



Universidad de las Fuerzas Armadas ESPE

Departamento de Ciencias de la Computación

Carrera de Ingeniería en Software

Aplicaciones Distribuidas

Proyecto Integrador Parcial II

Docente: Geovanny Cudco

3 de diciembre de 2025

1 Tema

LogiFlow – Plataforma Integral de Gestión de Operaciones para una Empresa de Delivery Multinivel.

2 Descripción General:

La empresa EntregaExpress S.A. es una compañía de logística y delivery que opera en tres modalidades de servicio:

- a. Entregas urbanas rápidas (última milla) mediante motorizados,
- b. Entregas intermunicipales dentro de la provincia con vehículos livianos (autos o camionetas),
- c. Entregas nacionales mediante furgonetas o camiones medianos/grandes.

Actualmente, la empresa gestiona estas operaciones mediante procesos manuales, hojas de cálculo y aplicaciones independientes no integradas, lo que genera errores en la trazabilidad, demoras en la asignación de rutas, inconsistencias en el estado de los pedidos y baja visibilidad operativa en tiempo real.

Con el fin de modernizar sus operaciones, EntregaExpress desea desarrollar una plataforma centralizada y escalable, basada en microservicios, que permita gestionar de

forma eficiente el ciclo de vida completo de un delivery: desde la recepción del pedido hasta la confirmación de entrega, incluyendo la gestión de flota, conductores, clientes, zonas de cobertura y facturación.

La plataforma debe contar con un panel de control web para supervisores y gerentes, así como APIs para integración con apps móviles de repartidores y clientes, y para futuras integraciones con sistemas de terceros (ERP, CRM, sistemas de pago, etc.).

3 Objetivo

Diseñar e implementar LogiFlow, una arquitectura de microservicios que soporte las operaciones multinivel de EntregaExpress, garantizando alta disponibilidad, escalabilidad, consistencia transaccional y seguridad.

La aplicación debe:

- Permitir el registro y seguimiento en tiempo real de pedidos, con actualizaciones dinámicas mediante WebSockets (p. ej., estado del pedido, ubicación del repartidor, alertas).
- Exponer funcionalidades mediante APIs REST (para operaciones CRUD simples y sincronización con apps móviles) y APIs GraphQL (para consultas complejas y personalizadas desde el panel de control y dashboards).
- Utilizar un API Gateway como punto único de entrada, encargado del enrutamiento, autenticación, rate limiting y composición de servicios.
- Implementar autenticación y autorización basada en JWT, con roles diferenciados: cliente, repartidor (motorizado/vehículo/camión), supervisor, gerente, administrador del sistema.
- Asegurar la integridad de operaciones críticas (p. ej., asignación de pedido + reducción de stock + creación de ruta) mediante transacciones ACID en contextos locales y, cuando sea necesario, coordinar flujos distribuidos con el patrón Saga (p. ej., cancelación de pedido después de despacho parcial).
- Aplicar principios de diseño orientado a objetos: usar clases abstractas para modelar comportamientos comunes pero no instanciables (p. ej.: *VehiculoEntrega*, con subclases *Motorizado*, *VehiculoLiviano*, *Camion*), y interfaces para definir contratos de comportamiento interoperable (p. ej.: *IProcesableEntrega*, *IRegistrableGPS*, *INotificable*).
- Incorporar al menos tres patrones de diseño relevantes (p. ej.: *Factory* para creación de tipos de entrega, *Observer* para notificaciones de estado, *Command* para operaciones reversibles en sagas).

4 Funcionalidades

El sistema *LogiFlow* distribuye sus capacidades entre capas especializadas. A continuación, se detallan las funcionalidades por ámbito de responsabilidad.

