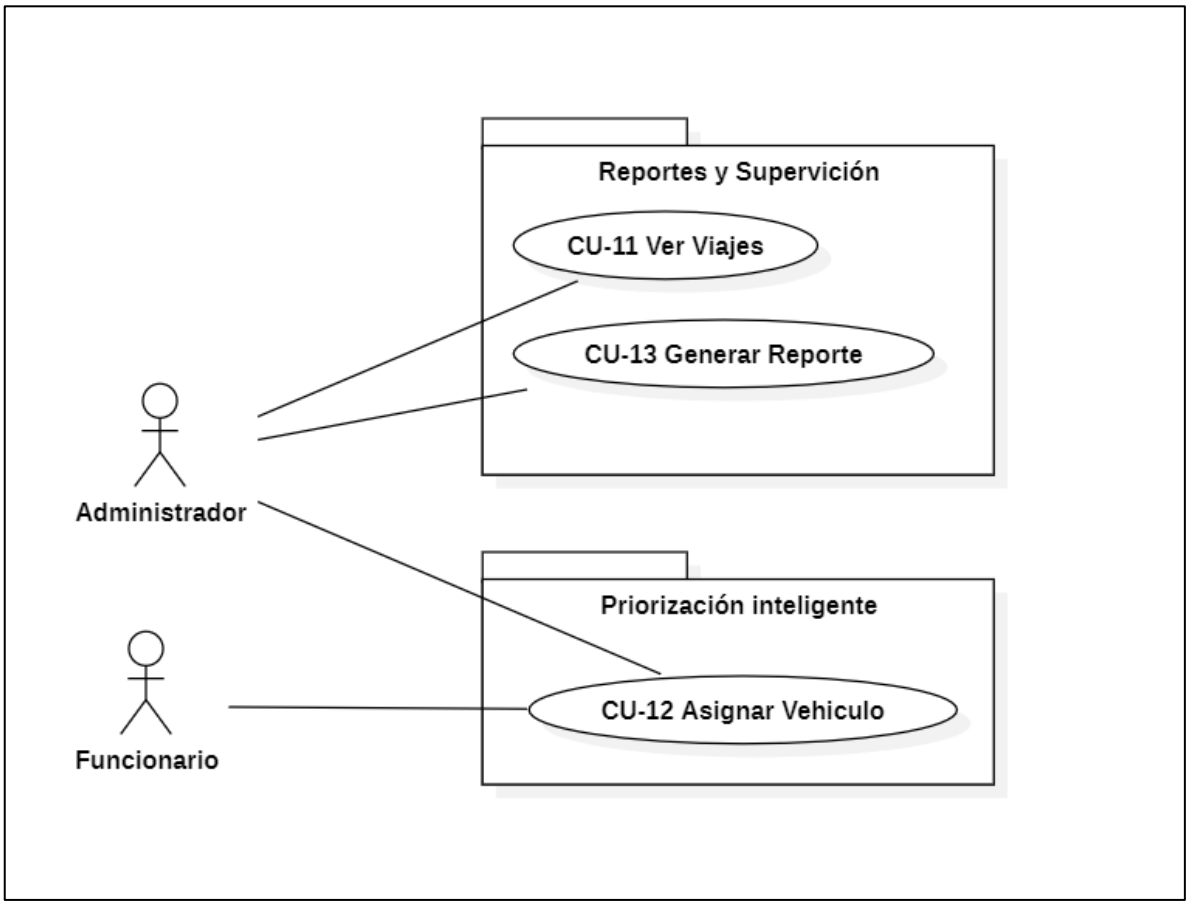

Figura 4.9 – Diagrama casos de uso "Reportes y Supervisión" y "Priorización inteligente"

Síntesis de los Principales Casos de Uso del Sistema.

Una vez definidos los módulos, se detallaron 23 casos de uso específicos (identificados como CU-01 a CU-23) que en conjunto describen todo el funcionamiento de FlotUp.

Solicitud y programación del viaje: El proceso inicia cuando el Funcionario solicitante ingresa al sistema para gestionar un traslado. A través de la plataforma web, el funcionario registra una nueva solicitud de viaje, especificando los datos necesarios: lugar de origen, destino, fecha y hora requerida de llegada, número de pasajeros y cualquier observación pertinente.

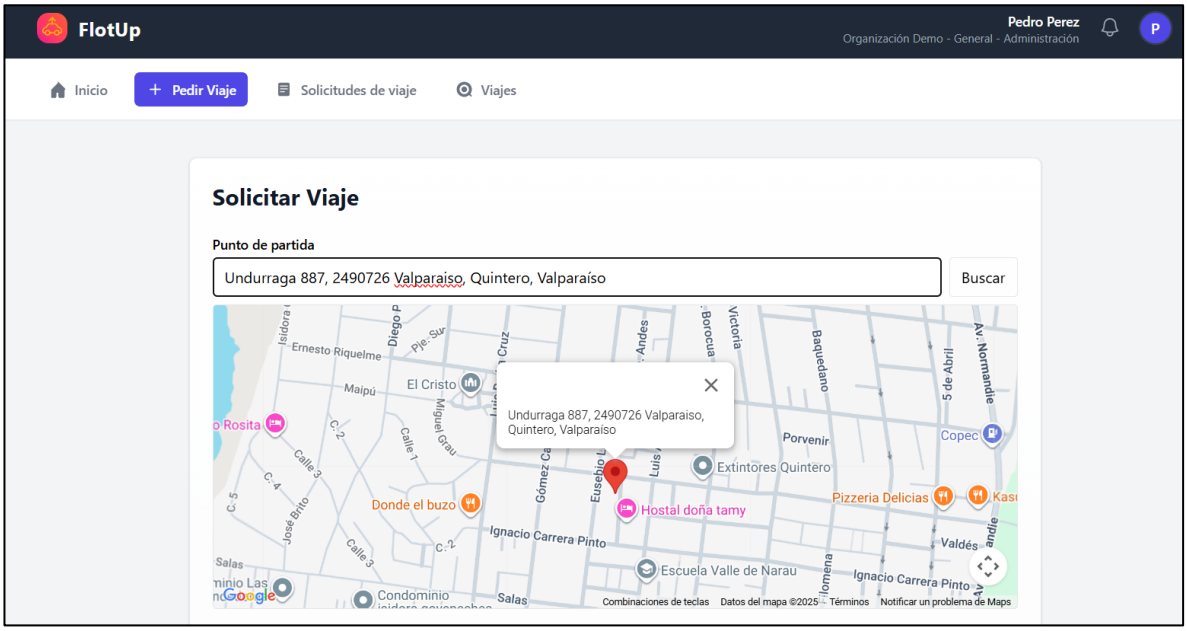

Figura 4.10 – Captura de pantalla "Formulario de solicitud de viaje - 1" de FlotUp.

Figura 4.11 – Captura de pantalla "Formulario de solicitud de viaje - 2" de FlotUp.

Este caso de uso fundamental permite formalizar digitalmente la petición que antes se hacía vía telefónica, quedando ahora almacenada en la base de datos con un identificador único y en estado "Pendiente". El funcionario puede consultar el estado de su solicitud en cualquier momento desde la misma interfaz, lo que materializa otro caso de uso enfocado en brindar trazabilidad: la plataforma muestra si la solicitud está en espera de aprobación, ya fue atendida, o si ha sido asignada a un vehículo. Incluso se

notifica al funcionario en tiempo real cuando su petición cambia de estado eliminando la incertidumbre propia del sistema manual previo.

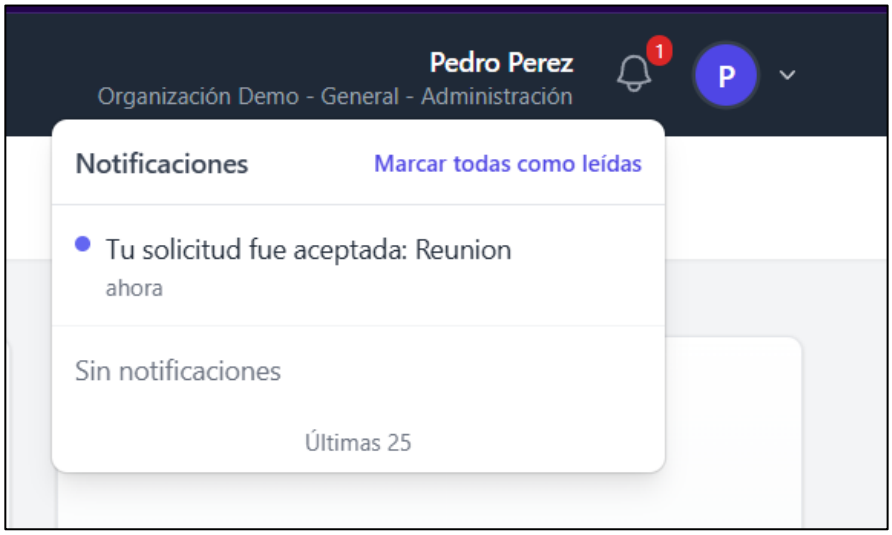

Figura 4.12 – Captura de pantalla "Notificación de funcionario" de FlotUp.

Revisión y aprobación por el administrador: Una vez existe al menos una solicitud pendiente, entra en juego el Administrador de flota, quien periódicamente accede al módulo de gestión para visualizar las solicitudes de viaje pendientes. En este caso de uso, el administrador ve un listado de todas las peticiones en cola junto con sus detalles.

