Universidad De Las
Americas

Examen Práctico

Progreso 1 – P2

INTEGRACION DE SISTEMAS

Ing. en Software

Oscar Rodriguez

Contenido

EXAMEN PRACTICO

CASO EMPRESARIAL: AGROTECH SOLUTIONS S.A.

AgroTech Solutions S.A. es una empresa de tecnología agrícola que desarrolla soluciones para optimizar el uso del agua y los recursos agrícolas mediante sensores de humedad y temperatura distribuidos en el campo. Actualmente, la empresa tiene tres sistemas independientes:

1. **SensData:** recopila lecturas de sensores de humedad y temperatura, exportando datos en archivos CSV cada hora.
2. **AgroAnalyzer:** procesa los datos para calcular promedios diarios y generar reportes.
3. **FieldControl:** controla los sistemas de riego y requiere consultar los últimos valores de los sensores críticos para activar o detener las bombas.

El intercambio de información entre estos sistemas es manual y poco confiable. La gerencia de TI te ha solicitado desarrollar un prototipo de integración completo, aplicando patrones clásicos de integración empresarial.

Objetivo

Diseñar e implementar una solución de integración funcional que permita conectar los tres sistemas de AgroTech aplicando los siguientes patrones:

1. Transferencia de Archivos (File Transfer) para mover datos desde SensData hacia AgroAnalyzer.
2. Base de Datos Compartida (Shared Database) como repositorio común entre AgroAnalyzer y FieldControl.
3. Remote Procedure Call (RPC simulado) para permitir la consulta directa desde FieldControl hacia AgroAnalyzer.

Requerimientos técnicos

Stack recomendado (línea base):

- Lenguaje: Java 17 o superior
- Gestor de dependencias: Apache Maven
- Framework: Apache Camel 4.x
- IDE recomendado: Visual Studio Code o el de su preferencia
- Base de datos: H2 o SQLite.

Escenario técnico por desarrollar

1. Instalación de herramientas

- Java JDK

Link: https://www.oracle.com/java/technologies/downloads/

- Apache maven

  Link: https://maven.apache.org/download.cgi

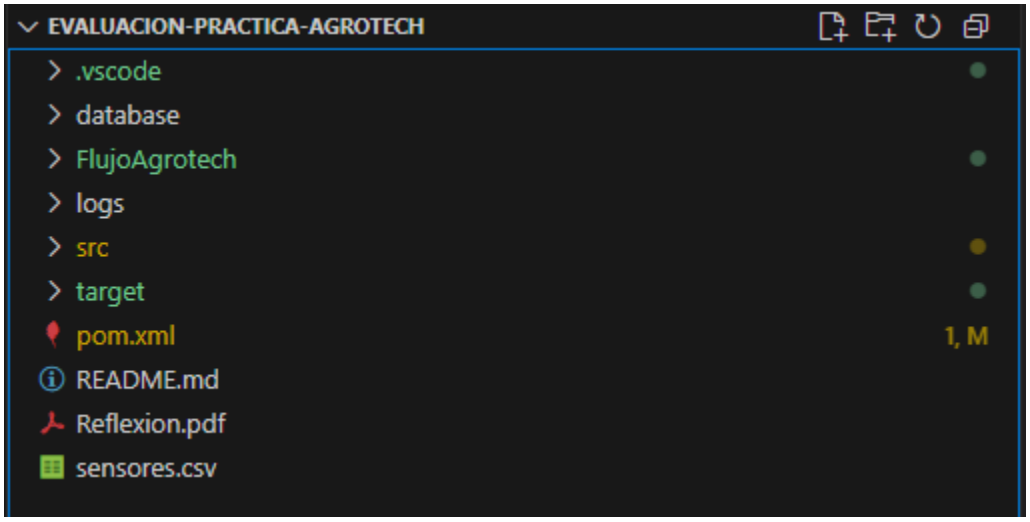
- IDE: Visual Studio Code

  Link: https://code.visualstudio.com/download

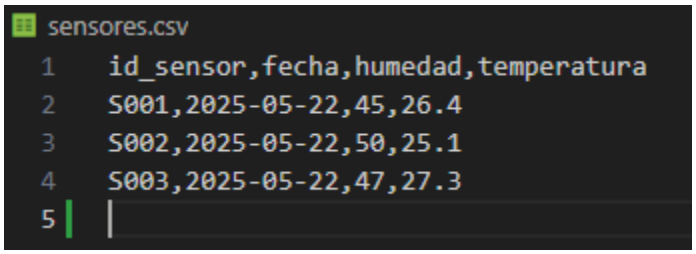
2. Crear estructura de carpetas del flujo.

