Documentación Técnica - Sistema de Gestión de Empleados

Arquitectura de Microservicios y Despliegue

Grupo 8: Josue Torres, Alexis Roman

2026-02-06

Table of contents

1 Introducción

El **Sistema de Gestión de Empleados** es una solución integral diseñada para la administración eficiente de recursos humanos dentro de una organización. El sistema centraliza la gestión de identidad,

control biométrico de asistencia, asignación de turnos y el cálculo automatizado de nómina, resolviendo la necesidad de integrar procesos dispersos en una sola plataforma.

2 Arquitectura del Sistema

El sistema adopta un patrón de arquitectura de **Microservicios**, priorizando el desacoplamiento, la escalabilidad horizontal y la independencia tecnológica de cada módulo.

2.1 Diseño de Alto Nivel

La comunicación externa hacia el sistema se gestiona exclusivamente a través de un **API Gateway (Kong)**, mientras que la comunicación interna entre servicios se realiza mediante peticiones HTTP síncronas (REST) en una red privada.

2.1.1 Componentes Principales

1. **Capa de Presentación (Frontend)**
   - **Cliente Móvil (Flutter):** Enfocada en el empleado para autogestión y marcación biométrica.
   - **Cliente Web (Svelte):** Panel administrativo para gestión de recursos y reportes.
2. **Capa de Entrada (Ingress)**
   - **Kong API Gateway:** Actúa como proxy inverso, punto único de entrada, manejo de enrutamiento, rate-limiting y terminación SSL.
3. **Capa de Servicios (Business Logic)**
   - Conjunto de microservicios desarrollados en **Python 3.12 (FastAPI)** ejecutándose en contenedores Docker.
4. **Capa de Datos (Persistence)**
   - **MariaDB:** Instancias lógicas separadas para garantizar el patrón *Database-per-Service*.

2.2 Flujo de Peticiones (Request Flow)

El flujo típico de una petición sigue la siguiente ruta:

1. Cliente envía petición a `api.midominio.com/v1/resource`.
2. Kong Gateway intercepta la petición en el puerto `8000`.
3. Kong resuelve la ruta (*Route Matching*) hacia el servicio interno correspondiente (*Upstream*).
4. Microservicio procesa la lógica, consultando su base de datos o comunicándose con otros servicios.
5. Respuesta es devuelta al cliente en formato JSON estándar.