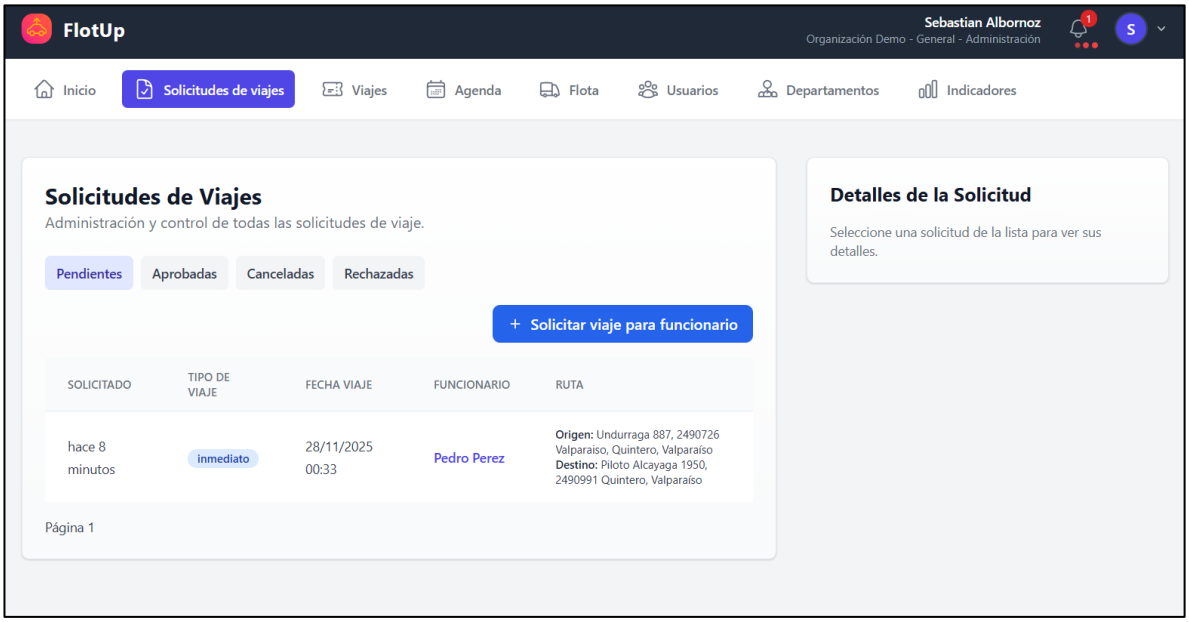

Figura 4.13 – Captura de pantalla "Solicitudes de viaje – administrador" de FlotUp.

El administrador evalúa la pertinencia de cada solicitud conforme a las políticas institucionales y la disponibilidad de recursos. Luego, mediante las funciones de la plataforma, decide aprobar o rechazar la solicitud. Si rechaza una petición (caso de uso correspondiente a denegar viaje), puede adjuntar una justificación que quedará visible al funcionario, cerrando así ese requerimiento en el sistema. Si, en cambio, aprueba la solicitud (caso de uso de aprobación de viaje), el sistema la cambia al estado "Aprobada"

y de inmediato se despliega la interfaz para proceder con la asignación del viaje. Todo ello ocurre manteniendo al administrador como autoridad final, de acuerdo con la lógica institucional de que ninguna salida se realiza sin su consentimiento explícito.

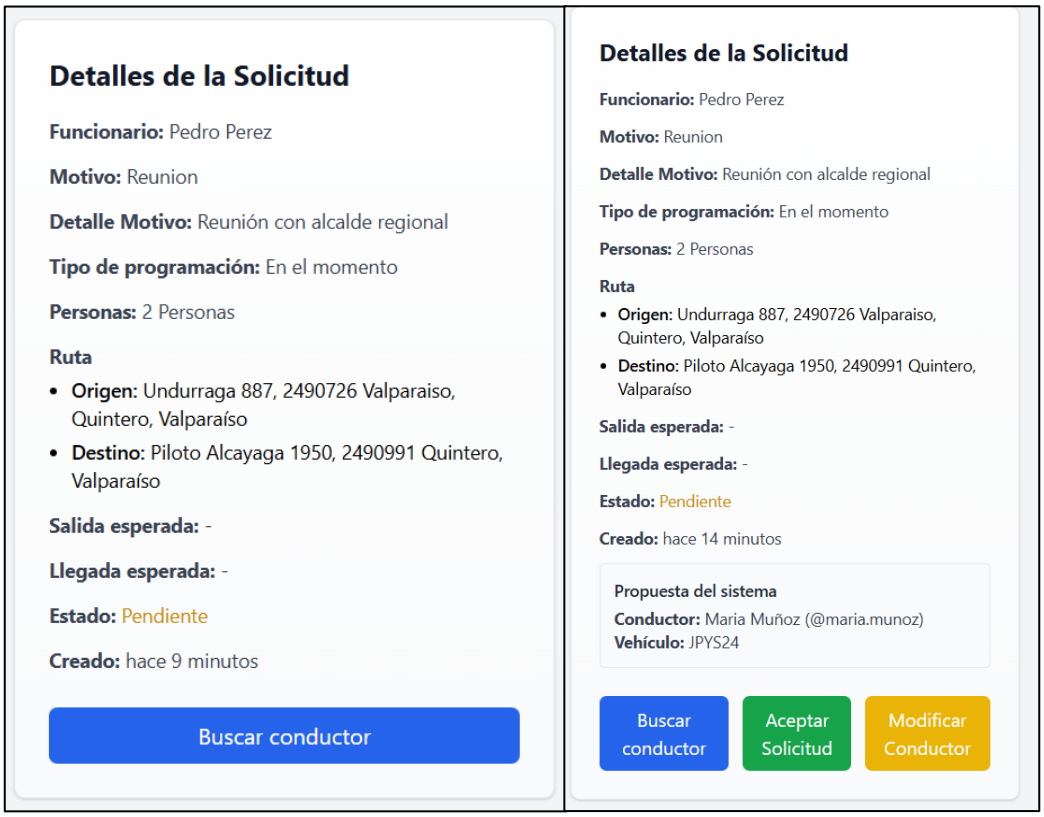

Figura 4.14 – Panel de decisión del administrador sobre una solicitud de viaje

Asignación dinámica de vehículo: Al aprobar una solicitud, se inicia el proceso de asignación del viaje, en el cual el Administrador selecciona qué vehículo y qué conductor atenderán el requerimiento.

Aquí se entrelazan varios casos de uso de distinta naturaleza. Por un lado, el administrador puede usar la funcionalidad tradicional de asignar manualmente un vehículo: revisa la lista de vehículos disponibles y elige uno que se ajuste a las necesidades del viaje. Asimismo, designa un chofer disponible para conducir ese vehículo, típicamente según turnos o carga de trabajo.

Por otro lado, FlotUp ofrece asistencia a través de su Priorización Inteligente: en el momento de la asignación, el sistema genera una sugerencia óptima considerando la ubicación actual de los vehículos (obtenida mediante geolocalización en tiempo real), sus condiciones (combustible, estado mecánico) y los horarios. El algoritmo también detecta oportunidades de viaje compartido, es decir, verifica si la solicitud recién aprobada puede combinarse con otra ruta ya planificada o con otra solicitud pendiente que tenga trayecto similar.

Estas recomendaciones automáticas se presentan al administrador en la interfaz de asignación: el caso de uso correspondiente describe cómo el administrador visualiza las sugerencias del sistema y puede aceptar la recomendación más conveniente con un solo clic. En caso de aceptar, el sistema realiza automáticamente la vinculación del vehículo

sugeridos a ambas solicitudes, actualizando los estados y notificaciones pertinentes. Si el administrador decide no seguir la sugerencia siempre puede optar por reasignar manualmente otra unidad o modificar la elección antes de confirmarla.

Finalmente, al completar la asignación, se crea la orden de viaje en el sistema: la solicitud aprobada pasa al estado "Asignada" con un vehículo y conductor determinados, y los actores involucrados son notificados. En especial, para el Conductor asignado se habilita inmediatamente un caso de uso específico en su aplicación móvil, que es la recepción de los detalles del viaje programado.

Ejecución del viaje por el conductor: Con el viaje ya asignado, el foco operativo se traslada al conductor, quien interactúa con el sistema a través de una aplicación móvil diseñada para su rol.

En el instante en que un administrador confirma la asignación, el conductor recibe una notificación y puede consultar el detalle del viaje asignado. Dicho detalle incluye la información necesaria: quién es el funcionario pasajero y su contacto, dónde debe recogerlo (punto de partida), cuál es el destino final, a qué hora se programó el inicio y cualquier indicación particular (por ejemplo "llevar cono de estacionamiento" o "esperar 15 minutos adicionales si es necesario").

En cuanto el chofer se dirige al punto de recogida, inicia el caso de uso de comenzar el viaje. Desde ese instante, el sistema empieza a monitorear el recorrido: mediante la geolocalización continua del dispositivo móvil o del vehículo, se registra la posición, habilitando el caso de uso de supervisión en tiempo real para el administrador. Durante el trayecto, el conductor puede ver la ruta sugerida (integrada con servicios de mapas) y el sistema puede notificar tanto al funcionario como al administrador de eventos básicos, por ejemplo "vehículo en camino" o "vehículo próximo a llegada".

