UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

DEPARTAMENTO DE ELECTROTECNIA E INFORMÁTICA
INGENIERÍA EN INFORMÁTICA

# Desarrollo de sistema BackEnd para la Optimización de Rutas y Gestión de Entregas en Comedores Solidarios de Viña del Mar.

Carlos Donoso Gómez
carlos.donosogo@usm.cl

Óscar Carrasco Vera
Profesor guía

**Resumen:** Tras los incendios masivos ocurridos a inicios de 2024 en Viña del Mar, se evidenciaron graves problemas logísticos en la distribución de alimentos hacia los comedores solidarios. Estos problemas incluían comunicación deficiente, distribución inequitativa y falta de rutas optimizadas, afectando negativamente a los beneficiarios. Debido a ello, surge la oportunidad de desarrollar un sistema BackEnd que mejore la gestión de entregas y otorgue rutas óptimas mediante una arquitectura modular y tecnologías como Django REST Framework, PostgreSQL y MapBox API. Se diseñó una solución que permite gestionar el abastecimiento de los distintos comedores solidarios, optimizando rutas y coordinando las entregas de manera eficiente. El sistema, probado y validado, reduce tiempos de entrega, mejora la equidad en la distribución y proporciona una base escalable para futuras extensiones, marcando un impacto positivo en la comunidad.

**Palabras Clave**: NeoRoute, BackEnd, optimización de rutas, gestión de abastecimiento, comedores solidarios, logística.

## 1. Introducción

### 1.1. Contexto y antecedentes

Tras la catástrofe vivida a inicios del 2024 con los incendios en Viña del Mar, entraron en operación una gran cantidad de comedores solidarios en la comuna para ayudar a los damnificados, pero, dada la repentina situación, el sistema adoptado por la municipalidad de Viña del Mar para abastecerlos, presentaba distintas falencias, desde una comunicación deficiente entre los distintos actores del sistema, lo que provocó una distribución inequitativa de los productos, donde habían comedores que resultaban ser sobre abastecidos, mientras que otros directamente no eran abastecidos en el momento necesario. Además, no existían rutas definidas a seguir para el abastecimiento, lo que provocó largos tiempos de espera en las entregas, causando así que en ocasiones se perdieran alimentos debido al mal estado en que finalmente llegaban, provocando por consecuencia un descontento por parte de la comunidad al sentirse “abandonados” frente a una situación tan crítica como lo es un incendio de ese nivel. A partir de esas falencias surgió la necesidad de mejorar dicha logística de distribución, dando la oportunidad de desarrollar e implementar un software que permita la gestión del abastecimiento de dichos comedores y el control de los productos a entregar. Además, se prevé entregar información sobre las rutas óptimas para reducir costos y optimizar los tiempos de entrega, satisfaciendo así las necesidades de la comunidad y dejando preparada a la municipalidad en caso de que ocurra una situación similar o peor en el futuro.

Este proyecto se desarrollará de manera grupal, dividiendo las responsabilidades entre los integrantes para garantizar un desarrollo eficiente y organizado. Cada miembro del equipo asumió un rol específico, contribuyendo desde distintas áreas clave:

- UI/UX: Un integrante será responsable de diseñar la experiencia de usuario y la interfaz gráfica de las distintas aplicaciones, asegurando que la plataforma web y la aplicación móvil sean intuitivas y sencillas de utilizar para los usuarios finales.

- Interfaz web: Otro integrante desarrollará la plataforma web que utilizarán los coordinadores para gestionar los comedores solidarios, planificar rutas y supervisar las entregas.

- Aplicación móvil: Otro integrante desarrollará una aplicación móvil que utilizarán los conductores, permitiendo visualizar las rutas asignadas y marcar la finalización de dicha ruta en tiempo real.

- BackEnd: Mi responsabilidad principal es el desarrollo del sistema BackEnd, el cual actuara como el núcleo del sistema. En esta función, me encargaré de diseñar la base de datos, implementar la lógica de negocio, desarrollar una API RESTful para la comunicación entre las distintas aplicaciones del sistema y la integración de la API de MapBox para la optimización de rutas.

### 1.2. Definición del problema.

Tras el incidente de los incendios masivos vividos a inicios de este año 2024, los comedores solidarios en Viña del Mar entraron en colapso debido a diversos factores, entre los que se encuentran:

- **Comunicación deficiente:** Los comedores solidarios realizan pedidos mediante llamados telefónicos a una única persona encargada en la municipalidad, lo que genera un cuello de botella y una gran falta de coordinación. Esto lleva a que algunos comedores reciban más alimentos de los necesarios, otros reciban menos, y en algunos casos, los pedidos no lleguen en absoluto, provocando así un gran descontento de las personas que “viven” en este momento de la ayuda de dichos comedores.
- **Distribución inequitativa:** Producto de la comunicación deficiente y la falta de un sistema estructurado para gestionar el abastecimiento de los comedores provoca una distribución desigual de los alimentos, afectando la capacidad de los comedores para atender a sus beneficiarios, donde hay casos en los que algunos comedores son sobre abastecidos y otros los cuales son “olvidados” y no son abastecidos, lo cual es una situación bastante grave, ya que hay damnificados los cuales su única ayuda recibida es la proporcionada por los comedores solidarios.
- **Falta de rutas óptimas:** No existen rutas predeterminadas y no todos los conductores utilizan aplicaciones de mapas para obtener rutas óptimas y así distribuir los productos de una manera eficiente, debido a eso, existen situaciones donde debido a los largos tiempos de espera y adicionalmente las condiciones climáticas del día, resultaban en la pérdida de algunos productos.

### 1.3. Breve descripción sobre la propuesta de solución

La solución propuesta consiste en desarrollar una plataforma web para los coordinadores, desde donde gestionarán los distintos comedores solidarios (puntos de entrega) y, así, sectorizarlos con sus respectivos datos, nombre, encargado, sector, etc. La plataforma web también permite realizar planificaciones semanales respecto al abastecimiento de los comedores por sectores, donde se generará una ruta óptima, se asociará un envío (lista de productos) proveniente del inventario y un conductor, quien será el responsable de seguir la ruta a cada sector correspondiente a una planificación diaria.

Por otro lado, se desarrollará una aplicación móvil diseñada específicamente para los conductores. A través de esta aplicación, podrán visualizar todas las rutas asociadas a él, de manera que al seleccionar cualquier ruta comenzará la navegación asistida. Una vez finalizada la ruta, el sistema notificará a los coordinadores y a la unidad de gestión de inventario que la ruta ha finalizado.

El rol del BackEnd es crucial en esta implementación, ya que será responsable de gestionar la lógica de negocio del sistema, lo que implica permitir la creación y edición de planificaciones semanales y diarias respecto al abastecimiento de los comedores solidarios, la asignación de conductores, la comunicación con la API de MapBox para la generación de rutas óptimas y la coordinación entre la plataforma web y la aplicación móvil.

El proyecto, como se mencionó anteriormente, incluirá el desarrollo un sistema de gestión de los distintos puntos de entrega (comedores solidarios) y sectorización, la asignación de rutas en tiempo real y la integración con el sistema de inventario. Sin embargo, no se incluirá la gestión ni el seguimiento en tiempo real de los vehículos.

### 1.4. Objetivos Generales y Específicos de la Tesina

**Objetivo General**: Diseñar e implementar la parte BackEnd dentro de un sistema que gestionará el abastecimiento de los comedores solidarios de la comuna de Viña del Mar.

**Objetivos Específicos**:

- Diseñar e implementar la base de datos del sistema.
- Implementar un sistema de autenticación y autorización basado en JWT.
- Diseñar y documentar una API RESTful para la gestión de comedores, usuarios, planificaciones y rutas.
- Integrar API de MapBox para la utilización de mapas y generación de rutas óptimas.

### 1.5. Justificación de proyecto

La implementación de este sistema resolverá las ineficiencias actuales en la distribución de productos, logrando que todos los comedores solidarios reciban los suministros necesarios de manera equitativa y a tiempo. Además, mejorará la coordinación entre los comedores y la municipalidad, permitiendo una gestión más transparente y efectiva de los recursos disponibles.

Beneficios:

- **Mejora en la Eficiencia Operativa:** Al optimizar la comunicación y la asignación de recursos, la distribución de productos será más eficiente, reduciendo el desperdicio y asegurando que las necesidades de cada comedor sean satisfechas.
- **Reducción de Errores:** Un sistema centralizado reducirá los errores en la comunicación y la gestión de entregas, asegurando que cada solicitud se procese correctamente.
- **Ahorro de Tiempo y Recursos:** La optimización de rutas permitirá a los conductores completar sus entregas en menos tiempo, utilizando menos combustible y reduciendo los costos operativos.
- **Transparencia y Control:** La capacidad de monitorear las entregas en tiempo real proporcionará un mayor control sobre el proceso de distribución, permitiendo una respuesta rápida ante cualquier problema.

- **Mayor Seguridad:** La utilización de JWT (Json Web Token), asegura que solo los usuarios con credenciales válidas pueden acceder al sistema y su información, protegiendo así la información sensible de terceros no autorizados.

- **Escalabilidad:** El sistema está diseñado para adaptarse a un aumento significativo en el volumen de datos y usuarios, asegurando que el rendimiento no se vea afectado, además está preparado para adaptarse fácilmente a nuevas funcionalidades.

### 1.6. Metodología

Para organizar las actividades en el desarrollo de este sistema BackEnd se utilizará la metodología ágil scrum, lo que permitirá gestionar el desarrollo de manera iterativa e incremental, trabajando en sprints, con entregas parciales y funcionales en cada iteración, lo que asegurará una mejora continua y que además permitirá hacer ajustes según las necesidades del cliente y los resultados obtenidos en cada iteración [1].

Además, dentro de este marco ágil se implementarán prácticas de Programación Extrema (XP), que se centran en mejorar la calidad del software y la capacidad de respuesta a cambios. Dentro de las practicas que se incluyen en XP [2], están:

- **Integración continua:** El código será integrado de manera constante en un entorno común, lo que permitirá detectar y resolver problemas rápidamente.
- **Refactorización continua:** Se realizarán mejoras continuas en el código para mantenerlo limpio y eficiente, sin dejar de lado la funcionalidad.
- **Pequeñas entregas:** Las funcionalidades del sistema, como la gestión de planificaciones semanales, diarias, usuarios y rutas, se entregarán de manera continua, asegurando la retroalimentación constante del cliente.

El proyecto se llevará a cabo en tres sprints planificados, cada uno con objetivos específicos que buscan garantizar el desarrollo progresivo y eficiente del sistema propuesto. La organización de los sprints es la siguiente:

– **Sprint 1: Creación de la base de datos, modelos y serializadores.**

Objetivos principales:

- Diseño e implementación de la base de datos:
    - Crear las tablas necesarias: usuarios, sectores, comedores solidarios, planificaciones semanales y diarias, etc.
    - Definir relaciones entre las tablas.
- Implementación de modelos y serializadores en django rest framework:
    - Definir los modelos para las entidades (usuarios, sectores, comedores solidarios, planificaciones semanales y diarias, etc.)
    - Crear serializadores para las interacciones con la API (serializadores para cada entidad y las relaciones).
- Entrega del sprint:
    - Base de datos implementada y conectada.
    - Modelos y serializadores completos en django rest framework.

