Proyecto 2 – CRM/IoT

Integración de aplicaciones

Marina Amaya, Ignacio Fernández y Britney Cuichan

Índice:

1. Contexto y objetivo

El objetivo del Tema 2 ha sido llevar todo lo que definimos teóricamente en el Tema 1 a una implementación real mediante dos microservicios independientes: un CRM simulado y un IoT simulado.

- En el CRM hemos gestionado la parte administrativa de clientes y pedidos.
- En el IoT hemos manejado sensores y lecturas operativas.

Nuestro propósito ha sido exponer datos válidos a través de APIs REST y validarlos usando los JSON Schema definidos previamente. De esta forma nos aseguramos de que toda la información (clientes, pedidos, sensores y lecturas) sea coherente, correcta y predecible.

2. Implementación del CRM

Tecnología: Node.js (v18+) con Express y AJV para validación de JSON Schema.

Endpoints implementados

- GET /clientes: devuelve listado de clientes con filtros opcionales:
    - ?q=: búsqueda por nombre o email (no sensible a mayúsculas).
    - ?ubicacionId=: filtra por ubicación.
    - ?page= y ?pageSize=: paginación.
- GET /clientes/:id: devuelve cliente por id.
- GET /pedidos: listado de pedidos con filtros opcionales clienteId y estado.
- GET /pedidos/:id: devuelve pedido por id.

Validación y errores

- Cada respuesta valida contra cliente.schema.json o pedido.schema.json.
- Códigos de estado implementados:
    - 200 éxito.
    - 400 parámetros inválidos.
    - 404 recurso no encontrado.
    - 500 datos no válidos según schema.

Datos semilla

- clientes.json: 50 clientes con distintos tipoForfait y saldoMonedero.
- pedidos.json: 10 pedidos para distintos clientes con estados variados (Completado, Pendiente de Pago, Cancelado).

Consideraciones

- Normalización de texto para búsqueda.
- Paginación segura con límites razonables.
- Respuestas deterministas desde ficheros JSON estáticos.

3. Implementación del IoT

Tecnología: Python 3.11 con FastAPI y jsonschema para validación de lecturas.

Endpoints implementados

- GET /sensores: devuelve listado de sensores, cada uno con:
    - id, tipo (aforo, temperatura, vibración), ubicacionId, estadoOperacional.
- GET /lecturas: listado de lecturas con filtros opcionales:
    - sensorId, ubicacionId.
    - from, to (ISO-8601).
    - limit (máximo 1000 por defecto).

Validación y errores

- Cada lectura valida contra lectura.schema.json.
- Códigos de estado:
    - 200 éxito.
    - 400 parámetros inválidos (fechas o límites).
    - 500 datos no válidos según schema.

Datos semilla

- sensores.json: 3 sensores distintos:
    - REM-TS01 (aforo)
    - REM-TC02 (vibración)
    - REM-TS01-TEMP (temperatura)
- lecturas.json: ≥5 lecturas por sensor, con valor, unidad y timestamp ISO-8601.

Consideraciones

- Filtros aplicados en orden: sensorId → ubicacionId → from → to.
- Determinismo asegurado mediante datos estáticos.
- Lógica de filtro de fechas aplicadab
- Todas las fechas convertidas a UTC (Z).

4. Integración y pruebas

- Se probaron todos los endpoints con Postman, exportando la colección a /docs/postman_collection.json.
- Se verificó que todas las respuestas cumplen sus respectivos JSON Schema.
- La paginación y los filtros funcionan correctamente para CRM e IoT.
- Todos los datos son deterministas y reproducibles.

5. Conclusión

Hemos desarrollado una primera versión completa de los dos microservicios del proyecto, cumpliendo todo lo que se pedía en el Tema 2. El microservicio CRM gestiona clientes y pedidos con búsqueda, filtros y paginación. El servicio IoT ofrece sensores y lecturas con filtros y validación de fechas en formato ISO-8601. Además, ambos microservicios validan sus respuestas usando JSON Schema para asegurarnos de que los datos estén siempre bien formados. Con esta base ya implementada, quedamos preparados para el siguiente Tema, donde integraremos ambas APIs en un único endpoint unificado.