Si ocurriera alguna incidencia en curso (como un desvío significativo o algún contratiempo) el conductor puede utilizar la app para informar la situación lo cual quedaría registrado inmediatamente. Finalmente, al arribar al destino y completar el traslado, el chofer finaliza el viaje marcándolo en la aplicación (caso de uso de término de viaje), momento en el cual el sistema actualiza el estado del viaje a "Completado".

Supervisión y control durante el viaje: Paralelamente a la ejecución, el Administrador está facultado para monitorizar el viaje en curso en todo momento, gracias al módulo de Reportes y Supervisión.

En la interfaz administrativa, un mapa en tiempo real muestra un marcador para cada vehículo activo, permitiendo al administrador seguir la trayectoria del conductor asignado (caso de uso de monitoreo de flota en tiempo real). Además de la posición, el administrador puede consultar información instantánea como la hora estimada de llegada calculada según las condiciones actuales, y cualquier alerta generada.

Este monitoreo activo habilita al administrador a tomar acciones rápidas ante imprevistos: por ejemplo, si observa que un vehículo está detenido anormalmente, puede contactar al conductor o enviar asistencia. También sirve para asegurar la transparencia del servicio, ya que queda constancia digital de los tiempos y rutas efectivamente realizados.

Simultáneamente, el administrador puede comenzar a registrar datos para reportes operativos: durante o tras el viaje, la plataforma compila automáticamente estadísticas,

que alimentan los casos de uso de generación de reportes. De este modo, la supervisión en tiempo real y la posterior elaboración de informes están estrechamente ligadas a cada viaje gestionado, formando parte integral del flujo de uso del sistema por parte del administrador.

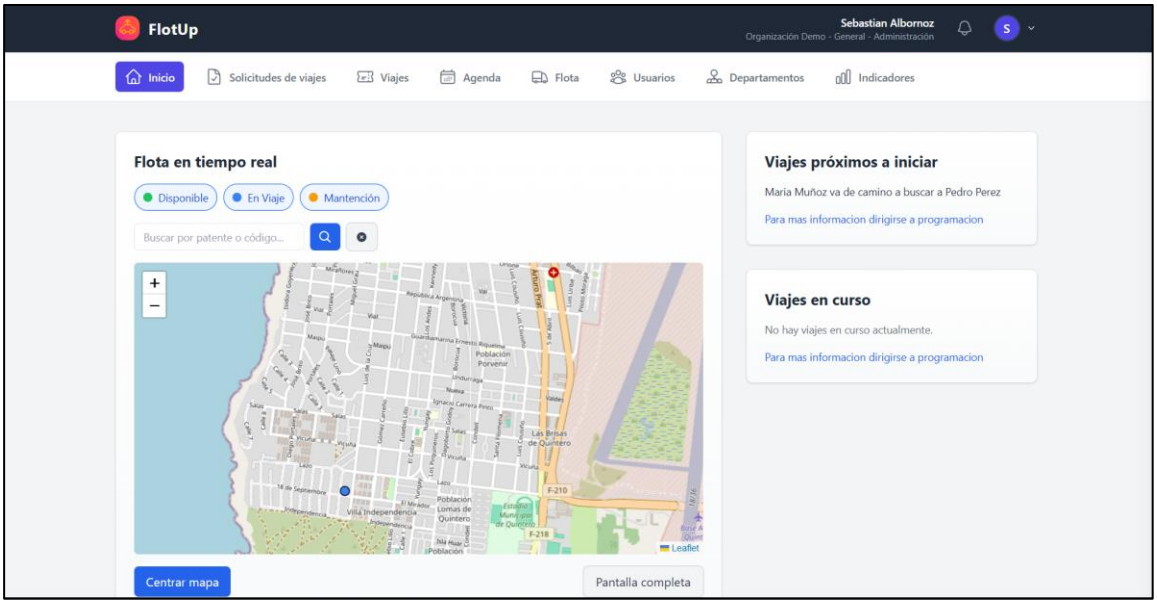

Figura 4.15 – Captura de pantalla "Home del administrador".

Finalización del viaje y retroalimentación: Al darse por concluido un viaje, el sistema cierra el ciclo operativo de ese traslado en particular. Es en este punto donde los casos de uso del módulo Evaluación e Incidencias entran en juego para capturar la retroalimentación y los eventos post servicio.

Primero, se activa el caso de uso de evaluación del viaje: el funcionario recibe un formulario breve para calificar el servicio recibido. En términos prácticos, el Funcionario puede asignar una puntuación (por ejemplo, de 1 a 5 estrellas). Esta evaluación voluntaria, pero altamente incentivada dentro de la cultura organizacional, tiene el propósito de medir la satisfacción del usuario y detectar oportunidades de mejora. Los datos de evaluaciones quedan almacenados vinculados al viaje correspondiente y al perfil del conductor, de manera que el Administrador pueda luego consultar las métricas de calidad de servicio por conductor o en general.

En segundo lugar, si el funcionario o el conductor desean reportar una incidencia específica relacionada con el viaje recién finalizado, pueden hacerlo mediante la opción destinada a tal fin. Un caso de uso cubre la notificación de incidentes: el funcionario podría reportar, por ejemplo, que el vehículo presentaba desperfectos, o que el viaje se demoró excesivamente por causas atribuibles al chofer; el conductor, por su parte, podría reportar situaciones como comportamiento inadecuado de pasajeros, un accidente ocurrido, o cualquier evento fuera de la normalidad. Este proceso cierra formalmente el flujo de ese viaje, asegurando que no sólo se completó en términos de transporte, sino que también se atendieron las consecuencias o evaluaciones posteriores.

Además del flujo central "solicitud-asignación-ejecución-evaluación" descrito arriba, el sistema contempla otros casos de uso complementarios que garantizan el funcionamiento continuo de la plataforma.

Entre ellos está la gestión de usuarios, que permite dar de alta nuevos funcionarios en la plataforma, asignar roles y mantener actualizada la lista de usuarios autorizados. Si bien estas tareas de administración de cuentas pueden considerarse externas al núcleo de Gestión de Viajes, son indispensables para controlar el acceso seguro al sistema.

Asimismo, casos de uso como inicio de sesión y cierre de sesión se dan por supuestos en la operación diaria: cada actor debe autenticarse con sus credenciales para acceder a las funciones descritas, asegurando que únicamente personal autorizado utilice la plataforma. Aunque estos casos de uso de autenticación no se detallan en el flujo narrativo principal por ser transversales y de soporte, forman parte del conjunto completo de 23 casos de uso identificados en la especificación del sistema.

4.6 Diagramas de secuencia UML.

Si bien en la Municipalidad de Viña del Mar el proceso de asignación de viajes es semiautomático (ya que toda solicitud debe ser previamente aprobada o rechazada por el administrador de flota), el sistema FlotUp fue diseñado con flexibilidad para organizaciones que no requieren validación previa, permitiendo así una asignación completamente automática desde la creación de la solicitud. Esta configuración hace que el motor de asignación sea más dinámico y adaptable, pudiendo ajustarse a distintos modelos operativos según las necesidades de cada entidad usuaria.

Asignación automática Inmediata (NOW).

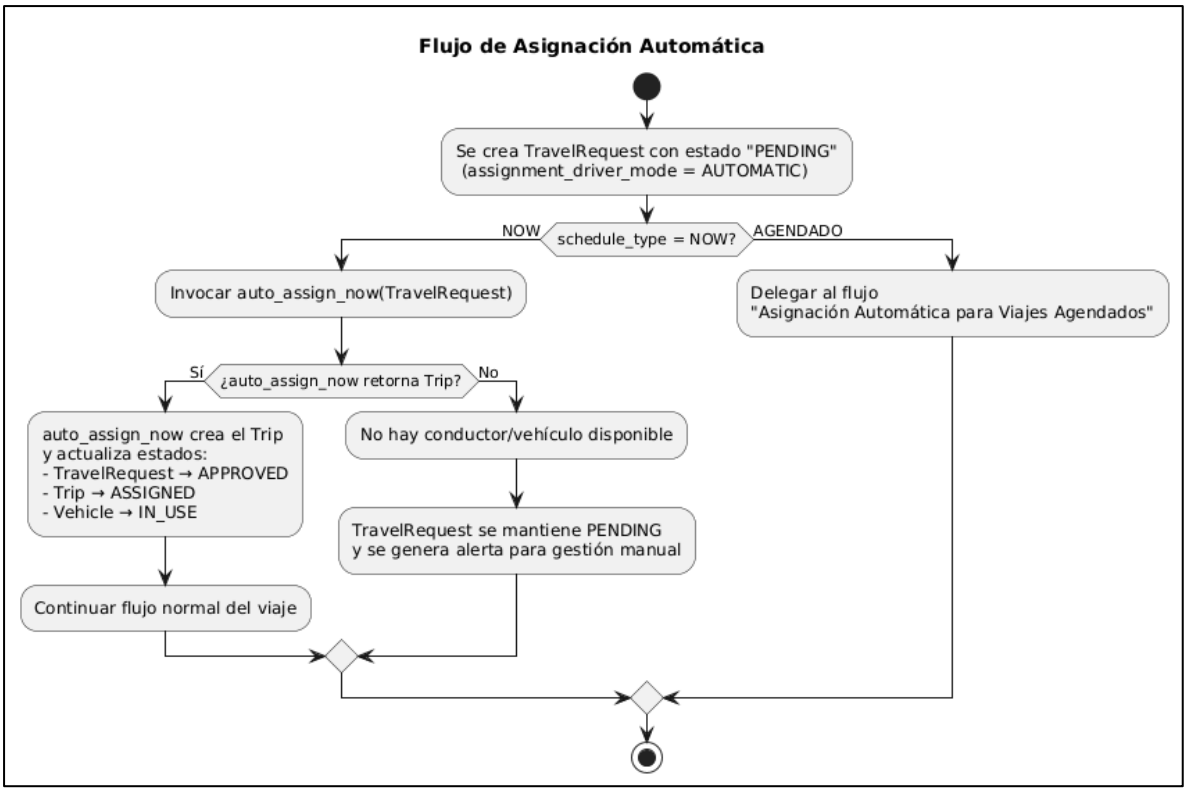

Figura 4.16 – Flujo de asignación automática.