– **Sprint 2: Autenticación, lógica de negocio y comunicación entre web y móvil.**

  Objetivos Principales:

  - Implementación del sistema de autenticación y autorización (JWT):
    - Crear un sistema de autenticación basado en JWT.
    - Configurar roles y permisos para usuarios (coordinadores y conductores).
  - Integración de la lógica de negocio:
    - Implementar la lógica de negocio para la creación y asignación de planificaciones diarias a conductores.
  - Asignación de rutas con MapBox:
    - Integrar MapBox API para generar rutas optimizadas.
  - Comunicación entre la web y la aplicación móvil:
    - Verificar que la web gestione las planificaciones semanales y diarias y la aplicación móvil muestre las rutas y planificaciones diarias asignadas a los conductores.
  - Entrega del sprint:
    - Sistema de autenticación y autorización implementado.
    - Lógica de negocio funcionando: creación de planificaciones semanales y diarias y asignación de conductores.
    - Comunicación entre la web y la aplicación móvil implementada.

– **Sprint 3: Optimización, testing e integración con el sistema de inventario**

  Objetivos principales:

  - Optimización del sistema:
    - Refinar la lógica de negocio y mejorar la eficiencia de las rutas.
    - Optimizar el rendimiento de la base de datos.
  - Testing del sistema:
    - Realizar pruebas unitarias y de integración.
    - Asegurar que la web y la app móvil funcionen correctamente.
  - Integración con el sistema de inventario:
    - Implementar la comunicación con el sistema de inventario externo.
    - Sincronizar los envíos disponibles en el inventario con las planificaciones diarias.
  - Entrega del sprint:
    - Sistema optimizado y probado.
    - Integración con el sistema de inventario lista.
    - Documentación de la optimización y pruebas realizada.

Por otro lado, la coordinación con el equipo se realizará siguiendo los principios de la metodología ágil SCRUM, permitiendo una colaboración constante y eficiente durante todas las etapas del proyecto. A continuación, se detallan algunas prácticas clave para asegurar la alineación entre los miembros del equipo:

- **Daily Scrum:** Se llevarán a cabo reuniones breves recurrentemente donde cada miembro compartirá:
  - Lo que logró el día anterior.
  - Las tareas que planea realizar.
  - Los impedimentos o desafíos que podrían afectar su progreso.

- **Planificación de Sprints:** Antes de iniciar cada sprint, el equipo se reunirá para definir los objetivos específicos de dicho sprint, priorizando las tareas en el Product Backlog. De esta manera aseguraremos que todos los miembros entiendan sus responsabilidades y cómo sus actividades contribuyen al objetivo general.

- **Sprint Review:** Al finalizar cada sprint, el equipo presentará los avances logrados a los clientes. Estas revisiones servirán Para obtener retroalimentación y realizar ajustes en función de las necesidades detectadas.

Gracias a este enfoque, se logrará mantener una comunicación abierta y un flujo de trabajo organizado, permitiendo al equipo adaptarse rápidamente a los cambios y asegurar la entrega de un producto mínimo viable que cumpla con los objetivos y expectativas del proyecto.

### 1.7. Breve descripción de la organización del informe en capítulos

El informe se divide en capítulos para facilitar su lectura y comprensión:

- **Introducción**: Expone el contexto, antecedentes y definición del problema, junto con los objetivos y justificación del proyecto.
- **Marco Teórico**: Detalla los fundamentos técnicos y tecnológicos relevantes para el desarrollo del sistema.
- **Metodología**: Explica el enfoque ágil utilizado y el diseño progresivo del sistema mediante sprints.
- **Diseño e Implementación**: Describe los componentes, su arquitectura y la integración con otros sistemas.
- **Pruebas y Validación**: Presenta las estrategias de pruebas aplicadas y sus resultados.
- **Conclusiones y Recomendaciones**: Resalta los logros del proyecto y su impacto, con sugerencias para futuros desarrollos.

## 2. Marco Teórico

### 2.1. Fundamentos del desarrollo BackEnd

El BackEnd es el “corazón” de cualquier sistema de gestión de datos y operaciones, encargado de procesar, almacenar y gestionar los datos de manera eficiente. Desde sus inicios, el desarrollo BackEnd ha sido fundamental para estructurar la lógica de negocios y el manejo de bases de datos en los sistemas informáticos. Este enfoque permite que los sistemas puedan funcionar de manera organizada, procesando grandes volúmenes de información y asegurando que los datos sean accesibles de manera eficiente [3].

En el contexto del Proyecto de Distribución Para Comedores Solidarios, el BackEnd se encarga de gestionar entregas, coordinar la asignación de conductores, y manejar en tiempo real las rutas de entrega. Estas operaciones críticas se llevan a cabo a través de una API, que facilita la comunicación entre las diferentes capas del sistema [4].

Para el desarrollo de un sistema BackEnd y/o API RESTful, existe una amplia variedad de tecnologías y frameworks, donde cada uno de ellos cuentan con sus propias ventajas y contras. Algunas de las tecnologías más populares y relevantes para el desarrollo BackEnd son [5, 6, 7]:

**Frameworks:**

- **Django REST Framework (DRF):** Es un framework robusto y popular basado en Django, que facilita la creación de APIs RESTful. Es ampliamente utilizado para proyectos de gran escala debido a su capacidad para manejar la gestión de datos, seguridad y autenticación de manera eficiente.
    - Pros:
        - Facilita la creación de APIs escalables y seguras para proyectos de gran escala.
        - Gestión eficiente de datos y rápida implementación gracias a sus herramientas integradas.
        - Gran comunidad de soporte y documentación extensa.
        - Integración perfecta con Django, lo que facilita el uso de ORM y otras funcionalidades de Django.
    - Contras:
        - Puede ser excesivo para proyectos pequeños o sencillos, ya que viene con muchas características que no siempre son necesarias.
        - Curva de aprendizaje moderada, especialmente para quienes no están familiarizados con Django.
- **Laravel**: Es un framework PHP ampliamente utilizado que ofrece una estructura organizada para crear aplicaciones web y APIs. Su enfoque en la simplicidad y el desarrollo rápido lo convierte en una opción popular para desarrolladores que buscan una solución sólida con características avanzadas como autenticación, manejo de rutas y respuestas JSON.
    - Pros:
        - Facilita la creación de APIs con rutas bien estructuradas y respuestas en JSON.
        - Soporte nativo para autenticación de tokens, lo que mejora la seguridad de las APIs.
        - Excelente para el desarrollo rápido debido a su sencilla configuración y herramientas integradas.
        - Amplia comunidad y documentación detallada.
    - Contras:
        - Mayor consumo de recursos en comparación con otros frameworks ligeros.
        - Laravel es más adecuado para proyectos medianos o grandes; puede ser una sobrecarga para aplicaciones más simples.
- **Flask**: Es un microframework en Python que permite construir aplicaciones web y APIs de manera rápida y eficiente. Ofrece una estructura mínima, lo que permite al desarrollador tener mayor control sobre los componentes que se agregan al proyecto, siendo ideal para proyectos pequeños o prototipos.
    - Pros:
        - Framework ligero y flexible, que permite añadir solo las extensiones necesarias.
        - Ideal para proyectos pequeños o cuando se requiere rapidez en la implementación.

- Permite gran control sobre la estructura de la aplicación y es fácil de usar para desarrolladores con experiencia.
- Extensiones como Flask-RESTful y Flask-Swagger permiten una rápida creación de APIs.

- Contras:
  - Al ser un microframework, requiere más configuración manual que frameworks más completos como Django o Laravel.
  - Menos adecuado para proyectos grandes o complejos, ya que necesita muchas extensiones para alcanzar la funcionalidad completa.

**Base de Datos:**

- **MongoDB**: Es una base de datos NoSQL basada en documentos, popular por su capacidad de manejar datos no estructurados y su soporte nativo para operaciones geoespaciales mediante el formato GeoJSON, lo que la hace ideal para aplicaciones con datos dinámicos.
  - Pros:
    - Soporte nativo para datos geoespaciales, útil para aplicaciones que necesitan seguimiento en tiempo real de ubicaciones dinámicas.
    - Flexibilidad para manejar grandes volúmenes de datos no estructurados y variados sin necesidad de esquemas rígidos.
    - Escalabilidad horizontal fácil de implementar para proyectos con crecimiento rápido en la cantidad de datos.
    - Muy eficiente para consultas de alta velocidad en grandes volúmenes de datos.
  - Contras:
    - No ofrece transacciones ACID completas en comparación con las bases de datos relacionales, lo que puede ser un inconveniente para aplicaciones que requieren operaciones de alta consistencia.
    - La flexibilidad de los esquemas puede llevar a problemas de inconsistencia si no se administra correctamente.
    - Menor rendimiento en operaciones altamente relacionales debido a la falta de integridad referencial.
- **PostgreSQL**: Es una base de datos relacional de código abierto ampliamente utilizada, conocida por su estabilidad y soporte avanzado para tipos de datos complejos y operaciones geoespaciales cuando se integra con la extensión PostGIS.
  - Pros:
    - Soporte para operaciones geoespaciales a gran escala con la extensión PostGIS, ideal para aplicaciones que requieren manejo de rutas y ubicaciones.
    - Altamente escalable y robusto, con soporte para grandes volúmenes de datos y concurrencia.
    - Cumple con el estándar SQL y ofrece transacciones ACID, lo que garantiza la consistencia y fiabilidad de los datos.
    - Amplia comunidad y soporte técnico.
  - Contras:
    - Puede ser más complejo de configurar y optimizar en comparación con otras bases de datos más simples.
    - Requiere mayor poder computacional para manejar grandes volúmenes de datos geoespaciales, especialmente si no se optimiza adecuadamente.

**Geolocalización y rutas:**

- **Google Maps API**: Es una de las herramientas más completas para la visualización de mapas y la generación de rutas, que ofrece una amplia gama de servicios como geocodificación, cálculo de rutas y representación de mapas interactivos.
    - Pros:
        - Amplia gama de servicios y herramientas, incluyendo geocodificación, cálculo de rutas, y visualización de mapas.
        - Extensa documentación y soporte global, lo que facilita su implementación en proyectos de cualquier escala.
        - Precisión y actualización constante de los datos geográficos.
        - Integración con una variedad de aplicaciones y plataformas.
    - Contras:
        - Costos elevados a medida que el uso de la API escala, especialmente en proyectos con grandes volúmenes de solicitudes.
        - Menos personalizable en términos de diseño y funcionalidad comparado con otras plataformas como MapBox.
        - Dependencia total de la infraestructura de Google, lo que puede generar problemas en casos de caídas del servicio.
- **MapBox API**: Es una plataforma de geolocalización que permite una mayor flexibilidad y personalización en la creación de mapas interactivos, así como servicios avanzados para la generación de rutas optimizadas.
    - Pros:
        - Mayor flexibilidad para personalizar la apariencia de los mapas y agregar funcionalidades específicas.
        - Ofrece servicios robustos para la geolocalización, generación de rutas optimizadas y visualización de mapas interactivos.
        - Modelos de precios más flexibles en comparación con Google Maps API.
        - Buena integración con aplicaciones de mapas que requieren una interfaz única o experiencias de usuario personalizadas.
    - Contras:
        - Menos extensiva en términos de documentación y soporte en comparación con Google Maps.
        - Requiere más configuración inicial para proyectos que necesiten funcionalidades avanzadas.
        - Precisión de los datos y la cobertura global pueden no ser tan extensas como la de Google Maps en algunas áreas.

