Especificación de requisitos de software

Para

Logistics company

Versión 1.0 aprobado

Preparado por Amaury Enrique Bula Salsas

Dilson Rivera Gutierrez

Nilson Rivera Gutierrez

Universidad Tecnológica de Bolivar

23/04/25

Tabla de contenido

Historial de revisiones

Nombre	Fecha	Motivo de los cambios	Versión
MVP	17/02/25	Creación del documento que contiene el MVP	0.1
Sprint #1	16/02/2025	Diseño de diagrama estructural, diagrama de contexto, levantamiento de requerimientos, Gestión de Proyecto	0.1.1
Sprint #2	29/23/25	Diseño estructural, Diseño de Mockup UI, Gestión de Repositorio, Creación estructura de proyecto en GitHub,	2

		Definición de restricciones del Proyecto	
Sprint #3	23/04/25	Creación de fastAPIs (Shipping Order, routing service, tracking service y driver service), integración de estas mismas entre si.	3

Descripción general

Las empresas de logística manejan múltiples procesos simultáneamente, como la gestión de pedidos, el rastreo de envíos, la planificación de rutas y la administración de flotas. Sin embargo, muchas de estas empresas han desarrollado o adquirido sistemas independientes a lo largo del tiempo, lo que ha generado silos de datos y dificultades en la optimización operativa.

Este proyecto busca desarrollar un sistema de gestión logística basado en microservicios que permita integrar todas las áreas operativas a través de una arquitectura moderna, facilitando el acceso unificado a los datos y optimizando la toma de decisiones.

1. Problema a resolver

La empresa enfrenta varios desafíos en su operación logística debido a la falta de integración entre sus sistemas:

- **Datos aislados**: La información sobre pedidos, rutas y flotas está dispersa en distintos sistemas, dificultando el acceso y la toma de decisiones en tiempo real.
- **Falta de visibilidad operativa**: No hay conexión entre la planificación de rutas y la disponibilidad de vehículos o cargas en los almacenes.
- **Procesos manuales ineficientes**: La programación de entregas y la asignación de flotas se realizan sin una sincronización óptima, aumentando costos y tiempos de entrega.
- **Falta de un almacenamiento centralizado**: No se cuenta con un sistema que unifique la información y permita realizar análisis para mejorar la eficiencia operativa.

2. Características del sistema

El Sistema de Gestión Logística con Microservicios se diseñará con un enfoque modular, escalable, seguro y eficiente, asegurando una integración fluida entre los distintos procesos logísticos, optimizando la gestión de pedidos, el seguimiento de entregas y la planificación de rutas.

Este sistema permitirá que la empresa reduzca costos, mejore la visibilidad operativa y aumente la eficiencia en la entrega de pedidos al proporcionar una arquitectura flexible basada en microservicios, con APIs bien definidas y un almacenamiento centralizado de datos para la toma de decisiones basada en análisis operativos.

2.1.1 Arquitectura Basada en Microservicios

- **Desacoplamiento de servicios**: Cada funcionalidad clave (gestión de pedidos, seguimiento, enrutamiento, etc.) estará separada en diferentes microservicios.
- **Escalabilidad**: Se podrán añadir nuevas funcionalidades sin afectar el resto del sistema.
- **Resiliencia**: Un fallo en un servicio no afectará el funcionamiento global.

2.1.2 Plataforma para Clientes

- **Seguimiento de envíos en tiempo real**: Se mostrará el estado y ubicación de los pedidos.

2.1.3 Gestión de Pedidos y Logística

- **Microservicio de seguimiento de pedidos**: Permite rastrear la ubicación y estado de cada pedido en tiempo real.
- **Microservicio de enrutamiento**: Calcula las rutas óptimas para las entregas.
- **Microservicio de programación de entregas**: Coordina horarios según la disponibilidad de flotas y almacenes.
  Microservicio de gestión de flotas: Administra la disponibilidad de vehículos y conductores.