El diagrama de secuencia de la figura 4.16 ilustra el proceso completo desde que un funcionario solicitante crea una nueva solicitud de viaje hasta la asignación automática de un vehículo y un conductor disponibles.

La secuencia comienza con el registro de la solicitud de viaje (TravelRequest) a través de la plataforma, utilizando los servicios REST del backend. Acto seguido, se invoca la rutina auto_assign_now(), que da inicio al proceso automatizado de búsqueda y asignación de recursos.

Asignación automática agendada.

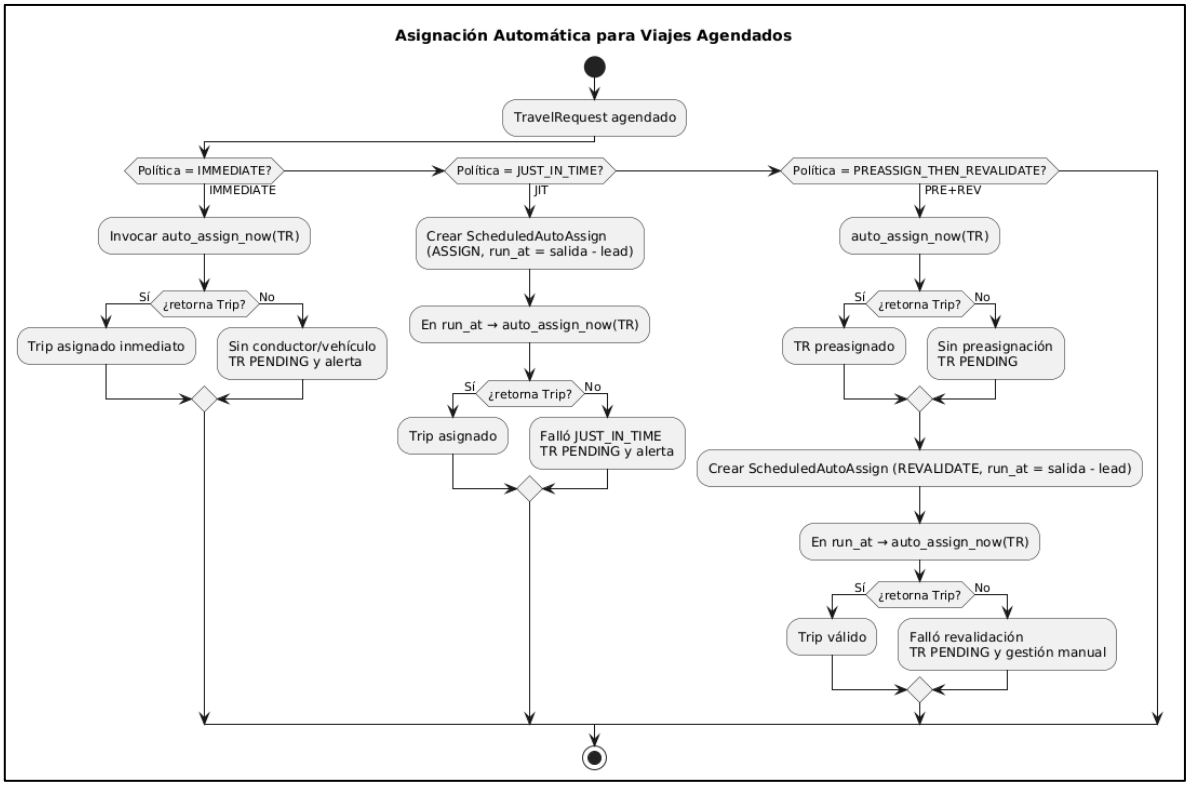

Figura 4.17 – Flujo de asignación automática agendada.

El diagrama de secuencia de la figura 4.17 describe el mecanismo de asignación automática de vehículos para solicitudes de viaje con programación futura (schedule_type = AGENDADO). A diferencia del flujo inmediato, aquí el sistema delega la asignación a una tarea programada (ScheduledAutoAssign) gestionada por un Scheduler, cuya ejecución se rige por la política de asignación configurada para cada organización: IMMEDIATE, JUST_IN_TIME o PREASSIGN_THEN_REVALIDATE.

En la política IMMEDIATE, aun cuando el viaje es agendado, el comportamiento es equivalente al flujo NOW: el vehículo se asigna en el momento en que se registra la solicitud, de modo que el viaje queda completamente resuelto desde el inicio.

En la política JUST_IN_TIME, en cambio, el viaje se registra pero la asignación se posterga hasta pocos minutos antes de la hora de salida, según un margen definido por la propia organización; en ese instante el Scheduler invoca auto_assign_now(), utilizando información actualizada de disponibilidad y geolocalización.

Finalmente, en la modalidad PREASSIGN_THEN_REVALIDATE, el sistema realiza primero una preasignación anticipada mediante auto_assign_now() y, posteriormente, ejecuta una revalidación a pocos minutos del inicio del viaje, volviendo a llamar a auto_assign_now() para confirmar o corregir la asignación inicial. En cualquiera de estos esquemas, si no se encuentra un candidato válido, la solicitud se mantiene en estado PENDING y se genera una alerta para su gestión manual por parte del administrador.

Flujo Interno de auto_assign_now()

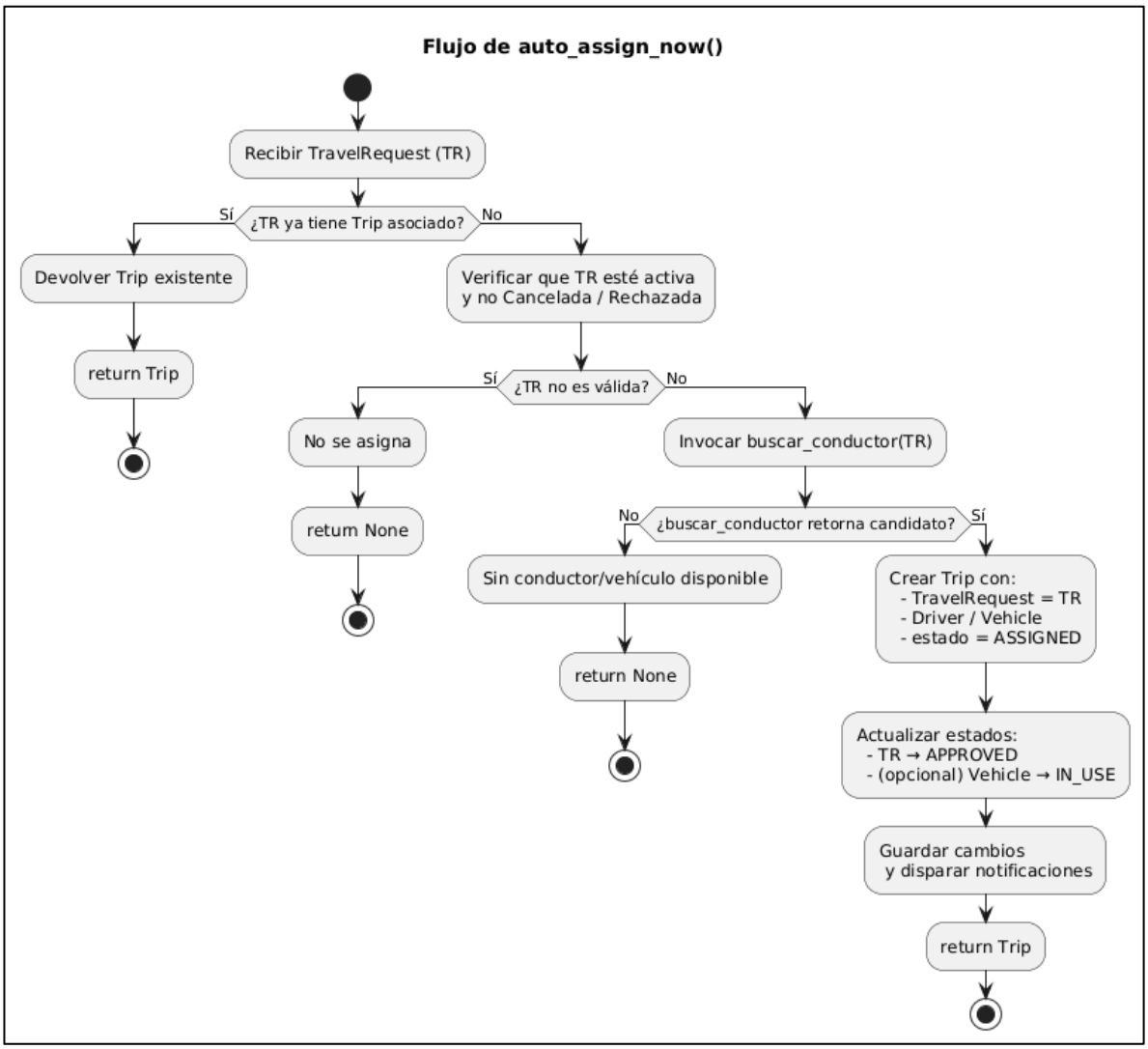

Figura 4.18 – Flujo de auto_assign_now().