### 2.2. Arquitectura de software

La arquitectura del sistema estará basada en una arquitectura que separará claramente las responsabilidades del sistema. Estas capas son:

- **Capa de presentación**: En esta capa se encuentran la interfaz web y móvil que interactúan con el usuario. La comunicación con el BackEnd se realiza mediante solicitudes HTTP utilizando formatos JSON.
- **Capa de negocios:** En esta capa se encuentra el BackEnd implementado en Django Rest Framework y la conexión con la API de MapBox, donde se implementa la lógica de gestión de

comedores y planificaciones, asignación de rutas y autenticación de usuarios entre otras funcionalidades del sistema.

- **Capa de Datos:** En esta capa se encuentra la base de datos relacional PostgreSQL, donde se manejará el almacenamiento, edición y eliminación de los distintos datos del sistema como usuarios, planificaciones semanales y diarias, comedores solidarios, etc.

A continuación, en la **Figura 1**, se observa de manera más sencilla como interactúa el software con los distintos componentes.

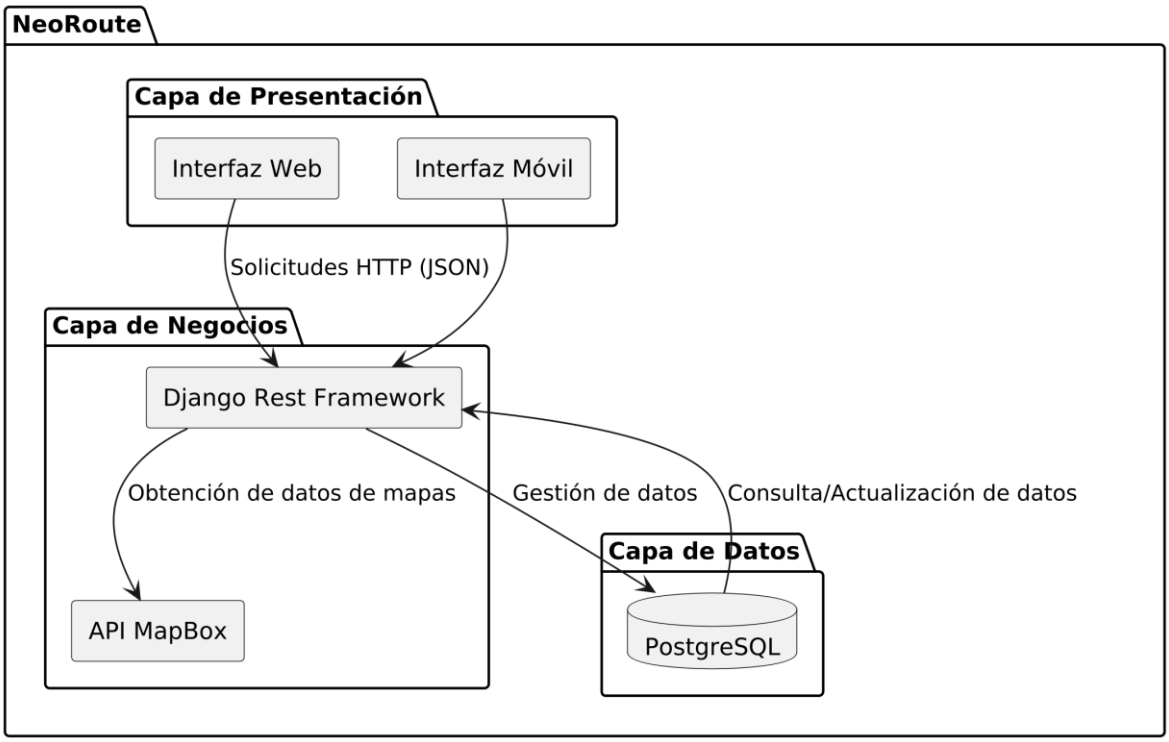


**Figura 1.** Arquitectura del sistema general NeoRoute

Gracias a esta arquitectura, se garantiza escalabilidad, modularidad y facilidad para integrar nuevos componentes en el futuro, además, se utilizarán los patrones de diseño observer y Template Method, principalmente porque son fundamentales al momento de implementar una comunicación en tiempo real entre la aplicación móvil y el BackEnd, dado que:

- **El patrón observer** establece una relación de suscriptor-publicador en la que el sujeto notifica automáticamente a sus observadores (suscriptores) sobre eventos o cambios de estado, en este caso, es esencial enviar la notificación al momento de asignar una ruta a un conductor, de manera que, al momento de realizar la asociación, el conductor en tiempo real obtenga la ruta.
- **El patrón Template Method** define el esqueleto de un algoritmo en un método base, permitiendo que las subclases sobrescriban partes específicas del proceso sin alterar su estructura general, permitiendo así que, al momento de asociar un conductor, lo que en otras palabras es que, al actualizar una planificación diaria, se pueda implementar la lógica de notificar al conductor al momento de ser asociado.

### 2.3. Tecnologías y frameworks utilizados

**Python**: Python es el lenguaje de programación que fue seleccionado principalmente por su versatilidad y eficiencia, algo completamente necesario en este proyecto donde se trabajan con datos geoespaciales y rutas, ya que cuenta con un extenso ecosistema de bibliotecas las cuales en caso de ser necesario facilitaría la implementación de cálculos matemáticos complejos para trabajar con estos datos geoespaciales o las rutas. Además, es el lenguaje base de django rest framework, lo que garantiza una integración fluida en la creación de la API de NeoRoute, y su compatibilidad con herramientas como la API de Mapbox y Websockets, permitiendo así una implementación eficiente y escalable.

**Django REST Framework**: Se seleccionó este framework debido a su robustez y facilidad para desarrollar APIs RESTful escalables, algo que es muy relevante en este proyecto debido al volumen de datos con el que es posible llegar a trabajar. Se utilizará para desarrollar la API, permitiendo gestionar los diferentes componentes del sistema de manera eficiente y segura.

**PostgreSQL y PostGIS**: Se seleccionó PostgreSQL como base de datos debido a su capacidad de trabajar con datos geoespaciales a través de la extensión PostGIS, lo cual al estar trabajando con rutas y puntos de entrega (en este caso comedores solidarios) es una extensión clave para el sistema, además PostgreSQL permite manejar grandes volúmenes de datos sin comprometer el rendimiento. Estas tecnologías se emplearán para gestionar la información de los pedidos y las ubicaciones geoespaciales, necesarias para la optimización de rutas.

**MapBox API**: Se eligió la API de MapBox principalmente porque dentro de las APIs que ayudan al despliegue de mapas y generación de rutas, MapBox es la más flexible a nivel de personalización y es la más económica frente a las otras opciones. La API será Integrada principalmente para generar rutas óptimas que ayuden a los conductores realizar las rutas de manera más eficiente.

**Postman**: Postman fue seleccionado como herramienta para probar y documentar la API debido a su facilidad de uso, permitiendo validar las solicitudes y respuestas de la API de manera eficiente, asegurando la correcta comunicación entre el BackEnd y las interfaces web y móvil.

**Wscat**: Se eligió Wscat para probar y depurar la comunicación en tiempo real mediante WebSockets. Esta herramienta permite enviar y recibir mensajes desde un cliente simulado, lo que es crucial para validar la interacción entre el BackEnd y la aplicación móvil en escenarios como la asignación de rutas a los conductores.

**Github**: Github fue seleccionado como la plataforma de control de versiones debido a que es la más popular dentro de la industria de desarrollo de software y su capacidad de facilitar el trabajo colaborativo en proyectos grupales como lo es NeoRoute. Con sus herramientas de gestión de versiones, permite mantener un historial detallado de cambios realizados en el código, lo que resulta esencial para garantizar la trazabilidad y resolución de problemas entre el equipo desarrollador.

## 3. Diseño e implementación.

### 3.1. Diseño de Componentes

El diseño de componentes es una etapa esencial en el desarrollo de un software, ya que es el que define como interactúan entre si los distintos componentes para satisfacer los requerimientos funcionales y no funcionales. En el caso del proyecto NeoRoute, el sistema se compone de diversos componentes que trabajan en conjunto para optimizar la gestión de rutas y la entrega de productos a los comedores solidarios.

A continuación, se presenta como están organizadas las partes principales del sistema y cómo trabajan juntas para cumplir con sus objetivos. Además, se detalla como el BackEnd conecta estas partes y garantiza que el sistema funcione correctamente, desde recibir las solicitudes de los usuarios hasta realizar las consultas para la generación de las rutas y coordinar las entregas.

### 3.1.1. Arquitectura y estructura de los componentes.

Como se mencionó anteriormente, el sistema de NeoRoute está diseñado bajo una arquitectura modular que separa claramente las responsabilidades entre sus principales capas: presentación, negocio y datos. Este enfoque asegura una mayor escalabilidad, mantenibilidad y facilidad para integrar nuevos componentes en el futuro:

- **Capa de presentación**: Incluye la plataforma web para los coordinadores y la aplicación móvil para los conductores. Ambas interfaces permiten a los usuarios interactuar con el sistema para gestionar los comedores solidarios, planificar las rutas y supervisar el cumplimiento de dicha ruta. La comunicación entre la capa de presentación y el BackEnd se realiza mediante solicitudes HTTP y respuestas en formato JSON, asegurando un intercambio de datos eficiente y seguro.

- **Capa de negocio**: Representada por el BackEnd del sistema, desarrollado en Django Rest Framework. Esta capa es donde se implementa y contiene la lógica de negocio necesaria para la gestión de los datos, generar rutas óptimas mediante la API de MapBox, y coordinar la información entre la plataforma web, la aplicación móvil y el sistema de inventario externo. El BackEnd además maneja la autenticación y autorización de usuarios, garantizando que las operaciones se realicen de manera segura y controlada.

- **Capa de datos**: Utiliza PostgreSQL con la extensión PostGIS para almacenar y gestionar información estructurada, como los datos de los comedores, usuarios y planificaciones.

Basado en lo anterior, el BackEnd actúa como un puente que conecta las diferentes capas y servicios externos del sistema. A través de una API RESTful, el BackEnd permite que las solicitudes de los usuarios (tanto de la web como de la aplicación móvil) sean procesadas, validando la información y aplicando la lógica de negocio antes de interactuar con la base de datos o servicios externos.

Además, la integración con la API de MapBox permite al BackEnd enviar una lista de coordenadas de los puntos de entrega y recibir rutas optimizadas, que son desplegadas en la web y posteriormente enviadas a los conductores en tiempo real mediante WebSockets. Asimismo, la comunicación con el sistema de inventario garantiza que las planificaciones diarias incluyan la lista de productos necesarios para cada ruta, sincronizando la información con ambas plataformas.

A continuación, en la **Figura 2** se muestra cómo es la arquitectura general del sistema.

