EXPEDIENTE TÉCNICO DE BACKEND Y ARQUITECTURA DE DATOS

Proyecto: SIGLO-F (Sistema Integral de Gestión Logística y Fidelización)
Tecnología: Java 17 + Spring Boot 3 + PostgreSQL
Versión de API: v1.0 (/api/v1/)
Desarrollado por: Eddam Eloy Huallpa

---

1. ESTÁNDARES DE DESARROLLO (API REST)

Para asegurar consistencia, todos los endpoints seguirán este formato de respuesta JSON

```
{
  "status": 200,
  "message": "Operación exitosa",
  "data": { ... objeto o lista ... },
  "timestamp": "2026-01-11T10:00:00"
}
```

- **Manejo de Errores:** Uso de @ControllerAdvice para capturar excepciones (EntityNotFound, BusinessRuleException) y devolver códigos HTTP correctos (404, 400, 403, 500).
- **Paginación:** Todos los listados (Clientes, Ventas, Logs) deben implementar Pageable de Spring Data.

2. MÓDULO DE SEGURIDAD Y USUARIOS (IAM)

2.1 Modelo de Datos (users, roles)

- **Tabla roles:**
  - id (PK), name (SUPER_ADMIN, ADMIN, REPARTIDOR).
- **Tabla users:**
  - id (UUID), username (unique), password (BCrypt), full_name, status (ACTIVE, INACTIVE), role_id (FK).

2.2 Endpoints Clave

- POST /auth/login: Retorna JWT Token + Refresh Token.
- POST /users (Solo Super Admin): Crea nuevos operadores o repartidores.
- PATCH /users/{id}/password: Reseteo de contraseña.

2.3 Reglas de Negocio

1. **Jerarquía:** Un ADMIN no puede crear ni editar a un SUPER_ADMIN.
2. **Inmutabilidad de Logs:** Cualquier cambio en un usuario queda registrado en la tabla de auditoría.
3. **Bloqueo:** Si un REPARTIDOR es desactivado, su Token JWT debe invalidarse inmediatamente (Blacklist o vigencia corta).

3. MÓDULO DE CLIENTES Y GEOLOCALIZACIÓN

3.1 Modelo de Datos (clients)

- id (UUID).
- document_number (DNI/RUC - Unique).
- name (Razón Social / Nombre).
- address (Texto).
- latitude (Double).
- longitude (Double).
- client_type (Enum: HOGAR, RESTAURANTE, DISTRIBUIDOR).
- status_commercial (Enum: FREQUENT, ACTIVE, RISK, INACTIVE).
- last_purchase_date (Timestamp).
- zone_id (Para agrupar por zonas de reparto).

3.2 Reglas de Negocio (Lógica del Semáforo)

Esta lógica debe correr cada noche (@Scheduled) o al realizar una venta.

1. **VERDE (Frecuente):** last_purchase_date <= 15 días.
2. **GRIS (Activo):** last_purchase_date entre 16 y 35 días.
3. **ROJO (Riesgo/Alerta):** last_purchase_date > 35 días.
4. **Validación Geo:** No permitir registrar cliente sin coordenadas (o poner coordenadas default de la central si es venta en puerta).

4. MÓDULO DE PRODUCTOS E INVENTARIO (ALMACÉN)

4.1 Modelo de Datos

- **products:** id, name (Gas 10kg, 15kg, 45kg), base_price, weight.
- **inventory_main:** Stock físico en planta.
  - product_id, quantity_full, quantity_empty.
- **inventory_movements:** Auditoría de stock.
  - id, type (IN_PURCHASE, OUT_ROUTE, IN_RETURN), quantity, user_id.

4.2 Reglas de Negocio

1. **Ley de Conservación:** El stock no se crea ni se destruye, solo se transforma o mueve.
   - *Compra:* Aumenta quantity_full.
   - *Salida a Ruta:* Disminuye quantity_full en Planta, Aumenta quantity_full en Unidad.
2. **Alertas:** Si quantity_full < min_stock, enviar notificación al Dashboard.

5. MÓDULO DE RUTAS Y VENTAS (EL CORE OPERATIVO)

Este es el módulo más complejo. Maneja el flujo diario del camión.

