



Universidad de las Fuerzas Armadas ESPE

Departamento de Ciencias de la Computación

Carrera de Software

Aplicaciones Distribuidas

Laboratorio N°1

Estudiante: Kleber Chavez, Pamela Chipe, Jhordy Marcillo,
Camilo Orrico

Docente: Ing. Angel Geovanny Cudco Pomagualli

17 de diciembre de 2025

Índice General

1	Introducción	1
1.1	Contexto	1
1.2	Justificación	1
1.3	Objetivos del estudio	1
2	Objetivos	2
2.1	Objetivo general	2
2.2	Objetivos específicos	2
3	Metodología	3
3.1	Enfoque de desarrollo	3
3.2	Arquitectura implementada	3
3.3	Pasos seguidos para la implementación	3
3.3.1	Paso 1: Creación de proyectos Spring Boot por microservicio . .	3
3.3.2	Paso 2: Implementación del servicio de autenticación (LogiFlow)	4
3.3.3	Paso 3: Implementación de Fleet Service	4
3.3.4	Paso 4: Implementación de Billing Service	4
3.3.5	Paso 5: Configuración del API Gateway	5
3.3.6	Paso 6: Pruebas y evidencias	5
4	Resultados	6
4.1	Evidencia de estructura del proyecto	6
4.2	Resultados del servicio de autenticación (LogiFlow)	6
4.2.1	Componentes implementados	6
4.2.2	Capturas de ejecución	6
4.3	Resultados del microservicio fleet-service	6
4.3.1	Componentes implementados	6
4.3.2	Capturas de ejecución	7
4.4	Resultados del microservicio billing-service	7
4.4.1	Componentes implementados	7
4.4.2	Capturas de ejecución	7
4.5	Resultados del API Gateway	7
4.5.1	Configuración y enrutamiento	7
4.5.2	Capturas de ejecución	9
4.6	Resultados de interfaz mínima (HTML)	10
4.6.1	Capturas	10

5	Discusión	12
5.1	Interpretación de resultados	12
5.2	Relación con el enunciado del proyecto	12
5.3	Limitaciones	12
6	Conclusiones	13
6.1	Conclusiones principales	13
6.2	Recomendaciones	13

Lista de Figuras

Figura 4.1	Prueba de login y obtención de JWT	6
Figura 4.2	Logs de ejecución del servicio de autenticación	6
Figura 4.3	Consumo de endpoints de fleet-service mediante API Gateway	7
Figura 4.4	Prueba de endpoints de billing-service	7
Figura 4.5	Evidencia de enrutamiento y validación de acceso mediante Gateway	9

Lista de Tablas

Introducción

1.1. Contexto

LogiFlow se plantea como una plataforma integral para la gestión de operaciones de una empresa de delivery multinivel, donde existen múltiples modalidades de entrega (urbanas, intermunicipales y nacionales) y se requiere centralizar el ciclo de vida completo del pedido, desde la recepción hasta la confirmación de entrega. Bajo este enfoque, una arquitectura basada en microservicios permite separar responsabilidades, habilitar despliegues independientes y mejorar escalabilidad y mantenibilidad, especialmente cuando se integran servicios como autenticación, flota, pedidos y facturación.

En el contexto de Aplicaciones Distribuidas, este proyecto se alinea con principios de separación de responsabilidades, interoperabilidad por contratos (APIs), y comunicación segura entre componentes. Además, se busca un diseño preparado para crecimiento, integraciones futuras y control operativo.

1.2. Justificación

La gestión manual o con herramientas aisladas genera inconsistencias, baja trazabilidad y poca visibilidad en tiempo real. Por ello, la adopción de microservicios y un punto único de entrada (API Gateway) permite estandarizar el acceso a los servicios, aplicar seguridad y auditoría centralizada, y facilitar la evolución del sistema.

1.3. Objetivos del estudio

De manera general, el propósito de este trabajo es diseñar e implementar una arquitectura de microservicios para LogiFlow, incorporando un API Gateway, autenticación basada en JWT, y servicios backend para flota y facturación, de modo que el sistema exponga funcionalidades mediante APIs REST y asegure el control de acceso por roles.

Objetivos

2.1. Objetivo general

Diseñar e implementar LogiFlow como una arquitectura de microservicios que soporte operaciones de delivery multinivel, incorporando un API Gateway y autenticación basada en JWT para garantizar acceso seguro, escalabilidad y separación de responsabilidades.