El diagrama de la figura 4.18 modela el flujo interno de la función auto_assign_now(), que actúa como núcleo del motor de asignación dinámica. Este método es invocado tanto en asignaciones inmediatas como agendadas, y su lógica incluye una serie de validaciones previas: verifica si la solicitud ya tiene un viaje asociado, si se encuentra activa y si su estado lo permite (no cancelada ni rechazada). Si cumple estas condiciones, se procede a invocar el método buscar_conductor().

La respuesta de esta función determina la continuidad del proceso: si retorna un candidato, se procede a crear el Trip, actualizar el estado de la solicitud y vehículo, y notificar a los involucrados; en caso contrario, se retorna None, dejando el TravelRequest sin asignación. Esta estructura robusta permite controlar múltiples escenarios desde un punto centralizado, garantizando coherencia y eficiencia.

Flujo Interno de buscar_conductor(TR)

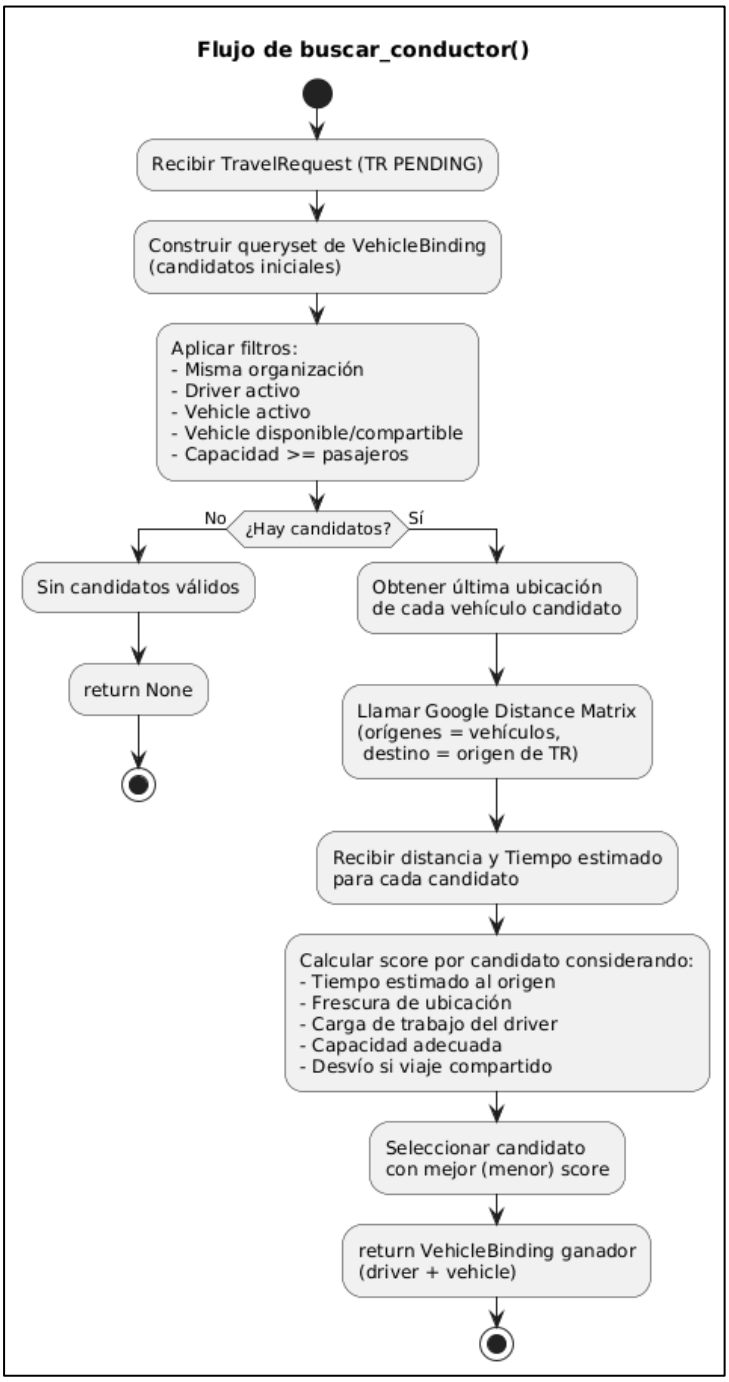

Figura 4.19 – Flujo de buscar_conductor().

El diagrama de la figura 4.19 corresponde al flujo detallado de buscar_conductor(TR), función encargada de seleccionar al conductor y vehículo más apropiados para una solicitud. El proceso inicia con la construcción de un queryset de objetos VehicleBinding, filtrados por criterios como pertenencia a la misma organización, estado activo, disponibilidad y capacidad suficiente para la solicitud.

Posteriormente, se obtiene la última ubicación de cada vehículo candidato, ya sea por base de datos o por geolocalización en tiempo real, y se consulta la API Google Distance Matrix para calcular la distancia y ETA (Tiempo estimado de llegada) hacia el origen del viaje. Con esos datos, se calcula un score para cada candidato, ponderando factores como: tiempo estimado de llegada, frescura de la ubicación, carga de trabajo actual del conductor, compatibilidad de capacidad y si la solicitud es compartible.

El candidato con menor puntaje es seleccionado como óptimo y retornado para su uso en auto_assign_now(). Si no hay candidatos válidos, se retorna None, lo cual permite al sistema gestionar la situación de manera adecuada (ya sea mediante reintento, alerta o asignación manual).

4.7 Criterios de diseño.

En el diseño del sistema FlotUp, y particularmente de su motor de asignación dinámica de vehículos, se aplicaron una serie de principios orientados a garantizar eficiencia, trazabilidad, mantenibilidad y adaptabilidad del sistema.

Eficiencia y proximidad en la asignación.

El algoritmo de asignación fue concebido para optimizar el uso de la flota, minimizando tiempos muertos y distancias recorridas innecesariamente. Para cada solicitud de viaje, el sistema sugiere automáticamente el vehículo óptimo disponible en función de criterios en tiempo real, priorizando aquel que se encuentre más cercano al origen de la solicitud y que esté libre.

De esta manera se reducen los tiempos de espera y se mejora el ETA (tiempo estimado de llegada) para recogida. Además, se contempló la compatibilidad de paradas y rutas con el fin de habilitar viajes compartidos: el sistema es capaz de detectar cuándo dos o más solicitudes tienen trayectos similares, destinos próximos y ventanas de tiempo solapadas para proponer su combinación en un solo recorrido. Esto permite agrupar pasajeros en un mismo vehículo cuando las condiciones lo favorecen, evitando recorridos duplicados y aumentando la eficiencia global.

Asimismo, el diseño considera la distribución de la carga de trabajo entre conductores de forma equitativa (por ejemplo, evitando sobrecargar siempre al mismo chofer), aunque siempre respetando restricciones operativas como límites de pasajeros y disponibilidad de cada vehículo.

Trazabilidad y consistencia en la ejecución.

Desde el inicio se priorizó que todas las decisiones del motor de asignación quedaran registradas y que el estado de cada solicitud y viaje fuera siempre consistente. Cada solicitud de viaje (TravelRequest) mantiene un registro completo de sus cambios de estado de modo que se puede trazar el ciclo de vida de la solicitud desde su creación por un funcionario hasta la finalización del viaje.

Cuando el algoritmo realiza una asignación, esta acción queda persistida en la base de datos mediante la creación de un objeto Trip asociado a la solicitud, garantizando así que la decisión quede almacenada para auditorías futuras.

Adicionalmente, para las asignaciones automáticas programadas, el sistema lleva un registro de la tarea en la entidad ScheduledAutoAssign, incluyendo campos de estado, marca temporal de ejecución y mensajes de error en caso de fallo. Esto significa que si una asignación automática no prospera (por ejemplo, por falta de conductores disponibles), el evento queda en registros con un mensaje descriptivo, facilitando el análisis y la depuración de problemas.

Para asegurar la consistencia de cada operación, se utilizaron transacciones atómicas en las secciones críticas del código, de forma que si algo falla durante el proceso de asignación, la base de datos revierte al estado previo, evitando así resultados parciales o estados indeterminados. Todo esto contribuye a un sistema confiable, donde cada paso del algoritmo es transparente y verificable.

Separación de responsabilidades (modularidad).

FlotUp se estructuró siguiendo una arquitectura modular clara, separando las responsabilidades clave en distintos módulos o capas. Por un lado, la lógica de negocio central relacionada con viajes y asignaciones reside en el módulo Trips, que administra las solicitudes y la creación de viajes, incluyendo la implementación del algoritmo de asignación dinámica. Por otro lado, la gestión de vehículos y conductores se encapsula en el módulo Fleet, mientras que todo lo referente a usuarios, perfiles y organizaciones corresponde al módulo Accounts.

Esta división en dominios garantiza que cada módulo se enfoque en un aspecto específico del sistema, reduciendo el acoplamiento. Además, se implementó un módulo Core común para funcionalidades transversales y utilidades compartidas, evitando duplicación de código y favoreciendo la consistencia en toda la aplicación.