4.1 Funcionalidades del Backend

El backend, basado en una arquitectura de microservicios, proporciona la lógica de negocio, garantiza consistencia transaccional y expone interfaces estandarizadas:

- **Autenticación y autorización centralizada:** El `AuthService` emite JWT con `claims` estructurados (`role`, `scope`, `zone_id`, `fleet_type`) y valida todas las peticiones en el API Gateway. Soporta rotación y revocación de tokens.
- **Gestión transaccional de pedidos:** El `PedidoService` permite crear, actualizar y cancelar pedidos. Cada operación crítica (ej. confirmación de entrega) se ejecuta mediante una transacción ACID local; operaciones distribuidas (ej. cancelación post-despacho) se coordinan con el patrón *Saga Orquestada*.
- **Asignación dinámica de flota y rutas:** El `FleetService` y `RoutingService` colaboran para asignar repartidores aptos según tipo de vehículo, disponibilidad y proximidad. Utilizan una jerarquía de clases: `VehiculoEntrega` (abstracta) y sus subclases (`Motorizado`, `VehiculoLiviano`, `Camion`), además de la interfaz `IRuteable` para estandarizar la generación de rutas.
- **Seguimiento geoespacial en tiempo real:** El `TrackingService` recibe actualizaciones GPS mediante API REST (punto final `POST /track`) y publica eventos en un bus de mensajes. Un *WebSocket broadcaster* (acoplado al API Gateway) los redistribuye a clientes suscritos.
- **Notificaciones asíncronas y eventos de negocio:** El `NotificationService` consume eventos desde colas (Kafka/RabbitMQ) y envía alertas por SMS, correo o push. También emite eventos para dashboards y auditoría.
- **Facturación y reportes técnicos:** El `BillingService` calcula tarifas dinámicas, genera facturas electrónicas y expone endpoints para reportes consolidados (diarios, por zona, por tipo de vehículo).
- **Exposición de APIs diferenciadas:**
 - *REST*: para operaciones CRUD y sincronización con apps móviles (ej. `GET /pedidos/{id}`, `PATCH /repartidores/{id}/estado`).
 - *GraphQL*: para consultas complejas y eficientes desde el panel de control (evita múltiples round trips y over-fetching).

4.2 Funcionalidades del Frontend

El frontend se implementa como una aplicación web adaptativa (responsive), con interfaces especializadas por rol y acoplamiento ligero al backend:

- **Autenticación y gestión de sesión:** Interfaz de login con soporte para múltiples roles. Gestión transparente de JWT (almacenamiento seguro en `httpOnly` cookies o `sessionStorage`), renovación automática y logout con invalidación remota.
- **Panel de control por roles:**

- *Cliente*: historial de pedidos, seguimiento en mapa interactivo (con marcadores en tiempo real), chat con repartidor y opción de cancelación.
 - *Repartidor*: lista de asignaciones pendientes/en curso, navegación asistida (integración con maps), escaneo de QR en entrega, reporte de incidencias (foto + descripción).
 - *Supervisor*: mapa de flota en su zona, alertas de retraso (¡15 min), herramienta de reasignación manual con arrastrar-y-soltar, reporte diario descargable.
 - *Gerente/Administrador*: dashboard de KPIs (OTD, costo por entrega, satisfacción), filtros por región/fecha, comparativos mensuales y visualización de logs de sagas fallidas.
- **Visualización en tiempo real mediante WebSocket**: Suscripción automática a canales temáticos según rol y contexto (ej. cliente: `/topic/pedido/P-123`; supervisor: `/topic/zona/QUITO_NORTE`). Actualización automática del UI sin recarga.
 - **Consultas eficientes con GraphQL**: Uso de clientes GraphQL (ej. Apollo Client) para recuperar datos estructurados y personalizados. Ejemplo: un supervisor puede solicitar métricas de desempeño de sus repartidores con una única consulta, sin campos innecesarios.
 - **Gestión de incidentes y retroalimentación**: Formularios para reporte de problemas (paquete dañado, dirección incorrecta) con adjunto de imagen y geolocalización automática. Estado visible en el historial del pedido.
 - **Exportación y accesibilidad**: Botones de exportación a CSV/PDF en reportes tabulares. Soporte para contraste alto y navegación por teclado (cumpliendo WCAG 2.1 nivel AA).

5 Fases del proyecto

El desarrollo de *LogiFlow* se organiza en tres fases progresivas, que permiten validar la arquitectura de forma incremental y garantizar la calidad en cada capa. Cada fase define un conjunto mínimo viable de componentes funcionales, con énfasis en cohesión, separación de responsabilidades y cumplimiento de requisitos técnicos.

5.1 Fase 1: Backend — Servicios REST para Operaciones CRUD y API Gateway

Esta fase establece la base transaccional y de seguridad del sistema. Debe garantizar integridad, autenticación y acceso controlado a los recursos fundamentales.

