1. Arquitectura de Software

La arquitectura del sistema define la estructura fundamental del proyecto, los componentes principales y las interacciones entre ellos, garantizando la coherencia funcional entre los distintos módulos del sistema y los objetivos planteados en el diseño general.

1.1. Patrón Arquitectónico y Modelo de Comunicación

Se ha seleccionado una **Arquitectura Cliente–Servidor** con un **Backend Monolítico Centralizado**. Esta decisión responde a que todos los módulos (operador, supervisor, usuario y SIP) dependen de un conjunto común de datos (*incidencias, unidades, rutas y usuarios*), lo que facilita el mantenimiento, la consistencia de la información y el despliegue del sistema.

El modelo de comunicación implementado es de tipo **híbrido**, combinando dos mecanismos complementarios para optimizar la eficiencia del intercambio de información:

- **API REST:** Se utiliza para operaciones transaccionales y puntuales, tales como el registro de incidencias mediante `POST /api/incidencias` o la validación por parte del supervisor con `PUT /api/incidencias/:id`. Este enfoque sigue los principios HTTP estándar y garantiza interoperabilidad y claridad en la estructura de las peticiones.
- **WebSockets (Socket.IO):** Se emplean para la comunicación continua en tiempo casi real. El servidor emite notificaciones y actualizaciones automáticas hacia los clientes, por ejemplo: `socket.on('nueva_incidencia')` o `socket.on('actualizar_posiciones')`. Este canal bidireccional es esencial para el monitoreo en vivo de las unidades y la notificación inmediata de alertas en las interfaces del usuario y del supervisor.

1.2. Componentes del Sistema

La arquitectura está compuesta por cuatro elementos principales interconectados, los cuales interactúan a través del backend centralizado:

1. **Backend Central (Node.js + Express + Socket.IO):** Es el núcleo del sistema y concentra la lógica de negocio. Gestiona las peticiones REST, controla las conexiones WebSocket, valida los datos y mantiene la integridad de las operaciones. Además, ejecuta la simulación del recorrido de las unidades, actualizando sus coordenadas en intervalos definidos y emitiendo eventos hacia los clientes conectados.
2. **Clientes Móviles (Kotlin / Android):** Se desarrollan dos aplicaciones nativas:
   - **Botonera de Incidencias (Operador):** permite reportar incidencias y enviar su ubicación al servidor mediante API REST y WebSocket.
   - **Aplicación del Usuario Pasajero:** visualiza en un mapa las unidades en movimiento y recibe actualizaciones e incidencias validadas en tiempo real.
3. **Clientes Web (HTML / JavaScript):** Se incluyen dos interfaces accesibles desde navegador:
   - **Panel del Supervisor:** muestra y valida incidencias, además de visualizar el estado general del sistema y las posiciones de las unidades.
   - **Sistema de Información al Pasajero (SIP):** despliega información visual y auditiva dentro de las unidades (o en simulación) sobre la estación actual, próximas estaciones y alertas activas.

4. **Base de Datos (PostgreSQL):** Gestiona la persistencia de la información del sistema, garantizando integridad referencial, consistencia de datos y soporte para consultas complejas. Su diseño relacional facilita la trazabilidad entre incidencias, operadores, unidades y supervisores.

1.3. Arquitectura de Datos

La base de datos emplea el gestor **PostgreSQL** bajo un modelo relacional. El esquema, definido en el archivo `ScriptCreacionTablas.sql`, mantiene integridad referencial entre las tablas principales:

- Cada **Incidencia** está asociada a un **Operador** existente en la tabla `usuarios`.
- Los **Supervisores** validan las incidencias a través de claves foráneas que reflejan su participación.
- Las **Unidades** se vinculan a rutas simuladas que determinan su posición y recorrido.

Esta estructura relacional garantiza coherencia entre los distintos componentes del sistema y permite realizar consultas combinadas para los módulos de monitoreo, validación y visualización.

1.4. Justificación de la Arquitectura

- **Monolito sobre Microservicios:** Se opta por una arquitectura monolítica debido a la estrecha dependencia entre los módulos y la necesidad de compartir datos de manera inmediata. Dado que se trata de un entorno académico y de alcance limitado, un enfoque distribuido en microservicios añadiría complejidad innecesaria en la fase actual del proyecto.
- **SQL sobre NoSQL:** Se selecciona PostgreSQL (SQL) por su robustez, su soporte a integridad referencial y la capacidad de realizar consultas relacionales complejas. Un sistema NoSQL, como Firebase o MongoDB, no aportaría ventajas significativas dado que los datos del sistema son altamente estructurados y dependen de relaciones entre entidades.
- **Modelo Híbrido REST + WebSocket:** La combinación de ambos modelos ofrece lo mejor de cada paradigma:
  - REST asegura operaciones estandarizadas y seguras en las transacciones.
  - WebSocket garantiza inmediatez y actualización continua sin sobrecargar al servidor con peticiones recurrentes.

  Esto permite mantener una comunicación eficiente entre operador, supervisor, usuario y SIP, optimizando el rendimiento general del sistema.

En conjunto, la arquitectura adoptada equilibra la simplicidad del desarrollo con la capacidad de respuesta en tiempo cercano a real, ofreciendo una base sólida para futuras extensiones o despliegues del sistema Mexibús.