5.1 Modelo de Datos

- **vehicles:** id, plate, gps_device_id.
- **vehicle_stock:** Stock actual rodando en el camión.
- **routes (Cabecera):**
  - id, vehicle_id, driver_id, start_time, end_time, status (OPEN, CLOSED), initial_load_json, final_return_json.
- **sales (Transacción):**
  - id, route_id, client_id, total_amount, payment_method (CASH, YAPE, CREDIT), timestamp, coordinates_delivery (Lat/Lon donde se hizo la venta).
- **sale_details:** product_id, quantity, unit_price.

5.2 CRUD y Endpoints

- POST /routes/start: Admin asigna carga y chofer. (Mueve stock de Planta -> Vehículo).
- POST /sales (App Móvil): Repartidor registra venta.
  - *Validación:* No se puede vender si vehicle_stock es insuficiente.
- POST /routes/close: Liquidación.

5.3 Reglas de Negocio de Liquidación

Al llamar a /routes/close, el Backend debe calcular:

1. **Balance de Balones:**
   - Carga Inicial - Ventas = Esperado en Retorno.
   - Comparar Esperado vs Retorno Real ingresado por Admin.
   - Si hay diferencia -> Generar alerta de **Pérdida de Balón**.
2. **Balance de Dinero:**
   - Sumatoria de sales (Filtrado por método de pago).
   - Debe coincidir con el dinero entregado.

6. MÓDULO DE FIDELIZACIÓN (CÓDIGOS QR)

6.1 Modelo de Datos (loyalty_cards)

- id (UUID).
- qr_code (String único, hash).
- points_balance (Integer).
- status (Enum: **AVAILABLE, ASSIGNED, BLOCKED**).
- client_id (FK, Nullable al inicio).
- printed_batch_id (Para saber en qué lote se imprimió).

6.2 Flujo de Estados y Reglas

1. **Estado AVAILABLE (Disponible):** El QR fue generado e impreso, pero está en una caja. No tiene dueño.
2. **Transición a ASSIGNED (Asignación):**
   - Endpoint: POST /loyalty/assign.
   - El Repartidor escanea el QR y selecciona al Cliente en la App.
   - Validación: El cliente NO debe tener ya una tarjeta activa. (Solo 1 tarjeta activa por cliente).
3. **Acumulación:**
   - Cada vez que se registra una sale para ese client_id, un trigger o servicio suma puntos a la loyalty_card activa.
   - *Regla:* 1 Balón 10kg = 1 Punto (Configurable).
4. **Estado BLOCKED:**
   - Si el cliente pierde la tarjeta, el Admin la bloquea.
   - Se permite asignar una *nueva* tarjeta y (opcionalmente) transferir los puntos de la vieja a la nueva.

7. MÓDULO DE GPS (INTEGRACIÓN)

7.1 Lógica de Integración

- No guardamos el historial de movimientos de 5 años (demasiada data). Guardamos la **última ubicación** y quizás el historial del día.
- **Tabla gps_logs:** vehicle_id, latitude, longitude, speed, timestamp.

7.2 Servicio en Segundo Plano (Cron Job)

- **Clase:** GpsSyncService.java
- **Frecuencia:** Cada 30 segundos.
- **Acción:** Consulta API externa -> Actualiza tabla vehicles (campos current_lat, current_lon) -> Guarda histórico en gps_logs.

8. MODELO ENTIDAD-RELACIÓN (RESUMEN TÉCNICO)

Para tu diseño en base de datos, estas son las relaciones críticas:

1. **One-to-One:** User <-> Repartidor (Si manejas perfiles extendidos).
2. **One-to-Many:** Client -> Sales.
3. **One-to-Many:** Vehicle -> Routes.
4. **One-to-One (Condicional):** Client <-> LoyaltyCard (Donde status = ASSIGNED).
5. **Many-to-Many:** Sales <-> Products (A través de SaleDetail).

9. REQUERIMIENTOS DE SEGURIDAD BACKEND

1. **CORS:** Configurar para permitir peticiones solo desde el dominio del Frontend y la IP de la oficina (opcional).
2. **Password:** Hash con BCrypt cost 10 o 12.
3. **Inyección SQL:** Uso obligatorio de JPA/Hibernate o PreparedStatement. Prohibido concatenar strings en queries.
4. **Roles en Endpoints:**
    - /admin/** -> Solo ROLE_ADMIN, ROLE_SUPER_ADMIN.
    - /mobile/** -> ROLE_REPARTIDOR.