En términos de arquitectura de software, se siguió el patrón MVT (Modelo-Vista-TEMPLATE) propio de Django, manteniendo la lógica del negocio en servicios o modelos (por ejemplo, funciones auxiliares como auto_assign_now() encapsulan la lógica de asignación), la lógica de presentación en las vistas/plantillas del módulo Panel, y la persistencia en los modelos de datos.

Los controladores de la API (vistas Django REST Framework) se limitaron a orquestar llamadas a estos servicios internos, aplicando validaciones de permisos, pero delegando la toma de decisiones de negocio al módulo correspondiente. Esta separación de responsabilidades facilita la mantenibilidad y escalabilidad del sistema: cada componente puede evolucionar independientemente.

Por ejemplo, la interfaz web (módulo Panel) está lo suficientemente desacoplada de la lógica de asignación que podría reemplazarse o modificarse (incluso migrar a un frontend JavaScript independiente en el futuro) sin alterar el núcleo del backend. Así, se logra una arquitectura limpia donde las APIs, la lógica de negocio y la capa de datos están bien delimitadas, siguiendo principios de diseño sólido.

Escalabilidad y visión SaaS.

Otro pilar del diseño fue garantizar que la solución pudiera escalar en cuanto a volumen de datos, usuarios y también adaptarse a múltiples organizaciones (multi-tenant), pensando en un futuro despliegue SaaS (Software as a Service). En su fase inicial, FlotUp está implementado para la Municipalidad de Viña del Mar, pero el sistema fue diseñado para funcionar con igual eficacia en otras municipalidades o entidades que pudieran adoptarlo.

Esto se refleja en varias decisiones de diseño. Primero, la aplicación soporta multi-organización: los datos y operaciones están segmentados por organización, de modo que un administrador de Viña del Mar solo ve y gestiona su flota, pero el mismo sistema podría dar servicio a otra municipalidad con sus propios vehículos y usuarios en aislamiento. Muchas reglas de negocio incluyen esta noción; por ejemplo, un conductor solo puede asignarse vehículos pertenecientes a su propia organización, protegiendo los límites entre clientes.

Asimismo, las políticas de asignación de viajes agendados son configurables por organización, permitiendo que cada entidad defina si sus solicitudes programadas se asignan inmediatamente, justo antes del inicio, o con un modelo híbrido. Esta flexibilidad por configuración garantiza que el algoritmo se adapte fácilmente a las necesidades de cada cliente sin cambiar el código.

En cuanto a escalabilidad técnica, el motor de asignación emplea estrategias eficientes de búsqueda de conductores: por ejemplo, utiliza consultas geoespaciales optimizadas (PostGIS) para encontrar conductores cercanos dentro de un radio determinado, ordenando por distancia. Esto limita el universo de búsqueda a los candidatos más relevantes, reduciendo la carga computacional incluso si la flota crece considerablemente.

También se introdujo un scheduler (planificador de tareas) para manejar asignaciones diferidas en el tiempo (viajes agendados), de forma que estas ocurran en segundo plano y no afecten el rendimiento del sistema en tiempo real.

La combinación de un diseño modular, consultas optimizadas y capacidades multi-tenant asegura que FlotUp pueda escalar tanto verticalmente (más solicitudes y vehículos dentro de una organización) como horizontalmente (nuevas organizaciones clientes), manteniendo tiempos de respuesta adecuados y confiabilidad.

Tolerancia a fallos y *fallback* manual.

Dado que en un entorno real pueden ocurrir escenarios imprevistos, el sistema se diseñó con mecanismos de tolerancia a fallos y recuperación manual para la asignación de vehículos. En el flujo de asignación automática, si el algoritmo no logra encontrar un conductor/vehículo disponible apropiado para una solicitud (por ejemplo, en horas pico con alta demanda o si ningún vehículo cumple los criterios), la solicitud no se pierde ni queda en un limbo: el sistema la mantiene en estado Pendiente y genera una alerta o notificación para que un administrador intervenga manualmente.

Este comportamiento asegura que ninguna solicitud de viaje quede desatendida por culpa de una automatización fallida; siempre habrá oportunidad de asignación manual por parte del administrador ante errores o resultados nulos del algoritmo.

De igual forma, se llevaron a cabo consideraciones de atomicidad: operaciones como la asignación de un vehículo a un conductor ocurren dentro de transacciones atómicas; si alguna sub-operación falla, la transacción completa se revierte para no dejar datos inconsistentes.

Finalmente, el sistema proporciona interfaces administrativas (vía Panel o el administrador de Django) donde las asignaciones pueden ser ajustadas o forzadas manualmente en cualquier momento por un usuario autorizado. Esto sirve como red de seguridad: ante cualquier contingencia donde la automatización no funcione como esperado, el control humano puede retomar la operación de forma ágil.

Modelo semiautomático con miras a automatización completa.

Por requisitos institucionales, la primera implementación de FlotUp en Viña del Mar optó por un esquema semiautomático en la asignación de vehículos. Esto significa que cada solicitud de viaje ingresada debe ser validada previamente por un administrador de flota, quien decide aprobarla o rechazarla antes de que ocurra cualquier asignación.

Este paso manual obedece a consideraciones de transparencia y control en la administración pública, asegurando que un responsable humano confirme la pertinencia del viaje, y se integra sin fricción en el flujo general del sistema. Sin embargo, uno de los criterios de diseño fue la flexibilidad para adaptarse a entornos con menor intervención humana.

El motor de asignación fue concebido de tal forma que, cambiando configuraciones, puede funcionar de manera completamente automática si así se desea. De hecho, FlotUp soporta diversos modelos operativos configurables: asignación inmediata en el momento de crear la solicitud, asignación just-in-time pocos minutos antes de la hora de salida programada, o un enfoque mixto de preasignación con revalidación posterior.

Estas modalidades se controlan por organización y se implementaron para que el mismo núcleo del algoritmo sirva a distintas políticas sin modificaciones estructurales. La justificación de mantener un modelo semiautomático en Viña del Mar fue asegurar la alineación con los procesos administrativos existentes y cumplir con normas internas de aprobación, pero gracias al diseño modular y parametrizable, el sistema está preparado para organizaciones que requieran un grado mayor de automatización.

5 Validación de la solución.

5.1 Metodología de validación del motor de asignación dinámica.

La validación del motor de asignación dinámica desarrollado para FlotUp se enfocó en comprobar su funcionamiento efectivo bajo diversas políticas operativas y escenarios representativos. Para ello, se combinaron pruebas funcionales en un entorno de desarrollo controlado con simulaciones que replican condiciones reales de uso.

Se abordaron las tres modalidades principales de asignación automática (IMMEDIATE, JUST_IN_TIME y PREASSIGN_THEN_REVALIDATE), junto con pruebas específicas de la función buscar_conductor(), incorporando además situaciones límite como solicitudes concurrentes, ausencia total de conductores y variaciones dinámicas en la disponibilidad.

Las pruebas se realizaron utilizando los mismos modelos de datos definidos para el sistema en producción: TravelRequest, Trip, Vehicle, VehicleBinding, entre otros. Cada escenario de validación consistió en crear datos representativos (incluyendo coordenadas reales para conductores y destinos) y ejecutar las funciones del motor de asignación bajo condiciones monitoreadas. Se definieron métricas claves alineadas con los criterios de éxito de la solución, tales como la proporción de asignaciones automáticas exitosas, el tiempo promedio de respuesta, la integridad de los estados y el manejo correcto de errores o fallas de asignación.

Un aspecto fundamental fue garantizar la trazabilidad del proceso. Se verificó que el sistema registrara con precisión cada acción programada mediante el modelo ScheduledAutoAssign, reflejando adecuadamente transiciones de estado y marcas temporales. También se inspeccionaron manualmente las entidades en base de datos tras cada prueba para asegurar coherencia entre la solicitud, el viaje y el conductor asignado. En los casos que lo ameritaban, se activaron mecanismos de depuración para examinar los cálculos de puntuación realizados por el algoritmo buscar_conductor().

5.2 Pruebas funcionales del algoritmo de asignación:

Validación del flujo IMMEDIATE.

El flujo IMMEDIATE corresponde a la asignación inmediata de un vehículo en cuanto se genera la solicitud de viaje. En el contexto de FlotUp, esto ocurre en dos situaciones: solicitudes con hora de salida "ahora" (no programadas), y solicitudes programadas cuyo parámetro de política de asignación diferida se establece en IMMEDIATE. En ambos casos, el sistema invoca la función auto_assign_now() inmediatamente tras crear la TravelRequest, con el fin de vincularla de forma instantánea con un conductor disponible.

Para validar este flujo, se ejecutó el caso de prueba creando una solicitud inmediata (ScheduleType.NOW) bajo una organización con al menos un conductor disponible. El resultado esperado era la creación de un objeto Trip asociado a dicha solicitud en cuestión de segundos y la actualización del estado de la solicitud a Aprobada. Los resultados confirmaron esta expectativa, el sistema seleccionó automáticamente un VehicleBinding activo entre los disponibles, priorizando la disponibilidad y la idoneidad operativa del recurso, y generó el Trip correspondiente.

La solicitud de viaje pasó de estado Pendiente a Aprobada de forma inmediata, registrándose así la aprobación automática y quedando el viaje en estado Asignado. Durante la validación, se observó que el sistema seleccionó correctamente un VehicleBinding disponible que cumplía con las condiciones de disponibilidad, evitando la asignación de conductores ya ocupados o no operativos. El comportamiento evidenció que el proceso de selección garantiza la elección de un único candidato apto por ejecución, manteniendo la integridad de la asignación y evitando conflictos.