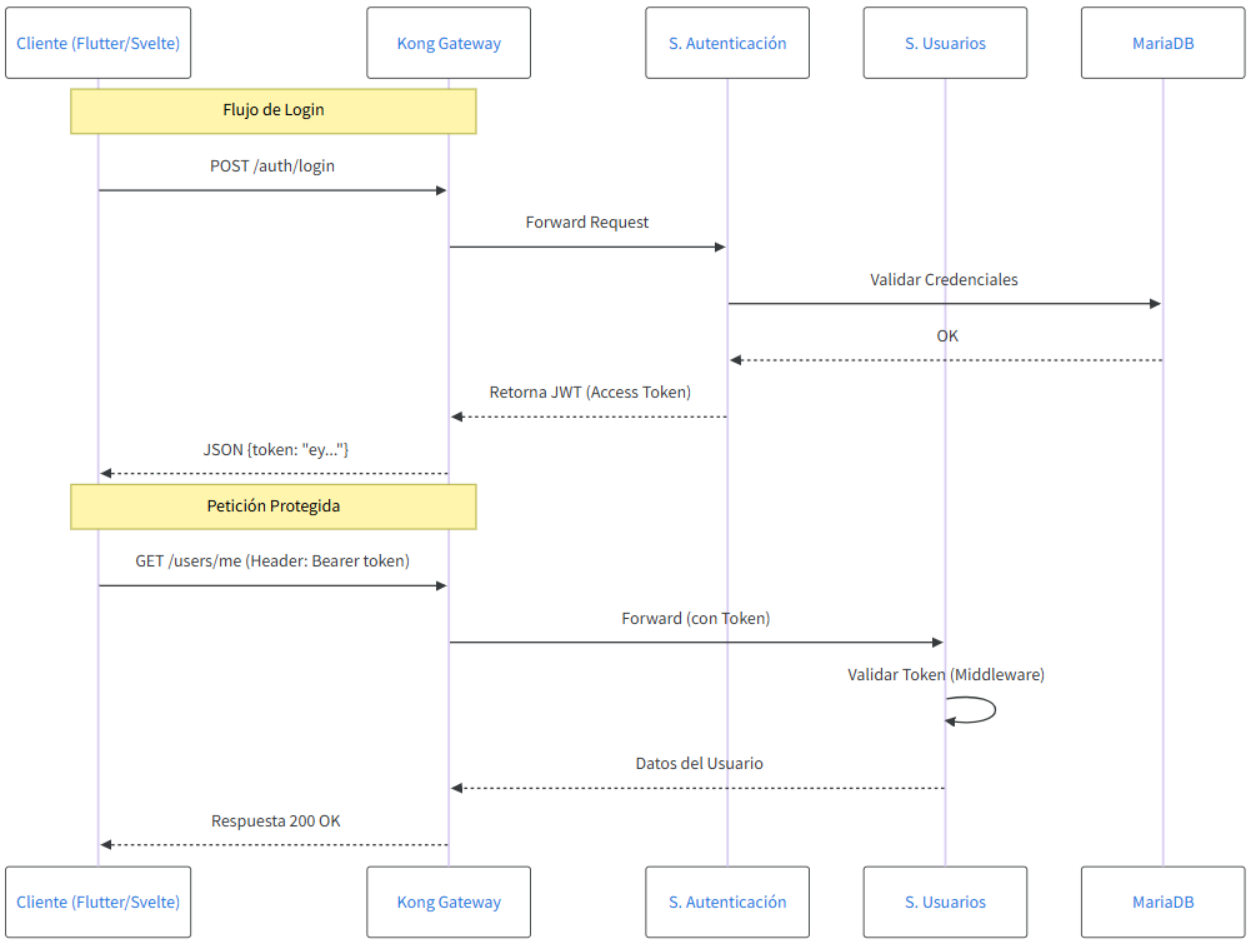

Figure 1: Flujo de Peticiones

3 Especificación de Microservicios

El backend se orquesta mediante **Docker Compose**. Todos los servicios comparten una red interna `backend-network` y no exponen sus puertos directamente a internet, solo al Gateway.

3.1 1. Servicio de Autenticación (`servicio-autenticacion`)

Encargado de la seguridad perimetral y la emisión de credenciales.

- **Puerto Interno:** 9000
- **Dependencias:** `servicio-usuarios` (para validar existencia del usuario).
- **Tecnologías:** Passlib (Hashing Bcrypt), PyJWT.

3.1.1 Endpoints Principales

Método	Ruta	Descripción	Payload
POST	`/auth/login`	Inicia sesión y devuelve tokens	`{email, password}`
POST	`/auth/refresh`	Renueva el token de acceso	`{refresh_token}`

Método	Ruta	Descripción	Payload
POST	`/auth/verify`	Validez del token (Uso interno)	`{token}`

3.2 2. Servicio de Usuarios (`servicio-usuarios`)

La fuente de verdad para la información de los empleados.

- **Puerto Interno:** 9001
- **Responsabilidad:** CRUD de usuarios, gestión de perfiles y roles.

3.2.1 Modelo de Datos (Esquema Simplificado)

- `id`: UUID
- `email`: String (Unique)
- `hashed_password`: String
- `role`: Enum (ADMIN, EMPLEADO, RRHH)
- `department_id`: Integer
- `hourly_rate`: Float (Para cálculo de nómina)

3.2.2 Lógica de Negocio

- No permite eliminar usuarios con registros de asistencia activos (*Soft Delete*).
- Sincronización de datos básicos con el servicio de nómina si es necesario.

3.3 3. Servicio de Asistencia (`servicio-asistencia`)

Registra los eventos temporales de los empleados.

- **Puerto Interno:** 9002

3.3.1 Endpoints Clave

- `POST /attendance/check-in`: Registra entrada. Valida que no exista una entrada abierta sin cierre.
- `POST /attendance/check-out`: Cierra la jornada actual.
- `GET /attendance/report?start_date=&end_date=`: Genera reporte JSON para nómina.