- **Microservicios REST con operaciones CRUD básicas**:
 - **AuthService**: registro, login, refresh/revoke token; persistencia de usuarios y roles. Endpoints: `POST /login`, `POST /register`, `POST /token/refresh`.
 - **PedidoService**: creación, consulta, modificación parcial (`PATCH`) y cancelación lógica de pedidos. Validación de tipo de entrega y cobertura geográfica.

- **FleetService**: gestión de repartidores y vehículos (alta, baja, actualización de estado: `DISPONIBLE`, `EN_RUTA`, `MANTENIMIENTO`).
 - **BillingService** (mínimo): cálculo de tarifa básica y generación de factura en estado `BORRADOR`.
- **API Gateway** (ej. Spring Cloud Gateway, Kong o Apigee):
- Enrutamiento por prefijo de ruta (ej. `/api/pedidos/**` → `PedidoService`).
 - Validación de JWT en todas las rutas protegidas; rechazo con `401/403` en caso de error.
 - Rate limiting por cliente (ej. 100 req/min).
 - Logging centralizado de método, URI, código de respuesta y `userId` (si aplica).
- **Requisitos técnicos mínimos**:
- Todas las operaciones de escritura son transacciones ACID (uso de `@Transactional` o equivalente).
 - Validación de esquema de entrada (con anotaciones o librerías como `celebrate/FluentValidation`).
 - Documentación OpenAPI 3.0 accesible en `/swagger-ui.html` o `/docs`.
- **Criterio de aceptación**: Un cliente autenticado puede crear un pedido urbano, y un supervisor puede consultarlo y ver su estado en `RECIBIDO`, usando únicamente endpoints REST y el API Gateway.

5.2 Fase 2: Backend — APIs GraphQL, Mensajería Asíncrona y Comunicación en Tiempo Real

Esta fase enriquece el backend con capacidades de consulta flexible, desacoplamiento y actualización dinámica, preparándolo para escenarios de alta concurrencia y monitoreo operativo.

- **API GraphQL** (ej. Apollo Server, GraphQL.NET, Spring for GraphQL):
- Schema que exponga tipos relacionados: `Pedido`, `Repartidor`, `Vehiculo`, `Zona`, `KPI`.
 - Resolvers eficientes con *data loaders* para evitar el problema $N+1$.
 - Consultas de ejemplo implementadas:
- ```
Dashboard supervisor
query PedidosEnZona($zonaId: ID!, $estado: EstadoPedido) {
 pedidos(filtro: {zonaId: $zonaId, estado: $estado}) {
 id, cliente {nombre}, destino, estado,
 repartidor {nombre, vehiculo {tipo}}
 tiempoTranscurrido, retrasoMin
 }
 flotaActiva(zonaId: $zonaId) {
 total, disponibles, enRuta
 }
}
```

- **Sistema de Mensajería** (RabbitMQ o Kafka):
  - Colas/exchanges definidos para eventos clave: `pedido.creado`, `pedido.estado.actualizado`, `repartidor.ubicacion.actualizada`, `saga.iniciada`.
  - Productores en `PedidoService`, `FleetService` y `TrackingService`.
  - Consumidor en `NotificationService` que dispara alertas (SMS/email) y actualiza cachés.
  - Mensajes idempotentes y con `messageId` para deduplicación.
- **WebSocket Server** (integrado al API Gateway o como servicio independiente):
  - Endpoint `/ws` con handshake JWT-autenticado (token en parámetro de consulta o encabezado).
  - Broadcast selectivo mediante tópicos (ej. `/topic/pedido/P-123`, `/topic/zona/Z-45`).
  - Reenvío de últimos eventos al reconectar (*replay* limitado por TTL).
- **Requisitos técnicos mínimos:**
  - El WebSocket broadcaster consume del bus de mensajes (no de peticiones HTTP directas).
  - La API GraphQL no expone operaciones de mutación críticas (ej. `confirmarEntrega`); éstas permanecen en REST para trazabilidad y control transaccional.
  - Monitoreo de colas (lag, tasa de rechazo) con Prometheus + Grafana.
- **Criterio de aceptación:** Un supervisor recibe, en menos de 2 segundos, una notificación push y una actualización automática en su interfaz cuando un pedido en su zona cambia a estado `EN_RUTA`, gracias a la cadena: REST (actualización) → Kafka → `NotificationService` + WebSocket.