Adicionalmente, se evaluó el comportamiento del sistema ante la ausencia total de conductores disponibles al momento de la solicitud. En este escenario, auto_assign_now retornó None, al no encontrar ningún vínculo habilitado. La prueba confirmó que el sistema gestionó correctamente esta situación: no se generó ningún objeto Trip y la solicitud permaneció en estado Pendiente. Desde la interfaz administrativa, el sistema notificó el resultado mediante un mensaje de advertencia, permitiendo a los operadores continuar el proceso de manera manual. Este comportamiento responde al diseño previsto de fallback manual en aquellos casos donde la automatización no logra asignar un recurso válido.

Validación flujo JUST_IN_TIME.

La modalidad JUST_IN_TIME difiere de la anterior en que las solicitudes de viaje programadas no se asignan de inmediato, sino unos minutos antes de la hora de salida prevista. Esto busca asignar el conductor "justo a tiempo", maximizando la probabilidad de que esté disponible y cercano al momento de partida.

En la implementación, al crear una TravelRequest agendada bajo política JIT, el sistema no crea un Trip inmediato sino un objeto ScheduledAutoAssign con acción ASSIGN y con run_at programada, según la política de la organización. Este registro queda en estado Pendiente (PENDING) hasta que un proceso de fondo ejecute la asignación llegada la hora indicada.

Para probar este flujo, se generó una solicitud de viaje con hora futura en una organización con conductores disponibles, configurando la política de asignación programada en JUST_IN_TIME. Al guardar la solicitud, se verificó que no existía un Trip inmediato y que en su lugar se había creado un ScheduledAutoAssign vinculado a la solicitud. Los campos de este objeto mostraron action = ASSIGN y status = PENDING, con un run_at aproximadamente 15 minutos antes de la hora de salida del viaje (ajustado según el lead time definido). Esta confirmación indicó que el sistema había registrado correctamente la tarea de auto-asignación diferida.

El siguiente paso de la validación fue simular la ejecución de dicha tarea programada. Al ejecutar el procesador, se observó que éste identificó la tarea pendiente y llamó internamente a auto_assign_now() para esa solicitud. En el caso donde había conductores disponibles, la función asignó exitosamente un conductor como en el flujo inmediato, creando el Trip y aprobando la solicitud en ese instante. Consecuentemente, el registro ScheduledAutoAssign pasó a estado DONE y se le estampó la hora de procesamiento, sin mensaje de error. El resultado práctico fue que, 15 minutos antes de la hora programada, el solicitante obtuvo la asignación automática de un conductor.

También se validó la rama de fallo, si al momento de la auto-asignación JIT no había conductores disponibles, la tarea fue marcada como FAILED, con el campo error_message indicando "No hay conductores disponibles". Este caso se reprodujo dejando intencionalmente a todos los conductores inactivos antes de la hora de asignación. El sistema, al no poder crear un viaje, actualizó el objeto programado a FAILED y retuvo la solicitud en estado pendiente.

Tal comportamiento coincide con lo esperado, la tarea JIT fallida actúa como alerta para los administradores, quienes pueden entonces intervenir manualmente. De hecho, en la interfaz de administración de FlotUp, comprobamos que una solicitud en esta situación muestra la opción de buscar conductor manualmente, dado que su auto-asignación programada no se concretó.

Validación PREASSIGN_THEN_REVALIDATE.

La política **PREASSIGN_THEN_REVALIDATE** combina asignación temprana con mecanismos de verificación posterior. Al momento de crear una solicitud agendada bajo esta política, el sistema intenta asignar un conductor disponible de forma inmediata. Si la asignación es exitosa, se genera un objeto *Trip* y se programa una tarea de revalidación (*ScheduledAutoAssign* con acción *REVALIDATE*) para ejecutarse minutos antes de la hora prevista del viaje. Esta revalidación permite confirmar que el conductor

asignado sigue estando disponible. En caso contrario, se intenta reasignar automáticamente un nuevo recurso. Si la asignación inicial falla, no se programa ninguna revalidación.

Caso 1: Conductor preasignado continúa disponible

Se configura una solicitud futura bajo esta política, con al menos un conductor disponible. El sistema asigna de inmediato un VehicleBinding activo, crea el correspondiente Trip y deja programada una tarea de revalidación minutos antes de la hora del viaje. Al llegar el momento, el sistema confirma que el conductor original mantiene su vínculo activo y decide no realizar ningún cambio. La tarea se marca como DONE y el viaje sigue su curso sin alteraciones. Este comportamiento confirma que la revalidación no interfiere innecesariamente cuando las condiciones iniciales se mantienen.

Caso 2: Conductor preasignado ya no disponible.

Este escenario evalúa la capacidad de adaptación del sistema. Se establece una solicitud con asignación inmediata exitosa, pero antes de la revalidación se modifica la disponibilidad del conductor (por ejemplo, se termina su vínculo o se asigna a otro viaje). Al ejecutarse la tarea REVALIDATE, el sistema detecta que el conductor ya no cumple los requisitos, y activa el proceso de reasignación automática.

- Si se encuentra un nuevo conductor, se crea un Trip temporal con el nuevo recurso, se actualiza el Trip original con esa información y el temporal se elimina. La solicitud conserva su aprobación y el usuario no requiere intervención adicional. La tarea se marca como DONE.

- Si no hay reemplazo disponible, el sistema marca la tarea como FAILED, registrando un mensaje de error. El viaje permanece asignado al conductor original, aunque este ya no está disponible, lo cual requiere intervención operativa. La interfaz administrativa refleja este fallo para facilitar la gestión manual.

Caso 3: Asignación inicial fallida

También se contempla la situación en la que no hay ningún conductor disponible en el momento de crear la solicitud. En ese caso, no se genera Trip ni se programa revalidación. La solicitud permanece en estado Pendiente, sin mecanismos de asignación posterior. Este comportamiento es coherente con la implementación actual, aunque representa una limitación: la política no contempla reintentos automáticos si la asignación temprana no se concreta.

Validación de buscar_conductor().

La función buscar_conductor() permite identificar al conductor más adecuado para una solicitud pendiente, mediante un esquema de puntuación que combina múltiples criterios: distancia estimada al punto de inicio (ETA), frescura de la ubicación reportada, carga de trabajo reciente (fairness) y capacidad del vehículo en relación con los requerimientos del viaje. Esta lógica de selección busca priorizar asignaciones eficientes, equilibradas y acordes con las condiciones operativas. La validación considera distintos escenarios representativos para evaluar la coherencia del algoritmo de selección:

Escenario 1: Distancia y frescura de ubicación

Se simula una situación con varios conductores disponibles, algunos con ubicaciones actualizadas y otros con datos más antiguos. Por ejemplo, un conductor se encuentra a 2 km del origen pero con ubicación de hace 30 minutos, mientras que otro está a 3 km con datos recientes (2 minutos). La función penaliza la antigüedad de la ubicación, favoreciendo al conductor con menor frescura de distancia, pero mayor confiabilidad en su posición. El resultado indica que el segundo conductor es seleccionado, evidenciando que el sistema privilegia información actualizada sobre pequeñas diferencias de distancia.

Escenario 2: Carga reciente de trabajo (fairness)

Para este caso, se introducen conductores con diferente nivel de actividad reciente. Uno de ellos presenta múltiples viajes asignados en la última hora, mientras otro no ha sido asignado recientemente. Al ejecutar la función, se observa una penalización proporcional sobre el conductor más activo, priorizando al menos cargado. Este comportamiento refuerza el principio de distribución equitativa de la carga operativa, evitando sobreasignaciones a un mismo recurso.

Escenario 3: Capacidad del vehículo

Se valida que la función respete la restricción de capacidad mínima del vehículo frente a los requerimientos de la solicitud. Para una solicitud que exige cuatro asientos, sólo se consideran candidatos con vehículos de capacidad igual o superior. Conductores con vehículos de menor capacidad quedan automáticamente descartados. El conductor elegido cumple con la condición, demostrando que el algoritmo filtra correctamente los recursos que no satisfacen los parámetros esenciales de seguridad o confort.

Escenario 4: Ausencia de datos de ubicación

Se evalúa el comportamiento del sistema cuando ningún conductor disponible cuenta con ubicación reciente o válida. Ante esta situación, la función activa una estrategia de contingencia, seleccionando aleatoriamente un conductor de entre los habilitados. Aunque esta selección no se basa en proximidad geográfica, el sistema prioriza mantener una respuesta válida siempre que exista al menos un recurso operativo. El resultado confirma que el sistema ofrece una asignación razonable aún en condiciones de datos incompletos.

En términos de eficiencia, cada ejecución de buscar_conductor() toma apenas unos milisegundos, incluso con hasta 100 candidatos en la lista evaluada. Esta eficiencia se debe al diseño del algoritmo, que realiza operaciones simples y escalables por candidato (cálculo de distancia, penalizaciones lineales y filtros básicos). De esta manera, se confirma que el rendimiento es adecuado para flotas de tamaño mediano sin requerir optimización adicional.

6 Impactos de la solución.

6.1 Impacto operativo.

La incorporación de FlotUp permite mejorar la eficiencia en la gestión de viajes diarios. Las tareas que antes se realizaban manualmente ahora están centralizadas en una única

plataforma. La asignación de vehículos se hace de forma automática, lo que reduce tiempos de espera y asegura que cada viaje cuente con un conductor y vehículo adecuados. Esto facilita la coordinación y permite atender más solicitudes con los mismos recursos disponibles.