2.1.4 Integración de Datos y APIs

- **API Unificada**: Todos los microservicios estarán conectados mediante APIs bien definidas.
- **Pasarela de seguridad para APIs**: Controla el acceso mediante autenticación y autorización.
- **Almacén de datos en la nube**: Consolida la información para análisis y reportes.

2.1.5 Seguridad y Control de Accesos.

- **Monitoreo de actividad**: Se registrarán eventos para detectar posibles incidentes.

2.1.6 Análisis y Optimización

- **Dashboard de reportes**: Permite visualizar métricas clave como tiempos de entrega y eficiencia de rutas.

2.1.7 Requisitos funcionales del sistema

RF1: El sistema debe permitir el desarrollo de **microservicios** para:

- Seguimiento de pedidos (*tracking*).
- Programación de entregas.
- Enrutamiento.
- Gestión de flotas.

RF2: Rastreo de Envíos

- Funcionalidad: Proporcionar actualizaciones en tiempo real sobre la ubicación y el estado de los envíos.
- Detalles: Integración con sistemas de GPS o tecnologías de seguimiento para proporcionar información precisa a los clientes y la empresa.

RF3: Análisis Operativo

- Funcionalidad: Generar informes y análisis sobre el rendimiento de la flota, eficiencia de las rutas y tiempos de entrega.
- Detalles: Utilizar datos históricos y en tiempo real para identificar áreas de mejora y tomar decisiones informadas.

RF4: Integración de Sistemas

- Funcionalidad: Integrar con sistemas existentes de gestión de inventario, ERP y otros sistemas operativos.
- Detalles: Asegurar que los datos fluyan sin problemas entre diferentes sistemas para una visión unificada.

2.2 Requisitos No funcionales

RNF1: La arquitectura debe evolucionar gradualmente hacia un sistema desacoplado basado en microservicios.

RNF2: Las APIs deben estar bien definidas y proporcionar acceso unificado a los datos de logística.

RNF3: El sistema debe permitir la coordinación entre diferentes capacidades del negocio, eliminando la fragmentación de datos.

RNF4: La integración entre los sistemas debe permitir una mejora en la toma de decisiones basada en datos.

RNF5: Se debe garantizar un acceso seguro a los datos a través de una pasarela.

3. MVP Propuesto

3.1 Microservicio de Seguimiento de Pedidos

- Permite a los clientes rastrear sus pedidos en tiempo real.
- Relacionado con: **RF1, RF2**

3.2 Microservicio de Enrutamiento

- Calcula rutas de entrega basadas en datos estáticos.
- Relacionado con: **RF1, RF2**

3.3 Microservicio de Programación de Entregas

- Gestiona horarios de entrega según la disponibilidad de la flota.
- Relacionado con: **RF1, RF2**

3.4 API Unificada de Datos Logísticos

- Proporciona acceso a los datos clave de entrega y activos mediante APIs bien definidas.
- Relacionado con: **RF2**