### 5.3 Fase 3: Frontend — Panel de Control Funcional y Experiencia de Usuario

Esta fase integra todas las capas anteriores en una interfaz usable, segura y adaptada a cada actor operativo.

- **Aplicación web monolítica o modular** (React, Angular o Vue), con enrutamiento por roles:
  - Rutas protegidas: redirección automática si el JWT es inválido o falta permiso.
  - Layouts diferenciados: cliente (enfocado en seguimiento), repartidor (enfocado en tareas), supervisor (mapa + lista), gerente (KPIs).
- **Integración con backend:**
  - Llamadas REST para mutaciones (crear pedido, cambiar estado).

- Cliente GraphQL (ej. Apollo) para consultas complejas en dashboards.
  - Conexión WebSocket persistente tras login; suscripción automática a tópicos relevantes según contexto (ej. al abrir un pedido, suscribe a `/topic/pedido/{id}`).
- **Componentes funcionales clave:**
- *Mapa interactivo* (con Leaflet o Mapbox): superposición de repartidores, rutas y zonas; actualización automática vía WebSocket.
  - *Lista de pedidos con filtros dinámicos* (por estado, fecha, zona) y acciones contextuales (reasignar, marcar como entregado).
  - *Formulario de incidencias*: captura de foto, geolocalización automática y envío con adjunto base64.
  - *Exportación de reportes*: botones que generan CSV/PDF mediante llamadas a endpoints del `BillingService` o `PedidoService`.
- **Requisitos técnicos mínimos:**
- Almacenamiento seguro de tokens (evitando `localStorage` para JWT sensibles).
  - Manejo de errores consistente (mensajes claros, sin exponer stack traces).
  - Pruebas de accesibilidad básicas (contraste, etiquetas ARIA, navegación por teclado).
- **Criterio de aceptación:** Un supervisor puede: (1) visualizar en el mapa a tres repartidores en tiempo real, (2) reasignar un pedido arrastrándolo a otro repartidor, (3) ver la actualización inmediata del estado en la lista y en el mapa, y (4) descargar un reporte CSV de sus operaciones del día — todo sin recargar la página.

## 6 Listado de Entregables por Fase

### 6.1 Fase 1: Backend — Servicios REST y API Gateway

1. **Informe técnico** usando la plantilla de *LaTeX* entregada previamente.
2. **Código fuente** de los microservicios: `auth-service`, `pedido-service`, `fleet-service`, `billing-service` (versión mínima), con estructura modular (controladores, servicios, repositorios).
3. **Contratos OpenAPI 3.0** (archivos `.yaml` o `swagger.json`) para cada microservicio, incluyendo ejemplos de solicitud/respuesta y códigos de estado.
4. **API Gateway configurado** (ej. YAML de Spring Cloud Gateway o declaraciones en Kong), con:
  - Enrutamiento a los 4 microservicios por prefijo.
  - Filtro de autenticación JWT (validación de firma, expiración y `claim role`).
  - Rate limiting por clave de cliente (`X-API-Key` o `sub` del token).

5. **Base de datos relacional** (esquema SQL o migraciones Liquibase/Flyway) con tablas para: usuarios, roles, pedidos, repartidores, vehiculos, facturas.
6. **Pruebas unitarias e integración** (JUnit/TestContainers o equivalentes) que cubran:
  - Creación de pedido con validación de tipo de entrega.
  - Asignación de repartidor disponible.
  - Rechazo de petición no autenticada (código 401) o sin permisos (403).
7. **Documento de diseño técnico (mínimo)**: diagrama de arquitectura de alto nivel (componentes + flujo de autenticación) y justificación de decisiones (ej. uso de transacciones locales, no uso de Saga en esta fase).

