Documentación del Sistema de Microservicios Imaginemos

# 1. Introducción

Este documento describe la arquitectura, implementación y funcionamiento del sistema de microservicios desarrollado para la empresa Imaginemos. El sistema incluye servicios independientes para la gestión de clientes, proveedores, facturación, inventario y autenticación, comunicándose a través de TCP y HTTP.

# 2. Arquitectura General

El sistema se compone de los siguientes microservicios:  
- Auth Service: gestiona autenticación y roles.  
- Supplier Service: CRUD de proveedores.  
- Client Service: CRUD de clientes.  
- Inventory Service: CRUD de productos e inventario.  
- Billing Service: creación de facturas y gestión de devoluciones.  
- API Gateway: punto único de entrada que enruta y balancea las solicitudes.

Cada servicio tiene su propia base de datos y comunica con otros microservicios a través de TCP.

- Se usa TCP mediante sockets entre billing-service e inventory-service.  
- Los mensajes son JSON estructurados con campos:  
 {  
 "action": "actualizacion",  
 "data": {  
 "nombre": "Silla de Playa",  
 "cantidad": 2  
 }  
 }  
- El inventory-service responde con JSON indicando éxito o error de la operación.

# 2.1 Estructura del proyecto

Tabla

El contenido generado por IA puede ser incorrecto.

# 3. Estructura de Base de Datos

Cada microservicio tiene su propia base de datos, aislada, conectada vía Docker Compose:

- client-service: tabla `clients`  
- inventory-service: `inventory`, `inventory\_transactions`  
- billing-service: `invoices`

Se han configurado archivos init.sql para crear las tablas necesarias en cada servicio.

# 4. Comunicación entre Microservicios (TCP)

Para la comunicación entre servicios (Billing -> Inventory), se ha implementado un servidor TCP y un cliente TCP que intercambian mensajes en formato JSON. Cada mensaje contiene una acción (crear, listar, actualizar, etc.) y los datos relevantes.

# 5. Endpoints expuestos por el API Gateway

- POST /billing/crear → Crea una factura y actualiza inventario  
- GET /billing/listar → Lista todas las facturas registradas  
- GET /billing/devoluciones → Lista devoluciones  
- POST /billing/devoluciones → Registra una devolución de producto  
- GET /clients → Lista de clientes  
- GET /suppliers → Lista de proveedores

# 6. Descripción de Microservicios

## 6.1 Auth Service

Maneja usuarios, roles y autenticación JWT con Spring Security y PostgreSQL.

1. Client-Service  
 - Funcionalidad: Registro y consulta de clientes.  
 - Base de datos: MySQL (clientdb)  
 - Endpoints vía Gateway:  
 \* GET /clients  
 \* POST /clients  
  
2. Inventory-Service  
 - Funcionalidad:  
 \* Crear, listar y actualizar productos.  
 \* Disminuir/Aumentar stock por ventas o devoluciones.  
 - Comunicación TCP en el puerto 4002 (no expone endpoints HTTP directamente).  
  
3. Billing-Service  
 - Funcionalidad:  
 \* Crear, actualizar, eliminar y listar facturas.  
 \* Gestionar devoluciones.  
 \* Actualizar stock comunicándose por TCP con inventory-service.  
 - Endpoints vía Gateway:  
 \* POST /billing/crear  
 \* GET /billing/listar  
 \* POST /billing/devolver  
  
4. Supplier-Service y Auth-Service  
 - Funcionalidad: Gestión de proveedores y autenticación de usuarios.  
 - Autenticación con JWT.  
  
5. Gateway-Service  
 - Expone todos los endpoints vía HTTP en el puerto 8085.  
 - Responsable de enrutar peticiones hacia cada microservicio.

# 7. Docker Compose

El sistema se despliega con Docker Compose. Cada microservicio y su base de datos se definen en un archivo docker-compose.yml. Las bases de datos incluyen healthchecks para garantizar que estén listas antes de levantar los servicios correspondientes.

# 8. Errores y Mecanismo de Recuperación

Los servicios TCP implementan manejo de errores centralizado devolviendo mensajes JSON con claves de error. El gateway también captura errores y los transforma en respuestas HTTP consistentes.

- Todas las respuestas desde los manejadores de solicitudes siguen el formato JSON:  
 \* Éxito:  
 {  
 "message": "Factura registrada",  
 "factura": {...},  
 "inventario": "Stock actualizado"  
 }  
  
 \* Error:  
 {  
 "error": "Producto no encontrado"  
 }

# 9. Pruebas Unitarias y de Integración

Las pruebas se enfocan en validar:

1. Lógica de los handlers TCP

2. Conexiones simuladas con mocks de sockets

3. Persistencia en las bases de datos a través de los repositorios

Ubicación sugerida:

src/test/java/org/imaginemos/{servicio}/handler/BillingRequestHandlerTest.java

src/test/java/org/imaginemos/{servicio}/tcp/BillingTcpClientTest.java

Se utilizan JUnit y Mockito. Las pruebas pueden incluir:

- Envío correcto de JSON

- Manejo de errores por conexión

- Validación de respuestas esperadas

- Mock de repository.save() y repository.findAll()

# 10. Seguridad

Se utiliza Spring Security con JWT para proteger los endpoints REST a través del gateway. Solo usuarios autenticados pueden acceder a servicios como creación de proveedores o facturas.

# 11. Escalabilidad y Balanceo de Carga

El sistema permite escalar horizontalmente duplicando instancias de servicios y exponiéndolas a través de TCP. El balanceo de carga puede implementarse con un componente externo (como Nginx o HAProxy) configurado para TCP.

# 12. conclusión

Este proyecto demuestra una arquitectura distribuida con microservicios, comunicación eficiente por TCP, integración con bases de datos y pruebas unitarias confiables.