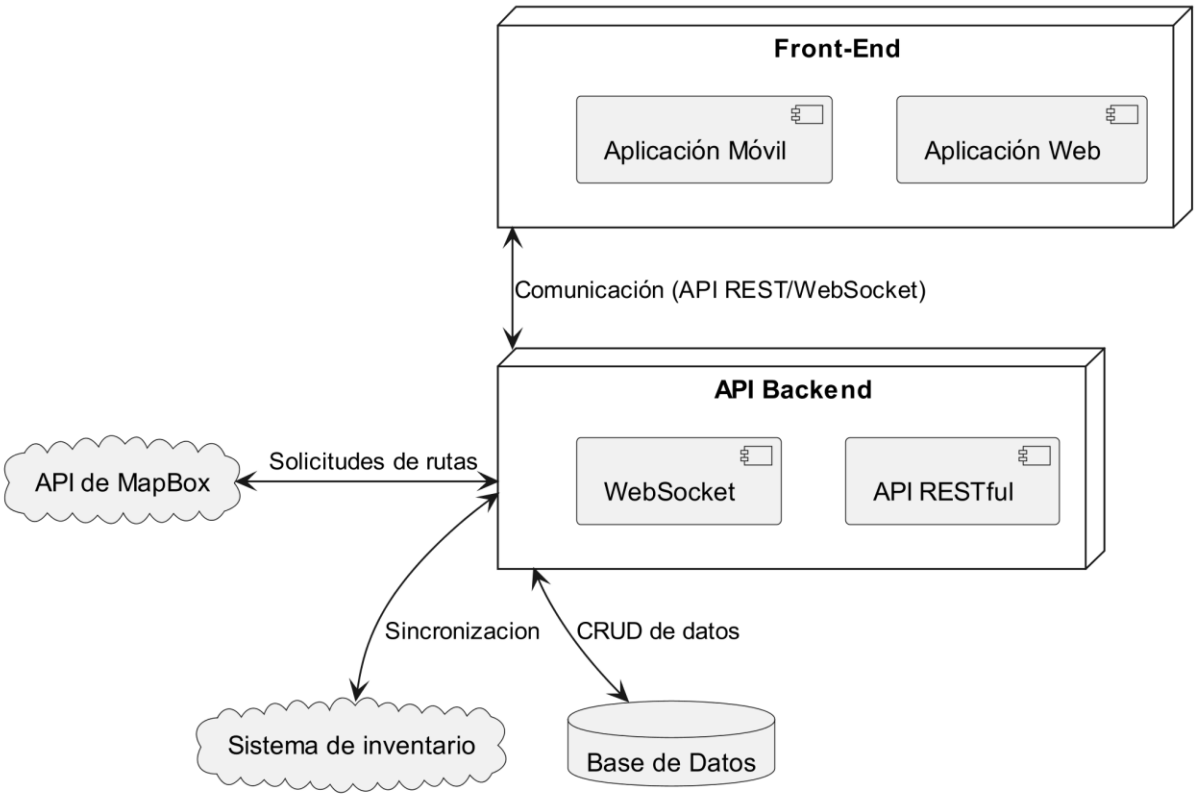


**Figura 2.** Arquitectura de NeoRoute

Basado en lo anterior, el BackEnd de NeoRoute se compone de los siguientes componentes principales:

- **Autenticación**: Se utilizará JSON Web Tokens (JWT) para gestionar la autenticación de usuarios y los permisos de manera segura.
- **API RESTful**: Facilita la comunicación entre la interfaz de usuario y el BackEnd, asegurando un intercambio de datos eficiente y seguro.
- **MapBox API**: Proporciona herramientas para la visualización de mapas interactivos y para la generación de rutas óptimas.
- **Websockets**: Establece una comunicación en tiempo real entre el BackEnd y la aplicación móvil, este componente se utilizará únicamente al momento de asociar un conductor a una planificación diaria (que contiene la ruta a seguir), de modo que el conductor obtenga la planificación diaria asociada en tiempo real.
- **Base de Datos**: Es donde se almacena de manera estructurada la información del sistema (usuarios, comedores solidarios, sectores, planificaciones semanales, etc.).

Tal como se observa en la **Figura 3**, los componentes mencionados interactúan de manera conjunta para garantizar la funcionalidad del sistema.

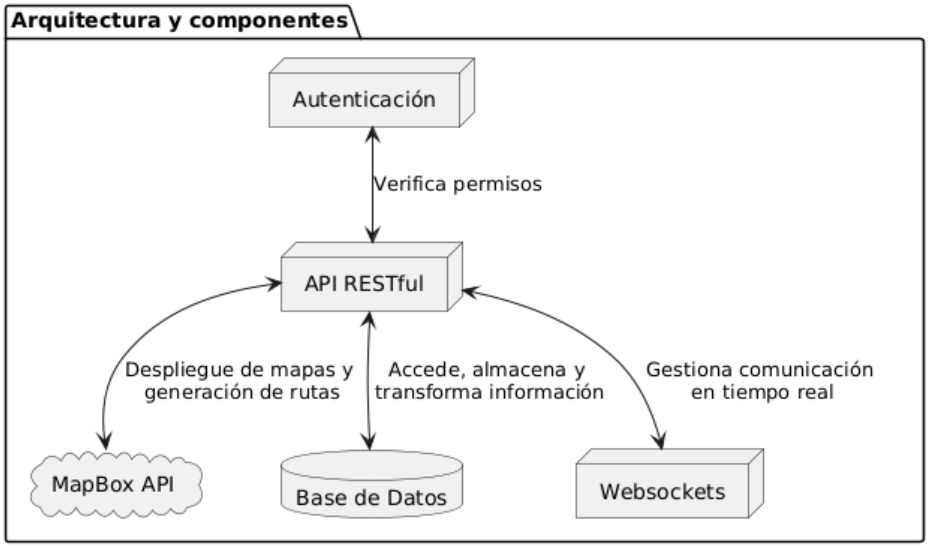


**Figura 3.** Arquitectura y componentes del BackEnd

### 3.1.2. Diseño Funcional.

Ya que la propuesta de solución de NeoRoute propone la creación de dos aplicaciones distintas, las cuales son, una plataforma web para los coordinadores y una aplicación móvil para los conductores, estos pueden realizar distintas acciones en cada aplicación, donde el coordinador en la plataforma web puede:

- Iniciar sesión.
- Crear un nuevo comedor solidario y sectorizarlo.
- Crear un nuevo sector.
- Crear una planificación semanal.
- Crear planificaciones diarias dentro de una semanal.
- Asociar un envío del sistema de inventario (lista de productos) a una planificación diaria.
- Asociar un conductor para cada planificación diaria.

Por otro lado, el conductor en la aplicación móvil puede realizar las siguientes acciones:

- Iniciar sesión.
- Visualizar las planificaciones asociadas a él.
- Seguir la ruta de una de las planificaciones diarias.
- Finalizar la ruta de una de las planificaciones diarias.

Las principales funcionalidades del sistema se aprecian en el siguiente diagrama de casos de uso.