## 6.2 Fase 2: Backend — GraphQL, Mensajería y WebSocket

1. **Esquema GraphQL** (`schema.graphqls`) con tipos, queries y resolvers para al menos:
  - `pedido(id: ID!): Pedido`
  - `pedidos(filtro: PedidoFiltro): [Pedido!]!`
  - `flotaActiva(zonaId: ID!): FlotaResumen`
  - `kpiDiario(fecha: Date!, zonaId: ID): KPIDIario`
2. **Servidor GraphQL funcional**, con resolvers que eviten N+1 (uso de `DataLoader` o equivalentes) y métricas de rendimiento (caché hit/miss).
3. **Configuración de RabbitMQ/Kafka**: declaraciones de queues/exchanges/topics (archivos `.json` o scripts `.sh`), con políticas de retención y replicación definidas.
4. **Productores y consumidores**:
  - Productor en `PedidoService` que publique evento `pedido.estado.actualizado`.
  - Consumidor en `notification-service` que reciba el evento y registre en log (simulación de envío).
  - Productor en `tracking-service` que publique ubicación.
5. **Servidor WebSocket** (independiente o integrado en API Gateway) con:
  - Endpoint `/ws` que valide JWT en handshake.
  - Mecanismo de broadcast selectivo por tópico (ej. `SimpMessagingTemplate` en Spring, `socket.io rooms` en Node.js).
  - Registro de suscripciones y desconexiones en log.
6. **Pruebas de integración asíncrona**:
  - Simulación de actualización de estado de pedido → verificación de publicación en cola.

- Consumo del mensaje y emisión vía WebSocket (validado con cliente de prueba, ej. `wscat`).
7. **Documentación de flujo de eventos:** diagrama de secuencia (Mermaid o PlantUML) que muestre la cadena: REST update → evento → consumidor → WebSocket broadcast.

## 6.3 Fase 3: Frontend — Panel de Control Funcional

1. **Código fuente de la aplicación frontend** (React/Angular/Vue), con:
  - Sistema de rutas protegidas (lazy-loaded por rol).
  - Servicios de autenticación (login, token refresh, logout).
  - Cliente HTTP para REST (ej. `axios`) y cliente GraphQL (ej. `Apollo Client`).
  - Servicio de WebSocket con reconexión automática y manejo de suscripciones.
2. **Vistas implementadas (mínimo):**
  - Login y registro (cliente/repartidor).
  - Cliente: lista de pedidos + vista de seguimiento con mapa y chat.
  - Repartidor: lista de asignaciones + formulario de confirmación de entrega (con foto).
  - Supervisor: dashboard con mapa, lista filtrable y botón de reasignación.
  - Gerente: vista de KPIs con gráficos (Chart.js o similar) y selector de rango de fechas.
3. **Integraciones verificables:**
  - Al crear un pedido (formulario → REST), se muestra en la lista sin recargar.
  - Al actualizar estado en backend (vía Postman o interfaz del repartidor), el supervisor ve el cambio en el mapa y la lista en ¡2 s (vía WebSocket).
  - Al ejecutar consulta GraphQL en el dashboard del supervisor, se obtienen datos sin múltiples llamadas REST.
4. **Exportación funcional:** botón *Exportar CSV* que descarga un archivo con formato válido (cabeceras, delimitador, codificación UTF-8).
5. **Pruebas de interfaz:** reporte de Lighthouse (accesibilidad 85, performance 75) o equivalente.
6. **Manual de usuario (mínimo):** guía de 1–2 páginas por rol, con capturas y pasos para: crear pedido, seguir entrega, reasignar, generar reporte.

## 7 Cronograma de Entregas

El desarrollo de *LogiFlow* se ejecuta bajo un cronograma de entregas incrementales, con hitos técnicos verificables. Las fechas límite son contundentes y corresponden a la entrega formal de los artefactos definidos en la Sección 6.

| Fase   | Contenido Principal                                           | Fecha Límite de Entrega        |
|--------|---------------------------------------------------------------|--------------------------------|
| Fase 1 | Backend: Servicios REST CRUD + API Gateway                    | <b>15 de diciembre de 2025</b> |
| Fase 2 | Backend: GraphQL, Mensajería (RabbitMQ/Kafka) y WebSocket     | <b>14 de enero de 2026</b>     |
| Fase 3 | Frontend: Panel de Control Funcional y Experiencia de Usuario | <b>13 de febrero de 2026</b>   |

### Notas:

- Las entregas incluyen *código fuente*, *documentación técnica mínima* y *evidencia de pruebas* (logs, capturas, reportes).
- Se aceptan entregas anticipadas para revisión formativa (retroalimentación sin penalización).
- Entregas posteriores a la fecha límite tendrán una penalización del 100 % (no se aceptan).
- En caso de imprevistos institucionales (ej. paro, emergencia), las fechas serán reprogramadas mediante comunicación oficial por escrito.