2.2. Objetivos específicos

- Implementar un servicio de autenticación (Auth) encargado de gestionar usuarios, roles y emisión/validación de tokens JWT.
- Implementar el microservicio de flota (Fleet Service) para gestionar repartidores y vehículos, modelando jerarquías de vehículos y estados operativos.
- Implementar el microservicio de facturación (Billing Service) para el cálculo y registro de facturas en estado inicial.
- Configurar un API Gateway como punto único de entrada para enrutar solicitudes hacia los microservicios y aplicar filtros de seguridad.
- Generar evidencias de ejecución (capturas y pruebas) que demuestren el funcionamiento de endpoints y el flujo de autenticación.

Metodología

3.1. Enfoque de desarrollo

El proyecto se desarrolló bajo un enfoque incremental, construyendo servicios independientes y verificables, conectados mediante un API Gateway. La arquitectura de microservicios permite aislar la lógica de negocio por dominio, reducir acoplamiento y facilitar pruebas por componente.

3.2. Arquitectura implementada

La solución se estructuró en múltiples módulos, cada uno como proyecto Spring Boot independiente:

- **LogiFlow (Auth Service):** módulo responsable de autenticación y autorización con JWT, configuraciones de seguridad y endpoints de login/registro.
- **fleet-service:** microservicio de gestión de flota, con modelos de vehículos, repartidores y estados.
- **billing-service:** microservicio de facturación, con entidad de facturas y estado.
- **gateway:** punto único de entrada para enrutar tráfico hacia los microservicios.
- **HTML/index.html:** interfaz mínima de prueba/consumo (si aplica) para evidenciar peticiones.

Esta organización responde al requerimiento de contar con un backend basado en microservicios, expuesto mediante APIs REST y centralizado por un API Gateway con autenticación JWT y roles.

3.3. Pasos seguidos para la implementación

3.3.1. Paso 1: Creación de proyectos Spring Boot por microservicio

Se crearon proyectos independientes con Maven, cada uno con su propia estructura estándar: `controller`, `service`, `repository`, `model` y `resources`. Esto permite organizar el código por capas y mantener separación de responsabilidades.

3.3.2. Paso 2: Implementación del servicio de autenticación (LogiFlow)

En el módulo LogiFlow se estructuró un paquete `auth` con:

- **config:** configuración de seguridad (por ejemplo, filtros, rutas públicas/privadas).
- **controller:** endpoints de autenticación (login/registro).
- **model:** entidades `Usuario`, `Rol` y enumeración `RolNombre`.
- **repository:** repositorios JPA para persistencia.
- **service:** lógica de autenticación y carga de usuarios.

El objetivo fue emitir y validar JWT para aplicar control de acceso por roles (cliente, repartidor, supervisor, etc.).

3.3.3. Paso 3: Implementación de Fleet Service

En `fleet-service` se implementó:

- Modelos de vehículo y repartidor, incluyendo tipos/estados mediante enumeraciones.
- Controlador REST para endpoints CRUD o de consulta.
- Servicio para reglas de negocio (disponibilidad, estado, registro).
- Repositorios para persistencia.

Esto se alinea con la gestión de flota indicada en los entregables y funcionalidades backend.

3.3.4. Paso 4: Implementación de Billing Service

En `billing-service` se implementó:

- Entidad `Billing` y estado (enumeración `EstadoType`).
- Controlador REST con endpoints base de facturación.
- Servicio con la lógica de creación/cálculo mínimo.
- Repositorio para persistencia.

Esto corresponde al requerimiento de incluir facturación mínima en la Fase 1.

3.3.5. Paso 5: Configuración del API Gateway

En el módulo `gateway` se configuró el enrutamiento para direccionar peticiones a: `auth`, `fleet-service` y `billing-service`. En este paso normalmente se definen:

- Rutas por prefijo (ej. `/api/fleet/**`).
- Filtros de seguridad (validación JWT).
- Logging, y opcionalmente rate limiting.

Esto coincide con el criterio de Gateway como punto único de entrada.

3.3.6. Paso 6: Pruebas y evidencias

Se realizaron pruebas con cliente HTTP (Postman/Insomnia o navegador) para:

- Autenticarse y obtener JWT.
- Consumir endpoints protegidos de flota y facturación.
- Verificar el enrutamiento a través del Gateway.

Se recopilaron capturas de ejecución y logs como evidencia.