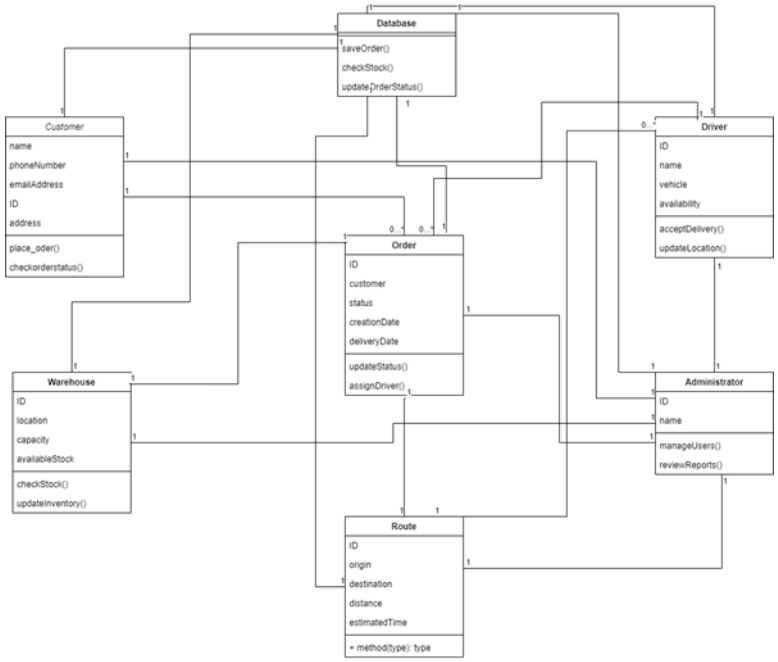
3.5 Pasarela Segura para APIs

- Controla el acceso a los servicios mediante autenticación y autorización.
- Relacionado con: **RF3, RNF5**

3.6 Almacén de Datos en la Nube

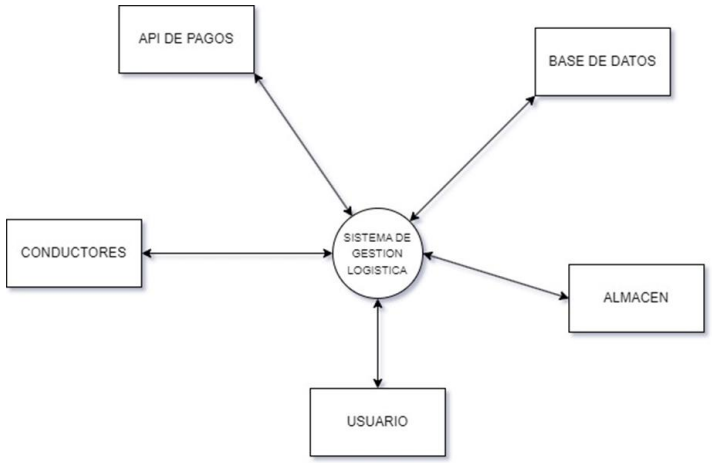
- Consolida los datos de pedidos, flotas y rutas para análisis.
- Relacionado con: **RF4, RNF3**

4. Sprint #1



4.1 Diagrama Estructural

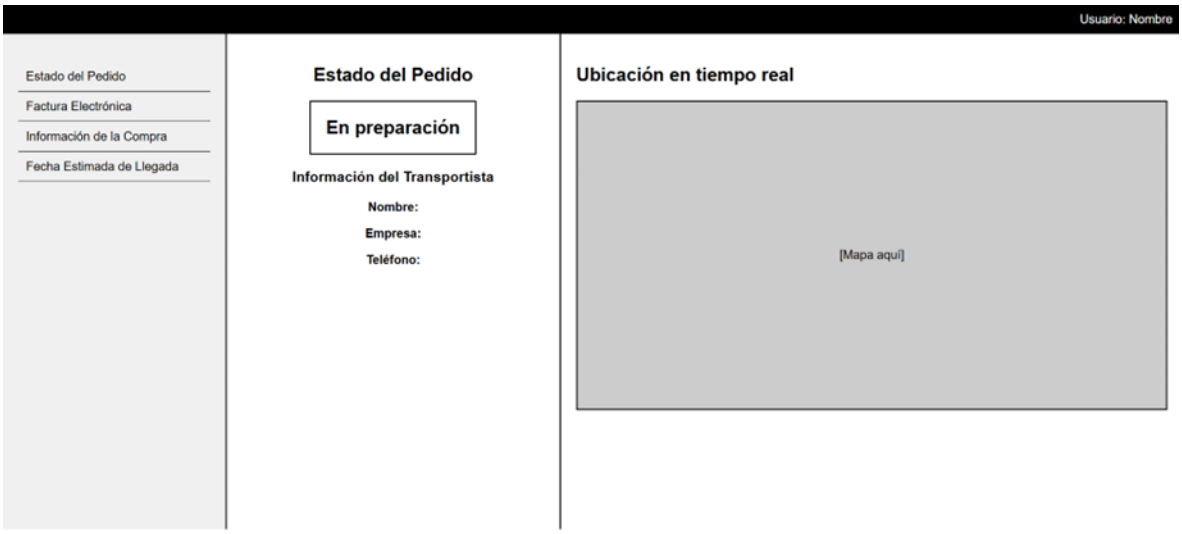
4.2 Diagrama de contexto



5. Sprint #2

5.1 Diseño estructural

5.2 Mockup UI



6. Sprint #3

El objetivo de este sprint es el desarrollo e integración de los microservicios que conformaran la aplicación de logística.