También se reduce la necesidad de llamadas telefónicas y coordinación manual entre áreas. Anteriormente, era común que los encargados de flota contactaran a conductores uno por uno para asignar viajes o confirmar disponibilidad. Con FlotUp, el sistema se encarga de enviar notificaciones automáticas y mantener actualizada la información para todos los usuarios, lo que disminuye errores de comunicación y acelera la respuesta.

Además, el sistema ayuda a evitar errores comunes en la gestión manual, como asignaciones duplicadas o datos incompletos. Cada solicitud pasa por reglas automáticas que validan fechas, disponibilidad y requisitos mínimos, lo que asegura coherencia en la información registrada. Al asignar conductores de forma automática, también se reduce la posibilidad de decisiones subjetivas o inconsistentes.

Por último, el flujo completo desde la solicitud hasta la ejecución del viaje es más rápido y ordenado. Antes, el proceso incluía varios pasos separados entre diferentes personas o medios. Ahora todo ocurre dentro del sistema: el solicitante registra su pedido, el responsable lo aprueba y el sistema asigna automáticamente el recurso disponible. Esto reduce demoras y permite dar respuesta con mayor agilidad. A través del monitoreo en tiempo real, también es posible seguir la ejecución del viaje y atender cualquier inconveniente que se presente.

6.2 Impacto en trazabilidad y transparencia.

FlotUp permite registrar todas las etapas del proceso de viaje, lo que mejora la trazabilidad y la transparencia en la gestión. Cada acción queda almacenada en el sistema: quién solicitó el viaje, quién lo aprobó, qué conductor y vehículo fueron asignados, los recorridos realizados y los horarios de inicio y término. Esta información permite reconstruir el historial completo de cada traslado, lo que facilita responder a consultas o auditorías con datos concretos sobre cómo se usaron los recursos.

Contar con este registro digital habilita un mejor control del uso de la flota. Los encargados pueden generar informes con indicadores como viajes por departamento, kilómetros recorridos o consumo estimado de combustible. Esto ayuda a identificar posibles irregularidades o patrones que requieran revisión. Además, los datos permiten demostrar públicamente cómo se están utilizando los vehículos municipales, fortaleciendo la rendición de cuentas.

FlotUp también promueve una asignación más justa. Al utilizar un algoritmo basado en criterios técnicos, como cercanía o disponibilidad, se evita la asignación manual que podría generar favoritismos. Así, todos los funcionarios tienen las mismas condiciones al solicitar un viaje, y el sistema mantiene un registro claro de cómo se tomaron las decisiones. Esto mejora la equidad y refuerza la confianza en el proceso.

6.3 Impacto económico y de eficiencia.

El uso de FlotUp permite mejoras económicas y de eficiencia en la gestión de la flota. Uno de los beneficios principales es la reducción de kilómetros recorridos sin pasajeros. Antes, es común que los vehículos vuelvan vacíos o hagan trayectos largos para buscar a un funcionario. Con la asignación automática, el sistema selecciona el vehículo más

cercano, lo que disminuye estos desplazamientos. Además, cuando hay solicitudes con horarios y rutas similares, se pueden combinar en un solo viaje. Esto optimiza los recorridos y reduce el uso innecesario de los vehículos.

Estas mejoras también se reflejan en un menor consumo de combustible y una disminución en el tiempo de conducción. Al optimizar rutas y agrupar traslados, se requiere menos tiempo y recursos para realizar las mismas tareas. Esto permite reducir gastos en combustible, mantenimiento y horas extra del personal. Si bien no se cuantifican los ahorros específicos en este caso, existen referencias de otras experiencias municipales donde estos sistemas logran beneficios importantes.

Además, se mejora el aprovechamiento de la flota disponible. Los vehículos realizan más viajes útiles por jornada y pasan menos tiempo inactivos, lo que evita la necesidad de incorporar nuevas unidades. Esto representa un ahorro en inversión, seguros y mantenimiento, y también ayuda a prolongar la vida útil de los vehículos al evitar sobreuso. Por último, al disminuir los kilómetros recorridos y el consumo de combustible, el sistema contribuye a reducir las emisiones, generando un impacto ambiental positivo que complementa los objetivos de eficiencia institucional.

7 Conclusiones y trabajo futuro.

7.1 Conclusiones generales.

El sistema FlotUp responde a una necesidad concreta en la gestión institucional de vehículos: coordinar viajes de forma eficiente, con trazabilidad completa y criterios objetivos de asignación. A través de su implementación, se demuestra que es posible digitalizar un proceso tradicionalmente manual sin comprometer la flexibilidad ni la capacidad de respuesta del área operativa. La integración de componentes como la asignación dinámica, el registro automático de eventos y el monitoreo georreferenciado, permite organizar el uso de la flota de forma más ordenada, equitativa y transparente.

Durante la validación, se comprueba que el sistema funciona adecuadamente bajo diferentes políticas de asignación y condiciones reales de operación. La plataforma muestra buen desempeño ante solicitudes simultáneas, cambios en la disponibilidad de recursos y escenarios sin cobertura inmediata. Además, el enfoque modular y basado en estándares facilita su mantenimiento y posible integración futura con otras soluciones institucionales.

7.2 Trabajo futuro y mejoras.

Si bien el sistema cumple con los objetivos planteados, existen oportunidades claras para continuar su evolución y mejorar su alcance. Algunas mejoras que se pueden seguir son, por ejemplo:

- Módulo de comunicación con usuarios: permitir al solicitante y al conductor intercambiar información desde la plataforma, a través de notificaciones o mensajería controlada.

- Automatización de mantenimientos preventivos: generar alertas a partir del kilometraje acumulado o de la frecuencia de uso del vehículo.

- Almacenamiento de direcciones georreferenciadas: guardar coordenadas previamente consultadas en la base de datos. Buscando disminuir la dependencia de terceros, mejorar los tiempos de respuesta y reducir costos asociados a consultas repetidas.

8 Referencias.

Arsys. (s.f.). Todo sobre la arquitectura cliente-servidor. Arsys. https://www.arsys.es/blog/todo-sobre-la-arquitectura-cliente-servidor

Django Software Foundation. (2024). Django FAQ – MVC vs. MVT. Django Documentation. https://docs.djangoproject.com/en/5.2/faq/general/#why-doesn-t-django-use-the-standard-mvc-naming-convention

Doonamis. (s.f.). Comparativa Ionic vs React Native vs Flutter: Frameworks multiplataforma. https://www.doonamis.com/ionic-vs-react-native-vs-flutter-comparativa-frameworks-multiplataforma

EDteam. (2023). ¿Por qué usar un ORM? Ventajas y desventajas. https://ed.team/blog/por-que-usar-un-orm-ventajas-y-desventajas

EspiFreelancer. (s.f.). Patrón Modelo-Template-Vista en Django. https://espifreelancer.com/mtv-django.html

Fireplugins. (2023). Google Maps vs OpenStreetMap: ¿Cuál es mejor para tu app?. https://www.fireplugins.com/blog/google-maps-vs-openstreetmap

KeepCoding. (2022). Casos prácticos para usar WebSockets. https://keepcoding.io/blog/casos-practicos-para-usar-websockets

MentoresTech. (s.f.). Convenciones para nombrar endpoints y recursos en APIs REST. https://www.mentorestech.com/resource-blog-content/convenciones-para-nombrar-endpoints-y-recursos-en-apis-rest

Mozilla Developer Network (MDN). (2024). Introducción a Django: Desarrollo web del lado del servidor. https://developer.mozilla.org/es/docs/Learn_web_development/Extensions/Server-side/Django/Introduction

MOX.one. (2023). Comparativa entre Node.js, Laravel y Django para el desarrollo backend. https://mox.one/es/blog/398/comparativa-entre-node-js-laravel-y-django-para-el-desarrollo-backend

Q2B Studio. (2023). Django, Flask y FastAPI: ¿Cuál elegir?. https://www.q2bstudio.com/nuestro-blog/18435/django-flask-y-fastapi-cual-elegir
Red Hat. (s.f.). ¿Qué es una API REST?. https://www.redhat.com/es/topics/api/what-is-a-rest-api

Apidog. (2023). Cómo integrar la API de Google Maps en tu app. https://apidog.com/es/blog/how-to-integrate-google-maps-api-maps-api-3

Wikipedia. (s.f.). Cliente-servidor. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Cliente-servidor

CodeNaEs. (s.f.). ¿Qué son los WebSocket? Comunicación en tiempo real para la web moderna. https://codenaes.com/blog/que-son-los-websocket-comunicacion-en-tiempo-real-para-la-web-moderna

OBS Business School. (2024). Roles, eventos y artefactos en la metodología Scrum. https://www.obsbusiness.school/blog/roles-eventos-y-artefactos-en-la-metodologia-scrum

Atlassian. (s.f.). Guía sobre Scrum. https://www.atlassian.com/es/agile/scrum

Agradecimientos:

Este trabajo representa el cierre de una etapa muy significativa en mi vida.

Agradezco profundamente a mis padres, Norma Díaz y Mario Albornoz, por ser un pilar fundamental y brindarme siempre su apoyo incondicional. A mi hermana, Jocelyn Albornoz, por su compañía y apoyo constante.

Agradezco a Dios por darme la oportunidad de alcanzar esta meta y acompañarme en el camino. Finalmente, gracias a mis amigos por su apoyo, y de manera especial a Paula Santelices, por estar siempre presente.