Resultados

4.1. Evidencia de estructura del proyecto

La implementación se organizó en módulos independientes: `billing-service`, `fleet-service`, `gateway`, `LogiFlow (auth)` y un recurso `HTML/index.html`. Esta estructura permite mantener servicios desacoplados y desplegados por separado.

4.2. Resultados del servicio de autenticación (Logi-Flow)

4.2.1. Componentes implementados

- `SecurityConfig`
- `AuthController`
- `AuthService` y `UserDetailsServiceImpl`
- `Usuario`, `Rol`, `RolNombre`
- `UsuarioRepository`, `RolRepository`

4.2.2. Capturas de ejecución

Figura 4.1: Prueba de login y obtención de JWT

Figura 4.2: Logs de ejecución del servicio de autenticación

4.3. Resultados del microservicio fleet-service

4.3.1. Componentes implementados

- `FleetController`
- `FleetService`
- `Vehiculo`, `Repartidor`, subtipos (`Moto`, `Liviano`, `Camion`)
- Enumeraciones: `TipoVehiculo`, `EstadoVehiculo`, `TipoLicencia`, etc.
- Repositorios: `VehiculoRepository`, `RepartidorRepository`

4.3.2. Capturas de ejecución

Figura 4.3: Consumo de endpoints de fleet-service mediante API Gateway

4.4. Resultados del microservicio billing-service

4.4.1. Componentes implementados

- BillingController
- BillingService
- Billing, EstadoType
- BillingRepository

4.4.2. Capturas de ejecución

Figura 4.4: Prueba de endpoints de billing-service

4.5. Resultados del API Gateway

4.5.1. Configuración y enrutamiento


```
server:
  port: 8080

spring:
  application:
    name: gateway
  cloud:
    gateway:
      globalcors:
        cors-configurations:
          '[/**]':
            allowedOrigins: "*"
            allowedMethods:
              - GET
              - POST
              - PUT
              - DELETE
            allowedHeaders: "*"

```

```
routes:
- id: auth-service
  uri: http://localhost:8081
  predicates:
    - Path=/auth/**
- id: pedido-service
  uri: http://localhost:8082
  predicates:
    - Path=/pedidos/**
- id: fleet-service
  uri: http://localhost:8083
  predicates:
    - Path=/fleet/**
- id: billing-service
  uri: http://localhost:8084
  predicates:
    - Path=/billing/**
```

4.5.2. Capturas de ejecución



imagenes/gateway_routes.png

Figura 4.5: Evidencia de enrutamiento y validación de acceso mediante Gateway

4.6. Resultados de interfaz mínima (HTML)

4.6.1. Capturas

[Info] Sistema listo.

1. Autenticación

Usuario / Contraseña

Usuario (ej: admin)

Contraseña

Iniciar Sesión

Usuario Conectado ID: **Ninguno**

REGISTRO NUEVO USUARIO

Datos de Registro

Username

Email

Password

Rol

Ciente

Registrarse

GESTIÓN (ADMIN)

ID User

Listar

Eliminar Usuario

2. Pedidos

Crear Pedido

Descripción del paquete

Origen

Destino

Crear Pedido

GESTIÓN DE PEDIDOS

Buscar / Operar por ID Pedido

Ej: 1

Buscar

Listar Todos

Actualizar Estado

EN_RUTA

Actualizar

Eliminar Pedido

3. Vehículos

Registrar Nuevo Vehículo

Moto

Datos Generales

Placa (Ej: PBA-1)

Marca

Modelo

Color

Año (Ej: 2023)

Cilindraje

Detalles de Moto

DEPORTIVA

¿Tiene Casco?

Registrar Vehículo

GESTIÓN DE FLOTA

Operar por Placa

Placa

Buscar

Listar

Estado

DISPONIBLE

Actualizar

Eliminar Vehículo

4. Repartidores

Nuevo Repartidor

Cédula Válida

Nombre

Apellido

Seleccione Li

Teléfono

Registrar

ASIGNACIÓN DE RECURSO

ID Repartidor y Placa Vehículo

ID Repartidor

Placa

Asignar Vehículo (PUT)

BÚSQUEDA Y ELIMINACIÓN

Valor de Búsqueda (ID o Cédula)

Ej: 1 ó 17100...