6.1 Microservicios desarrollados

6.1.1 Microservicio de Gestión de Pedidos (shipping_order_service)

6.1.1.1 Objetivo

Crear pedidos de envío, generar códigos de rastreo únicos y enviar eventos de seguimiento al microservicio de tracking.

6.1.1.2 Endpoints principales:

- POST /orders: Crea un nuevo pedido y notifica al tracking.
- GET /orders: Retorna todos los pedidos existentes.

6.1.1.3 Flujo:

1. Recibe datos del pedido.

2. Genera order_id y tracking_code.
3. Guarda el pedido en un archivo JSON.
4. Envía un evento “created” al servicio de tracking.

6.2 Microservicio de Gestión de Conductores (driver_service)

6.2.1 Responsabilidad:

Registrar y listar conductores disponibles.

6.2.2 Endpoints principales:

- POST /drivers: Registra un nuevo conductor.
- GET /drivers: Lista todos los conductores registrados.

6.3 Microservicio de Rutas (routing_service)

6.3.1 Responsabilidad:

Crear y gestionar rutas de entrega.

6.3.2 Endpoints principales:

- POST /routes: Crea una ruta de entrega.
- GET /routes: Lista todas las rutas.
- DELETE /routes/{route_id}: Elimina una ruta específica.

6.4 Microservicio de Seguimiento (tracking_service)

6.4.1 Responsabilidad:

Registrar y mostrar el historial de eventos de seguimiento de pedidos.

6.4.2 Endpoints principales:

- POST /tracking/track: Registra un evento de seguimiento.
- GET /tracking/track/{tracking_code}: Devuelve el historial de eventos de un pedido.

6.5 Integración entre Microservicios

Se implementó una comunicación asincrónica HTTP para conectar shipping_order_service con tracking_service. Al crear un nuevo pedido:

- Se realiza una petición POST al endpoint /tracking/track desde shipping_order_service.
- Se documentó el cliente tracking_service_client.py para esta tarea.

4.3 Repositorio de Datos

Todos los microservicios almacenan su información en archivos JSON dentro del módulo repository. Esto facilita el acceso a los datos y la persistencia local:

- Pedidos: shipping_order_service/app/repository/orders.json
- Conductores: driver_service/app/repository/drivers.json
- Rutas: routing_service/app/repository/routes.json
- Eventos de seguimiento: tracking_service/app/repository/tracking.json

6.6 Pruebas Unitarias e Integración

Durante esta etapa, se implementaron pruebas automatizadas para asegurar el correcto funcionamiento de los microservicios desarrollados. Estas pruebas incluyen:

Pruebas unitarias: Validan de forma aislada la lógica de negocio en funciones y servicios individuales. Por ejemplo:

6.5.1 Validación de datos recibidos en los DTOs.

Procesamiento de lógica en funciones del servicio (como la creación de pedidos o asignación de rutas).

Pruebas de integración: Simulan flujos completos entre microservicios y evalúan la comunicación HTTP entre ellos. Algunas coberturas incluyen:

Creación de un pedido y notificación al microservicio de tracking.

Creación y asignación de rutas a conductores.

Actualización del estado del pedido a través de eventos de seguimiento.

Las pruebas se ejecutaron usando pytest junto con pytest-asyncio para funciones asíncronas, y herramientas como httpx para pruebas HTTP. Se logró una cobertura significativa del código, reflejada mediante reportes coverage.xml y su análisis con SonarCloud.