3.3.2 Algoritmo de Validación

```
# Pseudocódigo de validación de entrada
def marcar_entrada(user_id):
    ultimo_registro = db.get_last_record(user_id)
    if ultimo_registro.tipo == 'ENTRADA' and not ultimo_registro.hora_salida:
        raise HTTPException(400, "Ya tiene una jornada abierta")
    # Proceder a crear registro...
```

3.4 4. Servicio de Horarios (`servicio-horarios`)

Gestiona la planificación temporal.

- **Puerto Interno:** 9003
- **Funcionalidad:** Permite definir turnos rotativos.
- **Relación:** Es consumido por el servicio de Asistencia para determinar atrasos (Comparando `hora_entrada_real` vs `hora_inicio_turno`).

3.5 5. Servicio de Nómina (`servicio-nomina`)

El servicio más complejo que actúa como agregador de información.

- **Puerto Interno:** 9004

3.5.1 Lógica de Comunicación entre Servicios

Para generar un rol de pago, este servicio realiza peticiones HTTP internas (S2S):

1. Solicita a **Usuarios**: Salario base, cargo y datos fiscales.
2. Solicita a **Asistencia**: Total de horas trabajadas y horas extra.
3. Calcula: `(Horas * Tarifa) + Bonos - IESS/Impuestos`.

4 Configuración del API Gateway (Kong)

Kong se configura mediante `kong.yaml` (modo DB-less) o mediante el Admin API.

4.1 Definición de Servicios y Rutas

Se utiliza la estrategia de **Path-Based Routing**.

Servicio (Upstream)	Ruta en Gateway (Path)	URL Destino (Docker DNS)
`servicio-autenticacion`	`/api/v1/auth`	`http://servicio-autenticacion:9000`
`servicio-usuarios`	`/api/v1/users`	`http://servicio-usuarios:9001`
`servicio-asistencia`	`/api/v1/attendance`	`http://servicio-asistencia:9002`
`servicio-horarios`	`/api/v1/schedules`	`http://servicio-horarios:9003`
`servicio-nomina`	`/api/v1/payroll`	`http://servicio-nomina:9004`

5 Frontend: Detalles de Implementación

5.1 Aplicación Móvil (Flutter)

- **Seguridad:** Almacenamiento seguro del JWT usando `flutter_secure_storage`.
- **Interceptors:** Uso de *Dio Interceptor* para inyectar automáticamente el header `Authorization: Bearer <token>` en cada petición y manejar errores 401 (Token expirado) redirigiendo al Login.
- **Biometría:** Implementación de `local_auth` para desbloquear el token almacenado sin reingresar contraseña.

5.2 Panel Web (Svelte)

- **Stores:** Uso de writable stores para mantener la sesión del usuario persistente en `localStorage`.
- **Vite Proxy:** Configuración de `vite.config.js` para redirigir `/api` al Gateway durante el desarrollo local, evitando problemas de CORS.

6 Guía de Despliegue y Ejecución

6.1 Requisitos Previos

- Docker Engine 20.10+
- Docker Compose v2+
- Node.js 18+ (Para frontend)
- Flutter SDK 3.19+ (Para móvil)

6.2 Pasos de Ejecución (Backend)

1. **Clonar Repositorio:**

```
git clone https://github.com/grupo8/gestion-empleados.git
cd gestion-empleados
```

2. **Configurar Variables de Entorno:**

   Crear un archivo `.env` en la raíz basado en `.env.example`.

```
DB_PASSWORD=secret_secure_password
JWT_SECRET=llave_maestra_para_tokens
KONG_ADMIN_URL=http://localhost:8001
```

3. **Levantar Stack de Microservicios:**

```
docker-compose up -d --build
```

   Esto iniciará los 5 microservicios, la base de datos MariaDB y el Gateway Kong.

4. **Verificación:**

   - **API Gateway Status:** `http://localhost:8000`
   - **Kong Admin:** `http://localhost:8001`
   - **Documentación OpenAPI (Autogenerada):** `http://localhost:8000/api/v1/users/docs` (gracias al ruteo de Kong hacia FastAPI).