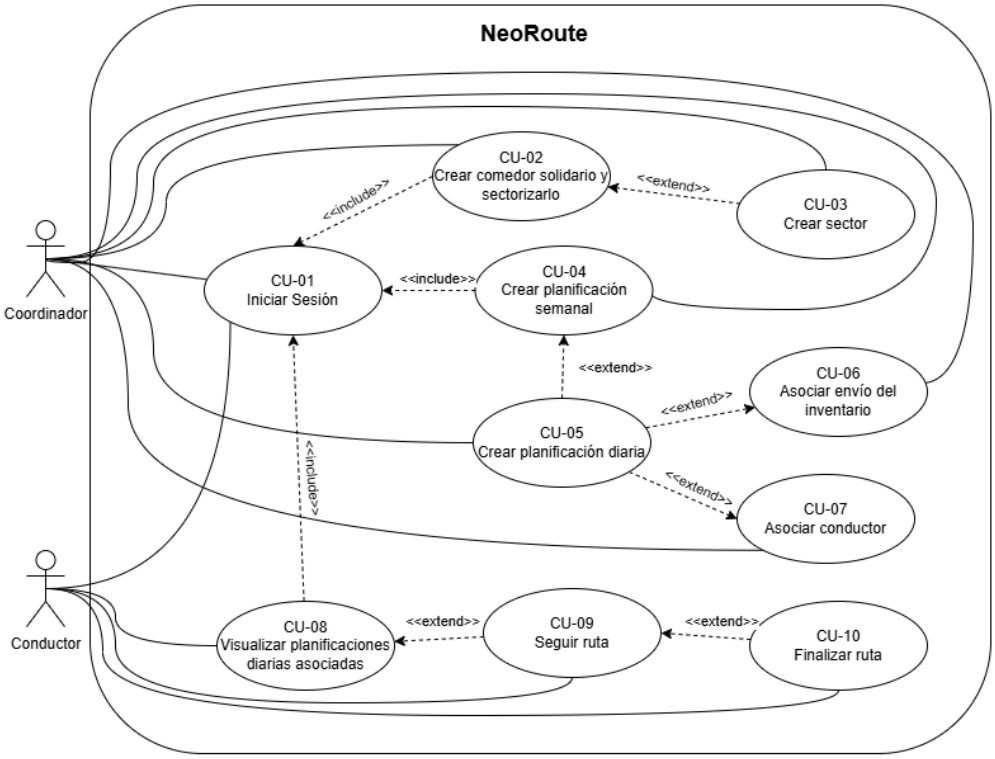


**Figura 4**. Diagrama general de Casos de Uso

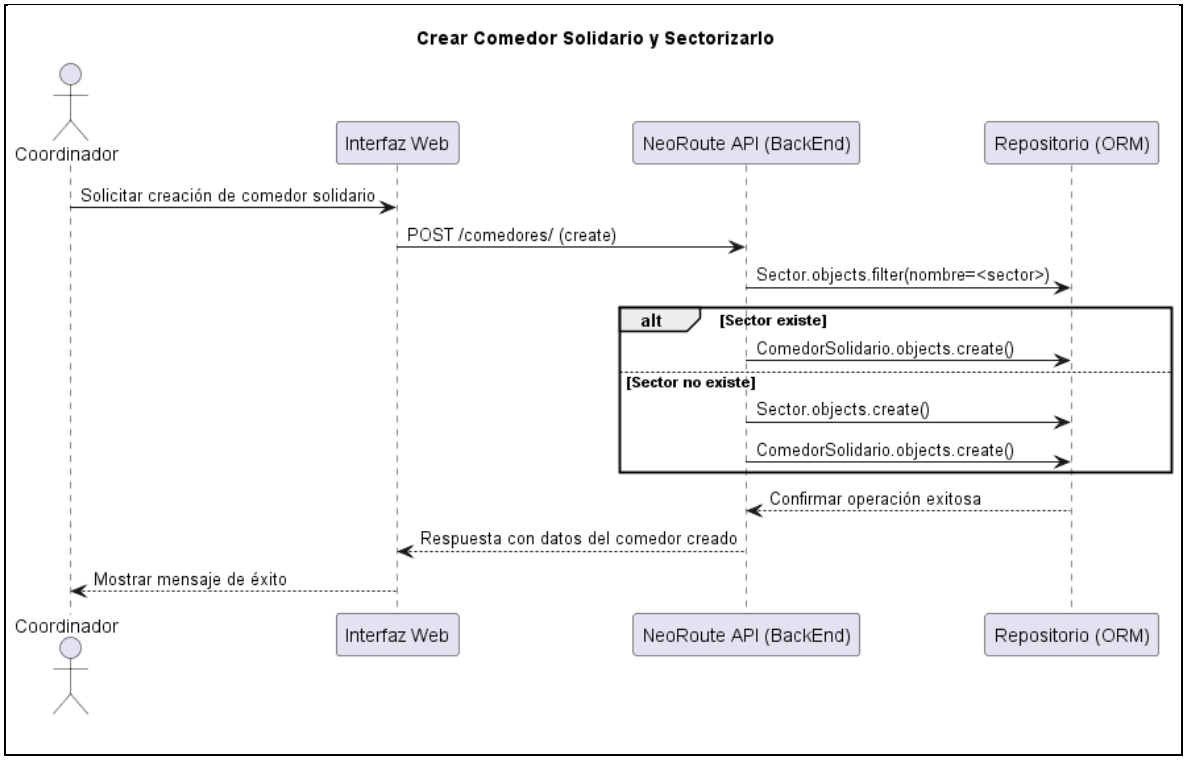


**Figura 5.** Diagrama de Secuencia: CU02 - Crear nuevo comedor solidario y sectorizarlo

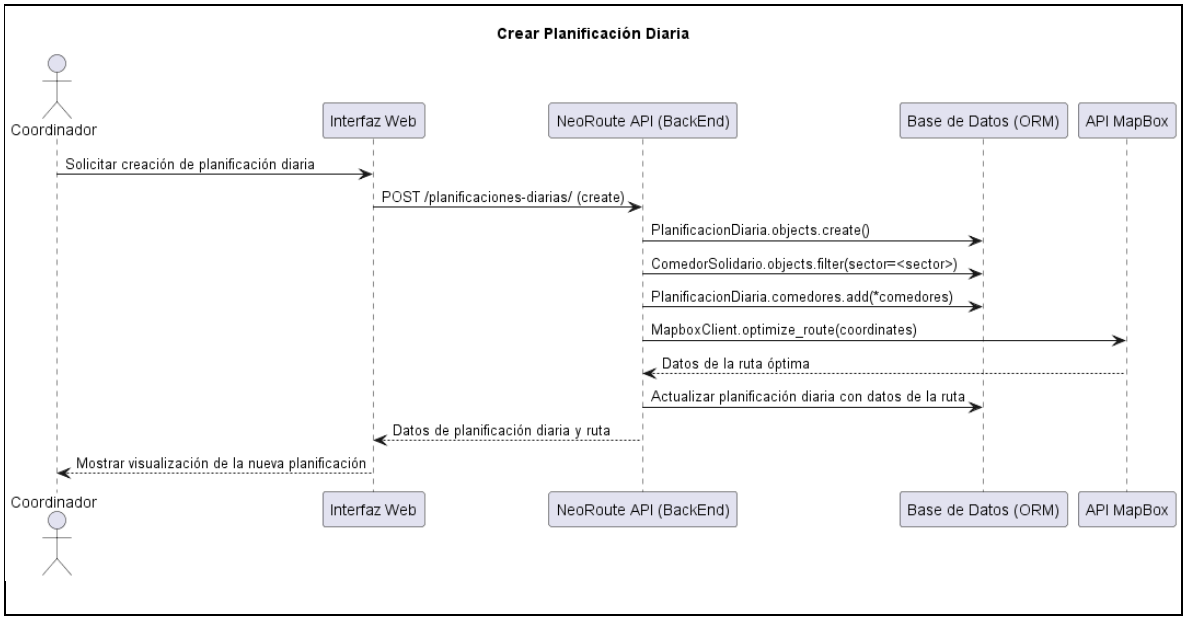


**Figura 6.** Diagrama de Secuencia: CU05 – Crear planificación diaria

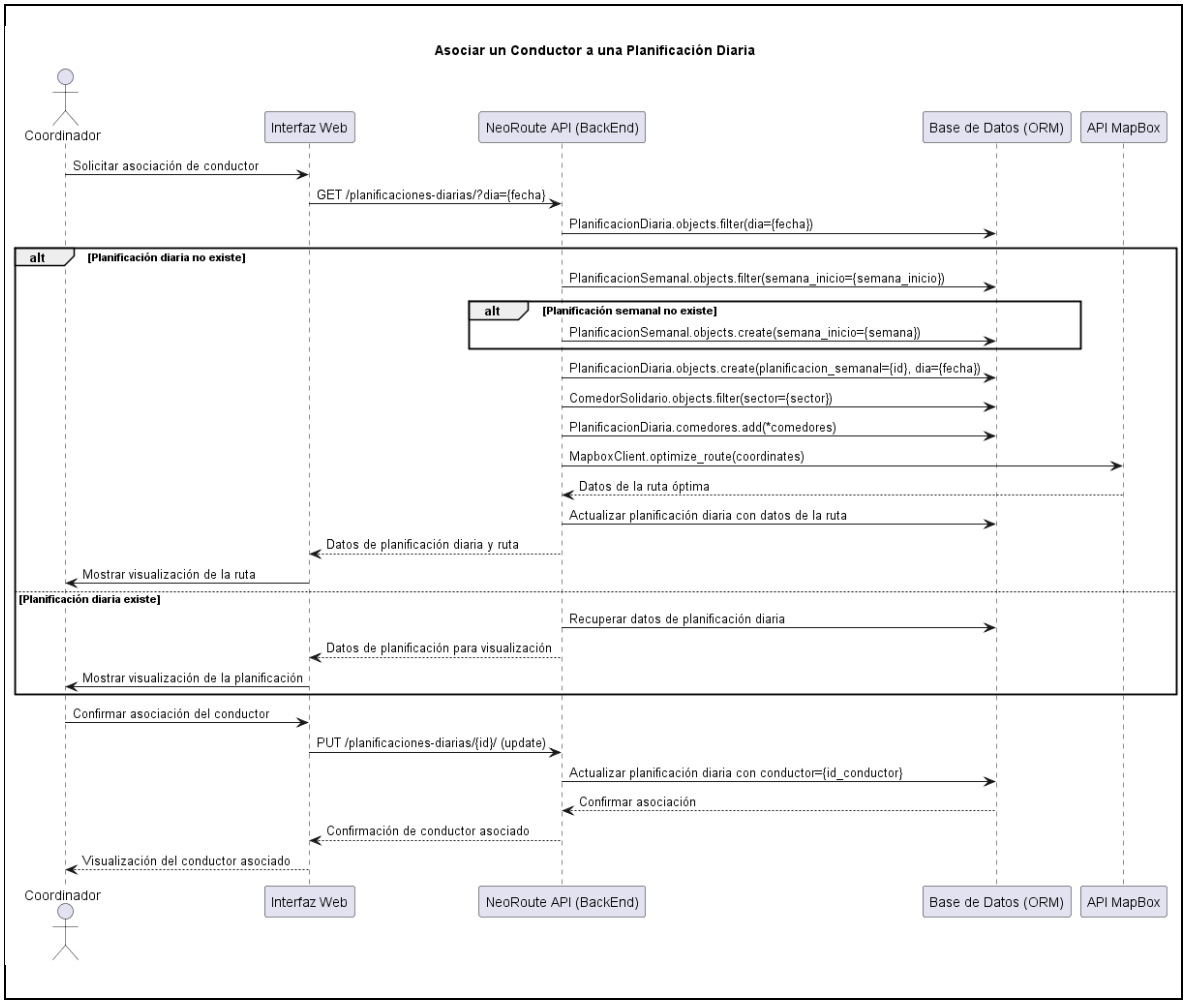


**Figura 7.** Diagrama de Secuencia: CU07 – Asociar conductor a una planificación diaria

### 3.1.3. Integración con el sistema general.

La integración con el sistema general se lleva a cabo a través de una API RESTful, que permite la comunicación fluida entre el FrontEnd (plataforma web y/o aplicación móvil) y el BackEnd La comunicación entre estos componentes se realiza mediante solicitudes HTTP y respuestas en formato JSON, que es un formato estándar para el intercambio de datos.

**Método de Integración:**

Petición HTTP (FrontEnd a BackEnd): El FrontEnd hace solicitudes HTTP a los endpoints del BackEnd utilizando el método adecuado (GET, POST, PUT, DELETE) según la operación que se desea realizar. Estas solicitudes pueden incluir datos en el cuerpo (en el caso de métodos como POST o PUT), o simplemente un parámetro de consulta en la URL (en el caso de GET).

Respuesta HTTP (BackEnd a FrontEnd): El BackEnd devuelve una respuesta en formato JSON, que incluye los datos solicitados o el resultado de la operación realizada, junto con el código de estado HTTP correspondiente para indicar el éxito o el fracaso de la operación.

A continuación, se describen algunos ejemplos de cómo se integrarán los distintos módulos del sistema utilizando endpoints típicos para la gestión de planificaciones semanales, diarias, usuarios, etc.

1. **Endpoint para crear un Rol**
   Método HTTP: POST
   URL: /api/roles/
   Descripción: Este endpoint permite a los usuarios crear nuevos roles (por ejemplo, coordinador).
   Solicitud:

```
{
    "nombre": "Coordinador"
}
```

   - Nombre: nombre del nuevo rol.

   Respuesta:
   Código de Estado: 201 Created (indica que el rol fue creado exitosamente).
   Cuerpo de la respuesta:

```
{
    "id": 1,
    "nombre": "Coordinador"
}
```

2. **Endpoint para crear un Usuario:**
   Método HTTP: POST
   URL: /api/usuarios/
   Descripción: Este endpoint permite a los usuarios crear un nuevo usuario.
   Solicitud:

```
{
    "rut": "12.345.678-9",
    "username": "Carlitos",
    "first_name": "Carlos",
    "last_name": "Donoso",
    "email": "prueba@gmail.com",
    "rol": 1
}
```

   - Rut: Identificador único Nacional.
   - Username: Nombre de usuario.
   - First_name: Nombre.
   - Last_name: Apellido.
   - Email: Correo eléctronico.
   - Rol: identificador de un rol ya creado (en este caso “coordinador”).

   Respuesta:
   Código de Estado: 201 Created
   Cuerpo de la respuesta:

```
{
    "id": 1,
    "rut": "21.289.171-1",
    "username": "Carlitos",
    "first_name": "Carlos",
    "last_name": "Donoso",
    "email": "carlitoxx102@gmail.com",
    "rol": 1
}
```

3. **Endpoint para crear un Sector:**
   Método HTTP: POST
   URL: /api/sectores/
   Descripción: Este endpoint permite a los usuarios crear un nuevo Sector.
   Solicitud:

```
{
```

```
    "nombre": "ACHUPALLAS"
}
```

- Nombre: Nombre del sector.

Respuesta:
Código de Estado: 201 Created
Cuerpo de la respuesta:

```
{
    "id": 1,
    "nombre": "ACHUPALLAS"
}
```

4. **Endpoint para crear un Encargado de un Comedor Solidario:**
   Método HTTP: POST
   URL: /api/encargados/
   Descripción: Este endpoint permite a los usuarios crear un nuevo Encargado de un Comedor Solidario.
   Solicitud:

```
{
    "nombre_completo": "Alfredo jose perez hernandez",
    "num_telefono": "+12345678910"
}
```

- Nombre_completo: Nombre completo del encargado.
- Num_telefono: Número de contacto del encargado.