Por ID

Por Cédula

Eliminar (Usando el ID)

Iniciar Sesión

Usuario Conectado ID: Ninguno

REGISTRO NUEVO USUARIO

Datos de Registro

jpmarcillo2

jpmarcillo@espe.edu.ec

Rol

Ciente

Registrarse

GESTIÓN (ADMIN)

ID User

Listar

Eliminar Usuario

Crear Pedido

GESTIÓN DE PEDIDOS

Buscar / Operar por ID Pedido

2

Actualizar Estado

EN_RUTA Actualizar

Eliminar Pedido

Registrar nuevo Vehículo

Liviano (Auto)

Datos Generales

PCA-1234	ASAS
SASA	PLATA
2017	1400

Detalles Liviano

SEDAN

SUPER MANUAL

5 5

1200

Registrar Vehículo

GESTIÓN DE FLOTA

Operar por Placa

PCA-1234

Buscar Listar

Estado

DISPONIBLE Actualizar

Eliminar Vehículo

Registrar nuevo Repartidor

1753866878

Juanito juanillo

Tipo B 09999999

Registrar

ASIGNACIÓN DE RECURSO

ID Repartidor y Placa Vehículo

ID Repartidor Placa

Asignar Vehículo (PUT)

BÚSQUEDA Y ELIMINACIÓN

Valor de Búsqueda (ID o Cédula)

Ej: 1 ó 17100...

Por ID Por Cédula

Eliminar (Usando el ID)

5. Facturación

Generar Nueva Factura

ID del Pedido a cobrar

Subtotal (\$)

Generar Factura

GESTIÓN DE FACTURAS

Operar por ID Factura

6

Listar Todas

Buscar ID Eliminar

Discusión

5.1. Interpretación de resultados

Los resultados evidencian que la separación por microservicios permite encapsular funcionalidades por dominio (autenticación, flota, facturación) y facilita la verificación individual de cada componente. El uso de un API Gateway centraliza el acceso y establece un punto controlado para aplicar seguridad y enrutamiento, lo cual es consistente con arquitecturas modernas de sistemas distribuidos.

5.2. Relación con el enunciado del proyecto

La solución implementada se alinea con la fase inicial de servicios REST y Gateway, incorporando autenticación basada en JWT y una organización por capas (controlador/service/repository/model), lo cual permite escalar hacia fases posteriores (GraphQL, mensajería y WebSocket) conforme el cronograma planteado. :contentReference[oaicite:8]index=8

5.3. Limitaciones

Se identifican limitaciones típicas de una fase inicial:

- Integración parcial de servicios (por ejemplo, sin mensajería asíncrona ni WebSockets).
- Evidencia basada en pruebas manuales (Postman/logs) en lugar de pruebas automatizadas extensivas.
- Validaciones y políticas de rate limiting en Gateway pueden depender del tiempo disponible.

Conclusiones

6.1. Conclusiones principales

Se implementó una base funcional orientada a microservicios, donde autenticación, flota y facturación operan como servicios independientes, integrados mediante un API Gateway. Esta arquitectura facilita la evolución hacia requerimientos avanzados como monitoreo en tiempo real, mensajería y consultas complejas.

6.2. Recomendaciones

- Incorporar pruebas de integración automatizadas (por servicio y end-to-end vía Gateway).
- Añadir documentación OpenAPI (Swagger) en cada microservicio para evidenciar contratos.
- Evolucionar a la Fase 2 incorporando GraphQL, mensajería (RabbitMQ/Kafka) y WebSocket según el cronograma del curso. :contentReference[oaicite:9]index=9

Bibliografía

newman2021

- [1] Newman, S. (2021). *Building Microservices: Designing Fine-Grained Systems* (2nd ed.). O'Reilly Media.

richardson2018

- [2] Richardson, C. (2018). *Microservices Patterns: With Examples in Java*. Manning Publications.

bass2012

- [3] Bass, L., Clements, P., & Kazman, R. (2012). *Software Architecture in Practice* (3rd ed.). Addison-Wesley.

fielding2000

- [4] Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures* (Doctoral dissertation). University of California, Irvine.

rfc7519

- [5] Jones, M., Bradley, J., & Sakimura, N. (2015). RFC 7519: JSON Web Token (JWT). IETF.

springboot

- [6] VMware Spring Team. (s. f.). *Spring Boot Reference Documentation*.

springsecurity

- [7] VMware Spring Team. (s. f.). *Spring Security Reference Documentation*.

springcloudgateway

- [8] VMware Spring Team. (s. f.). *Spring Cloud Gateway Reference Documentation*.

dame