🧩 Defensa del Proyecto – Microservicio ms-solicitudes

1. Introducción

El presente trabajo corresponde al microservicio **ms-solicitudes**, desarrollado como parte del **Trabajo Práctico Integrador** de la materia *Backend de Aplicaciones*.

Dentro del ecosistema del sistema, el microservicio *ms-solicitudes* cumple un rol central: **gestionar las solicitudes de traslado de contenedores**, interactuando con otros servicios del sistema.
El diseño busca garantizar **modularidad, escalabilidad, seguridad y facilidad de mantenimiento**, alineándose con la arquitectura moderna de aplicaciones backend.

---

2. Objetivo funcional

El microservicio **ms-solicitudes** se encarga de administrar el ciclo de vida de las solicitudes de transporte. Cada solicitud contiene información del cliente, la ruta a seguir, los tramos intermedios, el contenedor asignado, la tarifa y el camión involucrado. Entre sus principales funciones se encuentran:

- Registrar nuevas solicitudes.
- Consultar solicitudes existentes por número de seguimiento, cliente o estado.
- Gestionar rutas y tramos asociados a cada solicitud.
- Coordinar datos con otros microservicios para obtener información externa (cliente, camión, tarifa).
- Persistir toda la información en una base de datos relacional PostgreSQL.

De esta forma, *ms-solicitudes* representa el núcleo operativo del flujo logístico dentro del sistema global.

---

3. Arquitectura y organización interna

El microservicio está desarrollado en **Java 17** utilizando **Spring Boot 3.2**, y sigue una **arquitectura multicapa** que promueve la separación de responsabilidades y el bajo acoplamiento entre componentes:

Capa	Responsabilidad	Clases principales
Controller	Exposición de endpoints REST y comunicación con el frontend o API Gateway	`SolicitudController`, `RutaController`, `ContenedorController`, `TramoController`
Service / ServiceImpl	Contiene la lógica de negocio y orquestación de procesos	`SolicitudServiceImpl`, `RutaServiceImpl`, `TramoServiceImpl`, etc.
Repository	Gestión de persistencia mediante JPA	`SolicitudRepository`, `RutaRepository`, `ContenedorRepository`
Modelo (Entidades y DTOs)	Representación de los datos y su transporte entre capas	`Solicitud`, `Ruta`, `Tramo`, `Contenedor`, `TipoEstado`, `DTOs`
Configuración	Ajustes técnicos y de infraestructura	`FeignClientConfig`, `OpenApiConfig`, `SecurityConfig`
Integración	Comunicación entre microservicios	`UsuariosClient`, `TarifasClient`, `CamionesClient`

La base de datos se inicializa automáticamente con `data.sql`, lo cual facilita la ejecución y pruebas en entornos de desarrollo o contenedores Docker.

---

4. Comunicación entre microservicios

La integración con otros componentes del sistema se realiza mediante **Spring Cloud OpenFeign**, que permite invocar endpoints HTTP de forma declarativa sin necesidad de escribir código de cliente manual.

Por ejemplo, el microservicio se comunica con `ms-usuarios`, `ms-tarifas` y `ms-camiones` para obtener información externa relacionada a una solicitud.
Esto implementa el principio de **bajo acoplamiento e independencia funcional** característico de las arquitecturas basadas en microservicios.

Los endpoints externos se configuran dinámicamente a través de los archivos `application.properties` y `application-docker.properties`, permitiendo cambiar fácilmente entre entornos locales y contenerizados.

---

5. Persistencia y modelo de datos

El acceso a datos se implementa mediante **Spring Data JPA**, utilizando **PostgreSQL** como motor de base de datos.
Cada entidad está mapeada a una tabla del esquema relacional, y las relaciones entre ellas reflejan el dominio logístico del sistema:

- **Solicitud:** entidad principal que representa la orden de transporte.
  Contiene referencias a cliente, camión, tarifa, ruta y contenedor.
- **Ruta:** define el recorrido de la solicitud y agrupa varios tramos.
- **Tramo:** representa una etapa dentro de la ruta (depósito, traslado o entrega).
- **Contenedor:** identifica la carga transportada con su peso y volumen.
- **TipoEstado** y **TipoTramo:** clasifican los estados y etapas del proceso.

El archivo `data.sql` precarga información base como tipos de estado, tramos y una solicitud de ejemplo, facilitando las pruebas iniciales.

---

6. Seguridad y autenticación

El microservicio incorpora **Spring Security** configurado para funcionar como **Resource Server OAuth2**, preparado para integrar **Keycloak** como proveedor de identidad.
De esta manera, cada petición HTTP puede autenticarse mediante un **token JWT**, garantizando acceso seguro a los recursos.

En entornos de desarrollo, las propiedades de seguridad permanecen comentadas para facilitar las pruebas sin requerir autenticación.

---

7. Observabilidad y manejo de errores

Para mantener la trazabilidad y facilitar el diagnóstico de fallas, el proyecto incluye un manejo global de excepciones mediante la clase `GlobalExceptionHandler`, basada en `@ControllerAdvice`.
Esto permite capturar, registrar y devolver respuestas de error con un formato homogéneo, mejorando la mantenibilidad y experiencia de los consumidores de la API.

Asimismo, la configuración de **Logback (`logback-spring.xml`)** permite registrar los eventos del sistema tanto en consola como en archivos rotativos diarios, categorizados por nivel de severidad (`INFO`, `DEBUG`, `WARN`, `ERROR`).
Esta práctica se alinea con las buenas prácticas vistas en la **Unidad 11 – Logging y Excepciones** del programa.

---

8. Despliegue y contenerización

El proyecto incluye un **Dockerfile** que define la imagen del microservicio, permitiendo su ejecución aislada y reproducible.
El archivo `application-docker.properties` configura la conexión con la base de datos `solicitudes-db` dentro del entorno Docker, adaptando automáticamente las URLs de los Feign Clients.

Esta estrategia asegura la **portabilidad y consistencia del entorno**, evitando el clásico problema de "en mi máquina funciona" y garantizando despliegues estables en cualquier servidor.

---

9. Buenas prácticas aplicadas

El desarrollo sigue los principios fundamentales de ingeniería backend:

- **Separación de capas y responsabilidades**.
- **Uso de DTOs** para evitar exposición directa de entidades.
- **Control de excepciones y mensajes estandarizados**.
- **Uso de dependencias de Spring Boot y Spring Cloud** para reducir complejidad.
- **Inyección de dependencias y desacoplamiento** mediante `@Service`, `@Repository` y `@FeignClient`.
- **Persistencia ORM con JPA** y configuración declarativa.
- **Despliegue contenerizado con Docker**.
- **Documentación automática de endpoints con Swagger (OpenAPI)**.
- **Configuración flexible por entorno**.

10. Conclusión

El microservicio **ms-solicitudes** representa una implementación completa y madura de los conceptos de **backend moderno con Java y Spring Boot**, integrando:

- Lógica de negocio estructurada.
- Persistencia segura y eficiente.
- Comunicación entre microservicios desacoplada.
- Seguridad, logging, documentación y despliegue.

En conjunto, el sistema permite gestionar solicitudes de transporte de manera escalable, modular y mantenible, demostrando la correcta aplicación de los principios de **Backend de Aplicaciones**, **arquitectura de microservicios**, y **DevOps mediante contenerización**.