Respuesta:
Código de Estado: 201 Created
Cuerpo de la respuesta:

```
{
    "id": 1,
    "nombre_completo": "Alfredo jose perez hernandez",
    "num_telefono": "+12345678910"
}
```

5. **Endpoint para crear un nuevo Comedor Solidario:**
   Método HTTP: POST
   URL: /api/comedores/
   Descripción: Este endpoint permite a los usuarios crear un nuevo Comedor Solidario.
   Solicitud:

```
{
    "nombre": "OLLITA DORADA",
    "direccion": "Luis Hurtado López 115, Viña del Mar, Valparaíso 2520000, Chile",
    "latitud": "-33.0189250000000000",
    "longitud": "-71.4996110000000000",
    "estado": "activo",
    "descripcion": null,
    "encargado": 1,
    "sector": 1
}
```

- Nombre: Nombre del Comedor Solidario.
- Dirección: Dirección física del comedor solidario.
- Latitud: Coordenada geográfica del comedor.
- Longitud: Coordenada geográfica del comedor.
- Estado: Estado del comedor, puede variar entre activo o inactivo.
- Descripción: Información adicional en caso de ser necesario (se inicializa en blanco/nulo)
- Encargado: Identificador del encargado.
- Sector: Identificador del sector.

Respuesta:
Código de Estado: 201 Created
Cuerpo de la respuesta:

```
{
```

```
    "id": 1,
    "nombre": "OLLITA DORADA",
    "direccion": "Luis Hurtado López 115, Viña del Mar, Valparaíso 2520000, Chile",
    "latitud": "-33.0189250000000000",
    "longitud": "-71.4996110000000000",
    "estado": "active",
    "descripcion": null,
    "encargado": null,
    "sector": 1
}
```


6. **Endpoint para crear una nueva Planificación semanal:**
   Método HTTP: POST
   URL: /api/ planificaciones-semanales/
   Descripción: Este endpoint permite a los usuarios crear una nueva Planificación Semanal.
   Solicitud:


```
{
    "semana_inicio": "2024-11-11"
}
```


   - Semana_inicio: Fecha del inicio de una semana en formato “YYYY-MM-DD”.

   Respuesta:
   Código de Estado: 201 Created
   Cuerpo de la respuesta:


```
{
    "id": 1,
    "semana_inicio": "2024-11-11"
}
```


7. **Endpoint para crear una nueva Planificación diaria:**
   Método HTTP: POST
   URL: /api/ planificaciones-diarias/
   Descripción: Este endpoint permite a los usuarios crear una nueva Planificación diaria.
   Solicitud:


```
{
    "dia": "2024-11-11",
    "estado_ruta": "creada",
    "id_envio": null,
    "productos": [],
    "planificacion_semanal": 1,
    "sector": 1,
    "conductor": 2,
    "comedores": [
        9,
        10
    ]
}
```


   - Dia: Fecha en la que se realizara el abastecimiento de los comedores solidarios en formato “YYYY-MM-DD”.
   - Estado_ruta: Estado de la ruta, varía entre “Creada”, “En Camino” y “Finalizada”
   - Id_envio: Identificador de un envío del inventario externo.
   - Productos: Lista de productos de un envío del inventario externo.
   - Planificacion_semanal: Identificador de la planificación semanal.
   - Sector: Identificador del sector.
   - Conductor: identificador del usuario (con rol conductor).
   - Comedores: Lista de identificadores de Comedores Solidarios.

   Respuesta:
   Código de Estado: 201 Created
   Cuerpo de la respuesta:


```
{
    "id": 19,
```

```
    "comedores_asociados": [
        {
           "id": 9,
            "nombre": "JV LOMAS CHORRILLOS",
            "direccion": "Valdivia 214, Viña del Mar, Valparaíso 2520000, Chile",
            "latitud": "-33.04374700000000000",
            "longitud": "-71.54934700000000000",
            "estado": "active",
            "descripcion": null,
            "encargado": null,
            "sector": 2
        },
        {
            "id": 10,
            "nombre": "LOMAS LAS PALMAS",
            "direccion": "Autovía Las Palmas, Viña del Mar, Valparaíso 2520000,
Chile",
            "latitud": "-33.04700416563170000",
            "longitud": "-71.52685995871047000",
            "estado": "active",
            "descripcion": null,
            "encargado": null,
            "sector": 2
        }
    ],
    "dia": "2024-11-11",
    "estado_ruta": "creada",
    "id_envio": null,
    "productos": [],
    "planificacion_semanal": 1,
    "sector": 2,
    "conductor": 3,
    "comedores": [
        9,
        10
    ]
}
```


8. **Endpoint para asociar un envío a una Planificación diaria:**
   Método HTTP: POST o PUT
   URL: /asociar-envio/
   Descripción: Este endpoint permite a los usuarios asociar un envío del inventario a una Planificación diaria.
   Solicitud:


```
{
"id_envio": "a0633cd0-70c0-4ed9-a2e8-187d94c27255",
    "productos": [
        {
            "cantidad": 10,
            "producto": "Pan de Molde",
            "urlImagen":    "https://metroio.vtexassets.com/arquivos/ids/251368-800-
auto",
            "productoId": "20bacb2e-1e63-4d14-814f-8273a58c7a6d"
        },
        {
            "cantidad": 10,
            "producto": "Fideos Corbatas Carozzi",
            "urlImagen":
"https://carozziexport.com/assets/img/products/_large/101302_CAROZZI_CORBATA_80_25X4
00_GR.jpg",
            "productoId": "e0fd1543-362b-4833-b66c-c1245fab1971"
        },
        {
            "cantidad": 10,
            "producto": "Tenedor de Mesa",
```

```
            "urlImagen": "https://encrypted-
                                tbn0.gstatic.com/images?q=tbn:ANd9GcRtYbvKbAjm3n8-
                                1l_RAcddQ17MxWm5qU1kMA&s",
            "productoId": "894c067c-44fc-4095-a68a-82caadca1fe5"
        },
        {
            "cantidad": 10,
            "producto": "Jugo de Naranja",
            "urlImagen":
"https://tost.cl/cdn/shop/files/20JUX01_1_1200x.jpg?v=1721845283",
            "productoId": "d91bd3ad-dff1-4804-b454-0b931131805b"
        }
    ],
"planificación_id": 19
}
```

- Id_envio: Identificador del envío del inventario externo.
- Productos: Lista de Productos del envío del inventario externo.
- Planificacion_id: Identificador de la planificación diaria.

Posibles Respuestas:
Código de Estado: 200 OK.
Cuerpo de la respuesta:

```
{
  "message": "Envío asociado con éxito"
}
```

```
Código de Estado: 404 Not Found.
Cuerpo de la respuesta:
{
  "error": "Planificación no encontrada"
}
```

### 3.2. Diseño de la Base de Datos.

El modelo relacional de la base de datos fue diseñado para satisfacer las necesidades específicas del sistema NeoRoute, asegurando una estructura lógica, consistente y escalable. Este modelo actúa como el núcleo del sistema, almacenando y gestionando información crítica como usuarios, comedores solidarios, planificaciones semanales y diarias, etc. Además, la estructura relacional no solo garantiza la integridad y eficiencia del sistema, sino que también facilita la integración con otros componentes, como el sistema de inventario y la API de MapBox para la generación de rutas.

#### 3.2.1. Diseño Relacional.

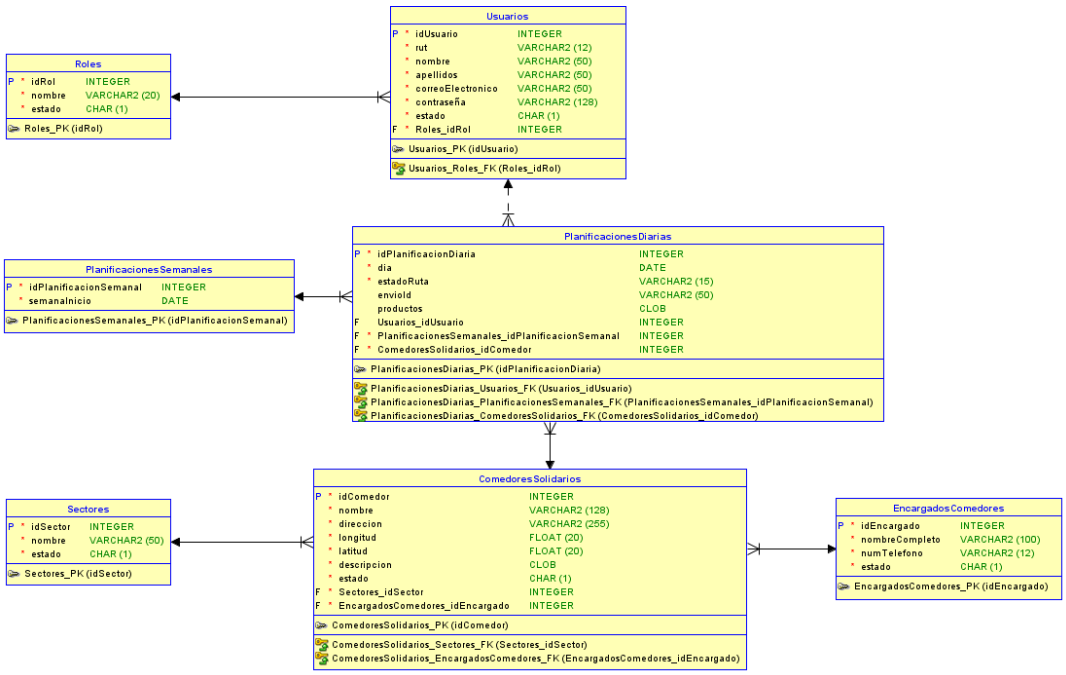


**Figura 8.** Modelo Relacional

#### 3.2.2. Diccionario de Datos.

**Tipos de Datos en PostgreSQL:**

PostgreSQL es un administrador de bases de datos con una amplia variedad de tipos de datos para definir cómo serán almacenados estos. A continuación, se muestra una lista completa de los tipos de datos que ofrece:

**Tabla 1.** Tipos de datos en PostgreSQL

| Categoría | Tipo de Dato | Descripción |
|---|---|---|
| Numéricos | SMALLINT | Entero pequeño (-32,768 a 32,767). |
| Numéricos | INTEGER (INT) | Entero estándar (-2,147,483,648 a 2,147,483,647). |
| Numéricos | BIGINT | Entero grande (-9,223,372,036,854,775,808 a 9,223,372,036,854,775,807). |

| | | |
|---|---|---|
| Numéricos | DECIMAL(p, s)<br>NUMERIC(p, s) | Número exacto con precisión y escala definidas. |
| Numéricos | REAL | Número de coma flotante de precisión simple (aproximadamente 6 dígitos decimales). |
| Numéricos | DOUBLE PRECISION | Número de coma flotante de precisión doble (aproximadamente 15 dígitos decimales). |
| Numéricos | SERIAL | Número entero autoincremental en el rango:-2,147,483,648 a 2,147,483,647. |
| Numéricos | BIGSERIAL | Número entero autoincremental en el rango: -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807 |
| Texto | CHAR(n)<br>CHARACTER(n) | Cadena de texto de longitud fija. |
| Texto | VARCHAR(n)<br>CHARACTER VARYING(n) | Cadena de texto de longitud variable con un límite máximo de n caracteres. |
| Texto | TEXT | Cadena de texto de longitud ilimitada. |
| Booleanos | BOOLEAN | Valores verdaderos (TRUE), falsos (FALSE) o nulos. |
| Fechas y Tiempos | DATE | Fecha (AAAA-MM-DD). |
| Fechas y Tiempos | TIME [WITH/WITHOUT TIME ZONE] | Hora (HH:MM:SS) con o sin huso horario. |
| Fechas y Tiempos | TIMESTAMP [WITH/WITHOUT TIME ZONE] | Marca de tiempo (fecha y hora). |
| Fechas y Tiempos | INTERVAL | Periodo de tiempo (duración). |
| Monetarios | MONEY | Representa valores monetarios con formato local. |
| Enumerados | ENUM | Tipo de dato personalizado que almacena un conjunto limitado de valores predefinidos. |
| Geográficos | POINT | Representa un punto en un plano cartesiano (x, y). |
| Geográficos | LINE | Representa una línea infinita. |
| Geográficos | LSEG | Representa un segmento de línea. |
| Geográficos | BOX | Representa un rectángulo definido por dos puntos opuestos. |
| Geográficos | PATH | Representa un conjunto de puntos conectados por segmentos. |
| Geográficos | POLYGON | Representa un polígono cerrado. |
| Geográficos | CIRCLE | Representa un círculo definido por un centro y un radio. |
| JSON y XML | JSON | Almacena datos en formato JSON. |
| JSON y XML | JSONB | Almacena datos en formato JSON en forma binaria, optimizado para consultas. |
| JSON y XML | XML | Almacena datos en formato XML. |
| Arreglos | ARRAY | Almacena matrices de cualquier tipo de dato soportado. |
| UUID | UUID | Almacena identificadores universales únicos. |
| Binarios | BYTEA | Almacena datos binarios (como archivos o imágenes). |
| Rango | INT4RANGE, INT8RANGE, NUMRANGE, TSRANGE, TSTZRANGE, DATERANGE | Representa un rango de valores (enteros, fechas, tiempos, etc.). |

| | | |
|---|---|---|
| Tipos Personalizados | Definidos por el usuario | Se pueden crear tipos de datos personalizados mediante CREATE TYPE. |

**Descripción de Tablas:**

**Tabla 2.** Diccionario de Datos: Roles

| **Nombre:** | Roles | |
|---|---|---|
| **Descripción:** | Almacena los roles que utilizarán los usuarios dentro del sistema | |
| **Clave Primaria:** | idRol | |
| **Claves Foráneas:** | | |
| **Campo** | **Tipo** | **Descripción** |
| idRol | Integer | Número único que individualiza a cada rol. |
| nombre | Varchar(20) | Texto alfanumérico en formato libre. Valor no nulo ni vacío. |
| estado | Char(01) | Estado del registro de Rol: “A”: Activo, “I”: Inactivo |

**Tabla 3.** Diccionario de Datos: Usuarios

| **Nombre:** | Usuarios | |
|---|---|---|
| **Descripción:** | Almacena los datos de los usuarios del sistema | |
| **Clave Primaria:** | idUsuario | |
| **Claves Foráneas:** | idRol → Roles.idRol | |
| **Campo** | **Tipo** | **Descripción** |
| idUsuario | Integer | Número único que individualiza a cada usuario. |
| rut | Varchar(12) | Identificador único chileno. Texto alfanumérico en formato libre. Valor no nulo ni vacío. |
| nombre | Varchar(50) | Nombre real del usuario. Texto alfanumérico en formato libre. Valor no nulo ni vacío. |
| apellidos | Varchar(50) | Apellidos reales del usuario. Texto alfanumérico en formato libre. Valor no nulo ni vacío. |
| correoElectronico | Varchar(50) | Correo electrónico del usuario. Texto alfanumérico en formato libre. Valor no nulo ni vacío. |
| contraseña | Varchar(128) | Contraseña establecida por el usuario. Texto alfanumérico en formato libre. Valor no nulo ni vacío. |
| idRol | Integer | Código identificador del rol del usuario. |
| estado | Char(01) | Estado del registro de Usuario: “A”: Activo, “I”: Inactivo |

**Tabla 4.** Diccionario de Datos: PlanificacionesSemanales

| **Nombre:** | PlanificacionesSemanales | |
|---|---|---|
| **Descripción:** | Almacena la fecha de inicio de cada planificación semanal. | |
| **Clave Primaria:** | idPlanificacionSemanal | |
| **Claves Foráneas:** | | |
| **Campo** | **Tipo** | **Descripción** |
| idPlanificacionSemanal | Integer | Número único que individualiza a cada planificación semanal. |
| semanaInicio | Date | Fecha del inicio de la semana. Valor no nulo ni vacío. |

**Tabla 5**. Diccionario de Datos: Sectores

| **Nombre:** | Sectores | |
|---|---|---|
| **Descripción:** | Almacena información de los sectores para agrupar los comedores solidarios | |
| **Clave Primaria:** | idSector | |
| **Claves Foráneas:** | | |
| **Campo** | **Tipo** | **Descripción** |
| idSector | Integer | Número único que individualiza a cada sector. |
| nombre | Varchar(50) | Nombre del sector. Texto alfanumérico en formato libre. Valor no nulo ni vacío. |
| estado | Char(01) | Estado del registro de Sector: “A”: Activo, “I”: Inactivo |

**Tabla 6.** Diccionario de Datos: EncargadosComedores

| **Nombre:** | EncargadosComedores | |
|---|---|---|
| **Descripción:** | Almacena información relevante de los encargados de cada comedor solidario. | |
| **Clave Primaria:** | idEncargado | |
| **Claves Foráneas:** | | |
| **Campo** | **Tipo** | **Descripción** |
| idEncargado | Integer | Número único que individualiza a cada encargado. |
| nombreCompleto | Varchar(100) | Nombre Completo del encargado. Texto alfanumérico en formato libre. Valor no nulo ni vacío. |
| numTelefono | Varchar(12) | Número telefónico de contacto. Texto alfanumérico en formato libre. Valor no nulo ni vacío. |
| estado | Char(01) | Estado del registro del Encargado: “A”: Activo, “I”: Inactivo |

**Tabla 7.** Diccionario de Datos: ComedoresSolidarios

| **Nombre:** | ComedoresSolidarios | |
|---|---|---|
| **Descripción:** | Almacena información relevante de los comedores solidarios. | |
| **Clave Primaria:** | idComedor | |
| **Claves Foráneas:** | idEncargado → EncargadosComedores.idEncargado<br>sectoresIdSector → Sectores.idSector | |
| **Campo** | **Tipo** | **Descripción** |
| idComedor | Integer | Número único que individualiza a cada comedor solidario. |
| nombre | Varchar(128) | Nombre del comedor solidario. Texto alfanumérico en formato libre. Valor no nulo ni vacío. |
| dirección | Varchar(255) | Dirección física del comedor solidario. Texto alfanumérico en formato libre. Valor no nulo ni vacío. |
| longitud | Float(20) | Dirección geográfica del comedor solidario. |
| latitud | Float(20) | Dirección geográfica del comedor solidario. |
| descripción | Text | Datos extras de cada comedor solidario. Puede estar vacío o nulo. |
| idEncargado | Integer | Código identificador del encargado. |
| idSector | Integer | Código identificador del sector. |
| estado | Varchar(01) | Estado del comedor solidario, varía entre “Activo” e “Inactivo”. |

**Tabla 8.** Diccionario de Datos: PlanificacionesDiarias

| **Nombre:** | PlanificacionesDiarias | |
|---|---|---|
| **Descripción:** | Almacena toda la información respecto a las planificaciones diarias. | |
| **Clave Primaria:** | idPlanificacionDiaria | |
| **Claves Foráneas:** | idPlanificacionSemanal → PlanificacionesSemanales.idPlanificacionSemanal<br>idUsuario → Usuarios.idUsuario<br>idComedor → ComedoresSolidarios.idComedor | |
| **Campo** | **Tipo** | **Descripción** |
| idPlanificacionDiaria | Integer | Número único que individualiza cada planificación diaria. |
| dia | Date | Fecha en la que se abastecerán comedores solidarios. |
| estadoRuta | Varchar(15) | Estado de la ruta. Varía entre “Pendiente”, “En reparto” y “Completada”. |
| envioId | Varchar(50) | Identificador único del envío obtenido del sistema de inventario asociado a una planificación diaria. Puede ser nulo o vacío. |
| productos | JSON | Listado de productos a repartir entre los comedores obtenidos del envío. |
| idPlanificacionSemanal | Integer | Código identificador de la planificación semanal a la cual corresponde. |
| idUsuario | Integer | Código identificador del conductor asociado. |
| idComedor | Integer | Código identificador de los comedores dentro de una planificación diaria (ruta). |

## 3.3. Implementación

### 3.3.1. Detalles de la codificación y desarrollo.

De acuerdo con lo mencionado anteriormente, el desarrollo del proyecto está dividido en tres sprints, donde cada uno cuenta con sus historias de usuario y tareas a realizar, tal como se observa a continuación:

**Sprint 1: Creación de la base de datos, modelos y serializadores.**

Historias de usuario y tareas:

1. Como coordinador, quiero poder almacenar, editar y eliminar los datos necesarios para el funcionamiento del sistema (usuarios, comedores solidarios, planificaciones semanales, planificaciones diarias, etc.), para que la información esté estructurada y disponible para la gestión.
    a. Diseñar la base de datos con tablas que incluyan las entidades claves.
    b. Configurar las relaciones necesarias entre las tablas.
    c. Implementar modelos en Django Rest para reflejar las entidades principales y sus relaciones.
    d. Crear serializadores para exponer los datos a través de la API REST.
    e. Implementar los primeros endpoints del sistema.

**Sprint 2: Autenticación, lógica de negocio y comunicación web-móvil.**

Historias de usuarios y tareas:

1. Como usuario (coordinador y conductor) quiero poder acceder al sistema de forma segura, para utilizar las funciones según mis permisos.
    a. Implementar un sistema de autenticación basado en JWT.
    b. Configurar roles y permisos para diferentes tipos de usuarios (coordinadores y conductores)

2. Como coordinador, quiero que, al momento de generar las planificaciones diarias, se previsualicen las rutas óptimas para el abastecimiento de los comedores y poder asignarles un conductor.
    a. Integrar MapBox API para generar rutas optimas.
    b. Establecer la lógica para asignar rutas a conductores según la planificación diaria.

3. Como conductor, quiero recibir en tiempo real la ruta asignada.
    a. Configurar la API para visualizar las planificaciones diarias asociadas al conductor.
    b. Implementación de Websockets para la comunicación en tiempo real.

**Sprint 3: Optimización, pruebas e integración con inventario.**

Historias de usuarios y tareas:

1. Como coordinador, quiero asegurarme de que el sistema funcione correctamente y sin interrupciones.
    a. Implementar pruebas unitarias para los componentes individuales.
    b. Realizar pruebas de integración para la API.

2. Como coordinador, quiero sincronizar los productos de un envío generado por el sistema de inventario a una planificación diaria.
    a. Desarrollar la integración con el sistema de inventario externo.
    b. Asegurar que las planificaciones diarias almacenen los datos relevantes del envío (id y productos).

A continuación, en la **Figura 9**, se detalla de manera más visual la programación de los sprints.

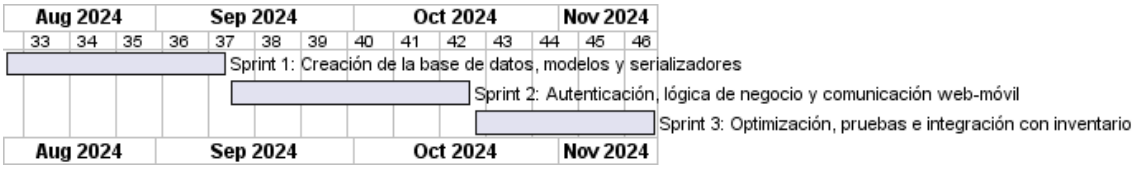


**Figura 9.** Carta Gantt del desarrollo del proyecto

Por otro lado, para gestionar el desarrollo del proyecto se utilizó GIT junto con un repositorio Github del proyecto NeoRoute general, es decir, existían ramas específicas para la aplicación móvil, la web y el BackEnd, por lo que no se crearon ramas específicas para dividir los sprints o funcionalidades, pero se realizaron commits frecuentes y descriptivos que reflejaban los avances de cada sprint en la rama del BackEnd.

### 3.3.2. Uso de buenas prácticas y patrones de diseño.

Los patrones y buenas prácticas seleccionados fueron elegidos debido a sus beneficios específicos y a cómo abordan las necesidades del sistema en desarrollo. A continuación, se explican las razones detrás de cada elección:

- **Patrón observer**: Debido a que establece una relación de suscriptor-publicador en la que el sujeto notifica automáticamente a sus observadores (suscriptores) sobre eventos o cambios de estado, en este caso, es esencial enviar la notificación al momento de asignar una ruta a un conductor, de manera que, al momento de realizar la asociación, el conductor obtenga en tiempo real la ruta.

- **Patrón Template Method**: Este patrón define el esqueleto de un algoritmo en un método base, permitiendo que las subclases sobrescriban partes específicas del proceso sin alterar su estructura general, permitiendo así que, al momento de asociar un conductor, lo que en otras palabras es que, al actualizar una planificación diaria, se pueda implementar la lógica de notificar al conductor mediante un websocket al momento de ser asociado.

Por otro lado, actualmente las buenas prácticas que se están aplicando son:

- **Commits frecuentes y descriptivos**: Cada cambio en el código se "documentó" con mensajes de commit claros y relacionados con tareas específicas, lo que facilita el rastreo de cambios y depuración de errores.
- **Entornos separados**: Se configuraron entornos específicos para desarrollo, pruebas y producción, asegurando que los cambios no afecten al sistema en producción de forma imprevista.
- **Uso de variables de entorno**: Para el entorno de producción, se utilizan variables de entorno para gestionar informaciones sensibles de configuración (tokens, claves de API, conexión a la base de datos, etc.)

- **Desacoplamiento de componentes**: La lógica de negocio, la gestión de datos y la interfaz de usuario se separaron claramente, siguiendo el principio MVT (Model-View-Template) específico de Django Rest Framework.

### 3.4. Pruebas y Validación

**Estrategias de testing aplicadas a los componentes.**

Para verificar que el BackEnd cumple con los requerimientos planteados, se realizaron las siguientes pruebas:

- **Pruebas de integración**: Para garantizar la correcta comunicación entre los diferentes componentes del sistema (BackEnd, plataforma web y aplicación móvil), se realizaron pruebas de integración de manera manual utilizando herramientas como Postman y Wscat:

    - Solicitudes HTTP: se utilizó Postman para probar los distintos endpoints del BackEnd. Esto permitió verificar que las solicitudes y respuestas HTTP funcionarán correctamente.

    - WebSockets: Para validar la comunicación en tiempo real, se utilizó Wscat simulando la asignación de rutas, asegurando que el mensaje se envíe y se recepcione de manera correcta.

  Una vez validadas estás funcionalidades en un entorno de pruebas, se procedió a integrar el sistema con la plataforma web y la aplicación móvil, verificando manualmente el correcto funcionamiento de las solicitudes HTTP y WebSockets, garantizando una experiencia funcional para los usuarios.

- **Pruebas de Usuario**: Se coordinaron una serie de reuniones con los usuarios finales para validar el funcionamiento del sistema en un entorno más cercano a lo real. Durante dichas reuniones, los usuarios utilizaron tanto la plataforma web como la aplicación móvil para realizar las tareas habituales, como la planificación de las rutas, la asignación de conductores y el seguimiento de una ruta. Los comentarios obtenidos de los usuarios luego de realizar las pruebas permitieron ajustar detalles en la interfaz y lógica del sistema, asegurando que sea fácil de usar para ellos y cumpliera con las necesidades del proyecto.

### 3.5. Integración con Otros Componentes

El sistema NeoRoute interactúa con los siguientes componentes externos:

1. **Sistema de inventario**: Para la integración del sistema de inventario en nuestro sistema, desde la web se debe enviar un POST a la URL: http://34.176.26.41/api/envios, donde en el body se debe enviar una fecha en formato “YYYY-MM-DD” para recibir los envíos disponibles con esa fecha.

2. **Sistema de Mapas y Rutas (MapBox API):** Como se mencionó anteriormente, MapBox API se utilizará principalmente para generar rutas óptimas, por lo que la integración se realiza mediante peticiones a la API, donde desde el BackEnd se envían las coordenadas de los puntos de entrega correspondientes a una planificación diaria al endpoint de MapBox para recibir la ruta optimizada.

A continuación, en la **Figura 10**, se muestra de manera más sencilla cómo interactúa el sistema NeoRoute con los distintos componentes externos.

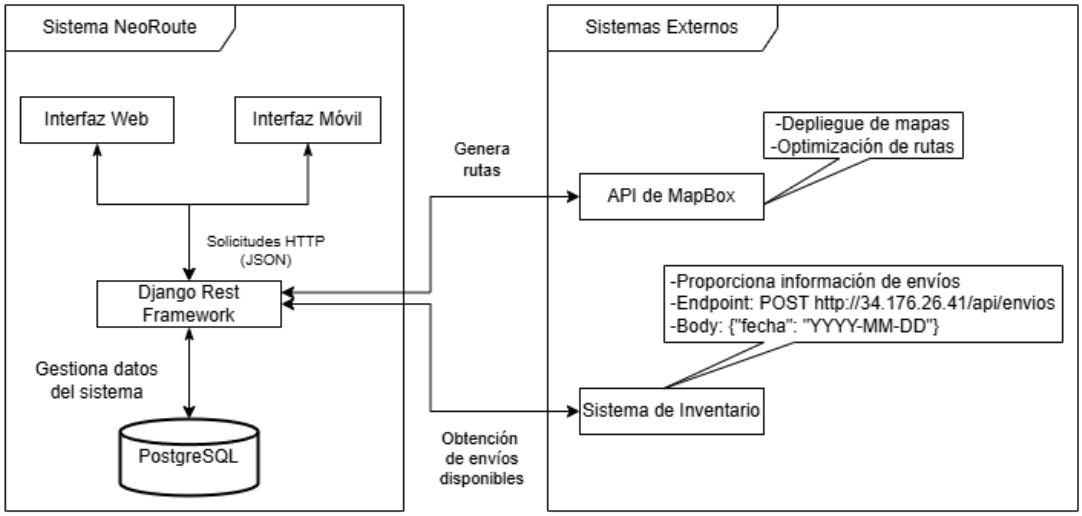


**Figura 10.** Arquitectura de NeoRoute con terceros

## 4. Conclusiones

El desarrollo del sistema BackEnd para la optimización de rutas y la gestión de entregas en los comedores solidarios de Viña del Mar representó un desafío significativo, pero también una gran oportunidad para aplicar conocimientos técnicos y resolver problemas reales con impacto social. Este proyecto no solo logró mejorar la distribución equitativa de recursos y reducir los tiempos de entrega, sino que también demostró el papel crucial de la informática en la mejora de procesos logísticos en situaciones de emergencia.

Entre las lecciones aprendidas destacan los desafíos enfrentados durante la integración de tecnologías clave, como MapBox API y PostgreSQL con PostGIS, para manejar datos geoespaciales y optimizar rutas en tiempo real. Por ejemplo, uno de los problemas fue el manejo eficiente de grandes volúmenes de datos relacionados con múltiples puntos de entrega, lo que requirió la refactorización de la lógica de negocio y la optimización de consultas a la base de datos. Estas experiencias fortalecieron habilidades como la resolución de problemas complejos, la planificación iterativa mediante sprints y el uso de patrones de diseño como Observer y Template Method, esenciales para garantizar la escalabilidad y modularidad del sistema.

Los logros más importantes incluyen la implementación exitosa de un sistema que mejora la logística de abastecimiento, asegurando un impacto positivo en los beneficiarios de los comedores solidarios y reduciendo costos operativos para la municipalidad. Además, la plataforma está preparada para futuras extensiones, como el seguimiento en tiempo real de vehículos y la incorporación de análisis predictivos basados en machine learning, lo que permitiría prever la demanda de recursos y planificar entregas más eficientes.

A futuro, se recomienda explorar técnicas avanzadas de análisis de datos históricos para predecir patrones de consumo y optimizar la planificación semanal de entregas. Asimismo, podría implementarse un sistema de notificaciones automatizadas para mantener informados a los usuarios y mejorar la comunicación en tiempo real. Estas mejoras continuarían fortaleciendo el impacto del proyecto, tanto en términos sociales como en su aporte a la disciplina de la ingeniería informática.

## 5. Agradecimientos.

Agradezco profundamente al profesor Oscar Carrasco por su guía y apoyo constante en cada etapa de este trabajo. Asimismo, expreso mi gratitud a mis compañeros y equipo del proyecto NeoRoute, cuya cooperación y compromiso hicieron posible el llevar a cabo el desarrollo de este nuestro software. Finalmente, un especial reconocimiento a mi familia por su paciencia y aliento durante todo el proceso.

## 6. Referencias

[1] J. S. Ken Schwaber, «La Guía Scrum,» Scrum, Noviembre 2020. [En línea]. Available: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Spanish-European.pdf. [Último acceso: Abril 2024].

[2] A. Raeburn, «Asana, Inc,» 13 Febrero 2024. [En línea]. Available: https://asana.com/es/resources/extreme-programming-xp. [Último acceso: Abril 2024].

[3] F. Machuca, «crehana,» 19 Mayo 2022. [En línea]. Available: https://www.crehana.com/blog/transformacion-digital/que-es-el-backend-y-como-usarlo/. [Último acceso: Abril 2024].

[4] A. W. S. (AWS), «Amazon Web Services,» [En línea]. Available: https://aws.amazon.com/es/what-is/restful-api/. [Último acceso: Mayo 2024].

[5] C. Frisoli, «HubSpot,» 10 Noviembre 2023. [En línea]. Available: https://blog.hubspot.es/website/framework-desarrollo-web. [Último acceso: Mayo 2024].

[6] p. E. R. Moraguez, «lovtechnology,» [En línea]. Available: https://lovtechnology.com/que-es-postgresql-como-funciona-y-para-que-sirve/. [Último acceso: Mayo 2024].

[7] K. Ostrowska, «learnSQL,» 4 Enero 2023. [En línea]. Available: https://learnsql.es/blog/las-bases-de-datos-mas-populares-en-2023/. [Último acceso: Mayo 2024].