Formato de entrega:

```
/evaluacion-practica-agrotech/
├── sensores.csv
├── src/
├── database/
├── FlujoAgrotech/
├── logs/
├── README.md
└── Reflexion.pdf (Documento con todas las evidencias de funcionamiento
    del proyecto y respuestas a las preguntas de reflexión)
```



3. Crear un archivo “sensores” CSV de ejemplo.



4. Código Fuente

Clase App.java:

```
public class App extends RouteBuilder
{
    public static void main( String[] args ) throws Exception···

    @Override
    public void configure() {···

    // Crea la base de datos y la tabla si no existe
    private void crearTablaSiNoExiste() {···

    // Intenta convertir un String a double, devuelve defaultVal si falla
    private static double tryParseDouble(String s, double defaultVal) {···
}
```

Librerías Usadas:

```
package agrotech;

import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.main.Main;
import org.apache.camel.component.file.GenericFile;
import com.fasterxml.jackson.databind.ObjectMapper;
import java.io.File;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.Statement;
import java.util.*;
```

Método Main:

```
public static void main( String[] args ) throws Exception
{
    Main main = new Main();
    main.configure().addRoutesBuilder(new App());
    main.run(args);
}
```

Método crearTablaSiNoExiste:

```
// Crea la base de datos y la tabla si no existe
private void crearTablaSiNoExiste() {
    try (Connection conn = DriverManager.getConnection(url: "jdbc:sqlite:database/AgroAnalyzer.sqlite")) {
        File dir = new File(pathname: "database");
        if (!dir.exists()) dir.mkdirs();

        Statement stmt = conn.createStatement();
        stmt.execute("""
            CREATE TABLE IF NOT EXISTS lecturas (
                id_sensor VARCHAR(10) NOT NULL,
                fecha TEXT NOT NULL,
                humedad DOUBLE,
                temperatura DOUBLE
            );
        """);
        stmt.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Método tryParseDouble:

```java
// Intenta convertir un String a double, devuelve defaultVal si falla
private static double tryParseDouble(String s, double defaultVal) {
    try {
        return Double.parseDouble(s);
    } catch (Exception e) {
        return defaultVal;
    }
}
```

Clase App.java:

```java
package agrotech;

import org.apache.camel.Header;

public class ServicioAnalitica {
    public String getUltimoValor(@Header("id_sensor") String id) {
    return """
            {\"id\":\"%s\",\"humedad\":48,\"temperatura\":26.7,\"fecha\":\"2025-05-22T10:30:00Z\"}
            """.formatted(id);
    }
}
```

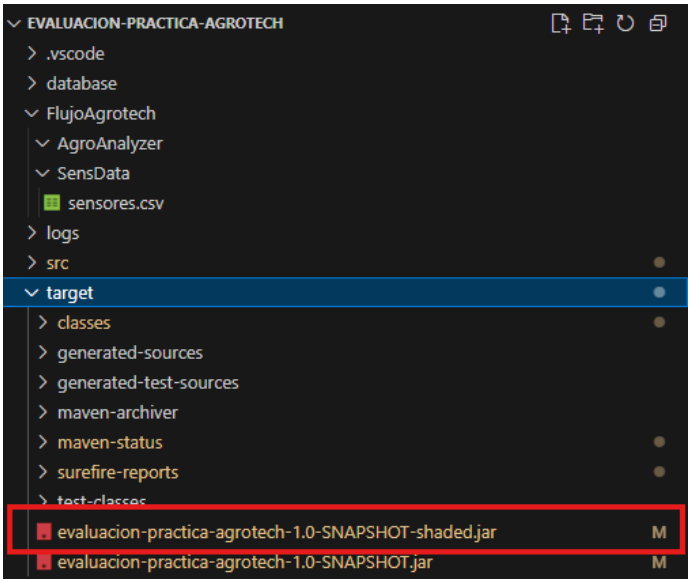
5. Compilar el Proyecto

Comando: *mvn clean package*

Resultado de la compilación:

```
[INFO] Attaching shaded artifact.
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  8.528 s
```

Se generará el archivo ejecutable:



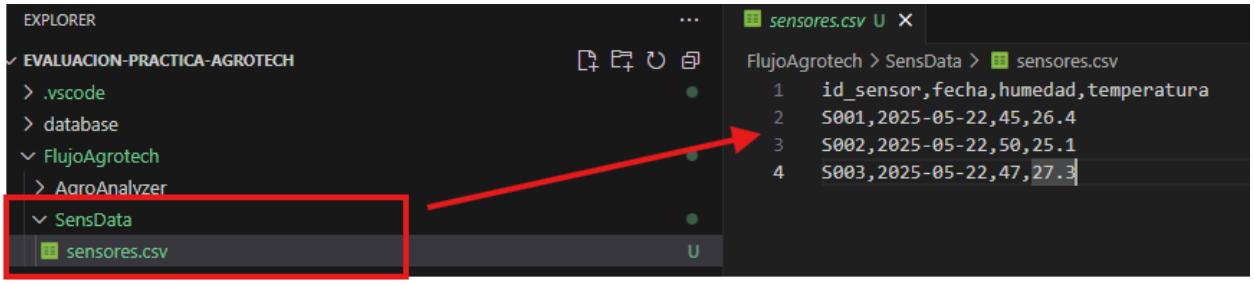
Se ejecuta el flujo Camel

Comando: *java -jar target/evaluacion-practica-agrotech-1.0-SNAPSHOT-shaded.jar*

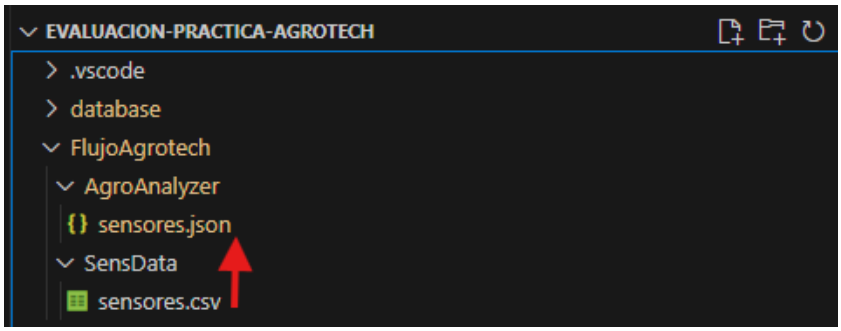
6. Log de ejecución

7. Transferencia de archivos (SensData → AgroAnalyzer).

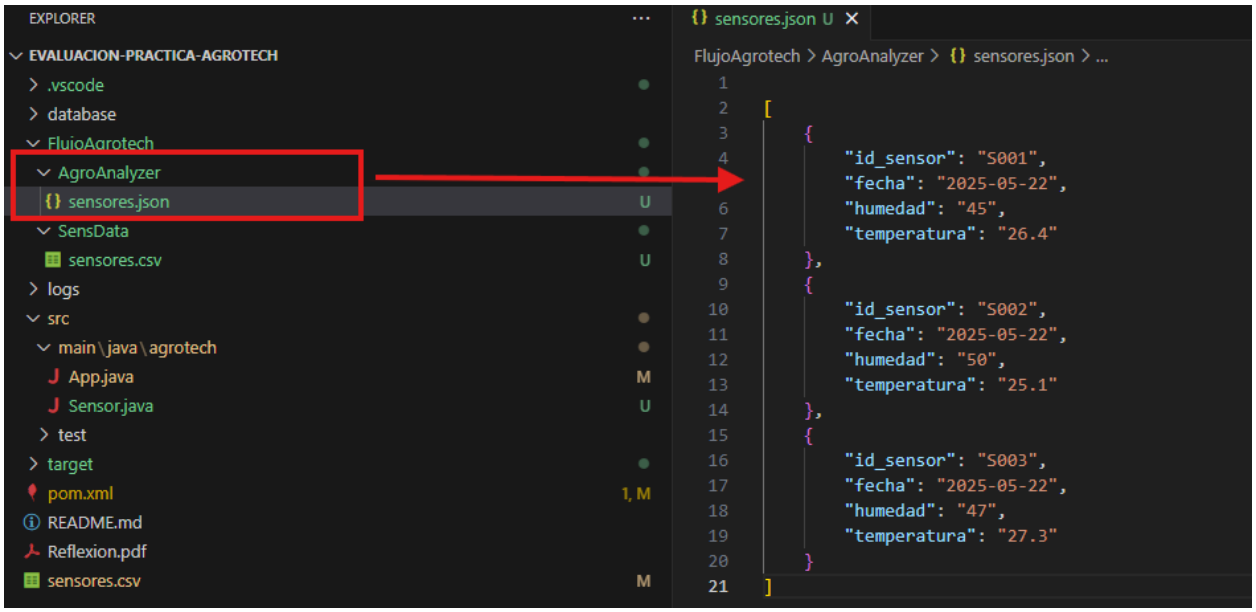
- Generar un archivo sensores.csv con lecturas.



- La ruta debe leer este archivo y transferirlo automáticamente (moverlo o copiarlo a otra carpeta).

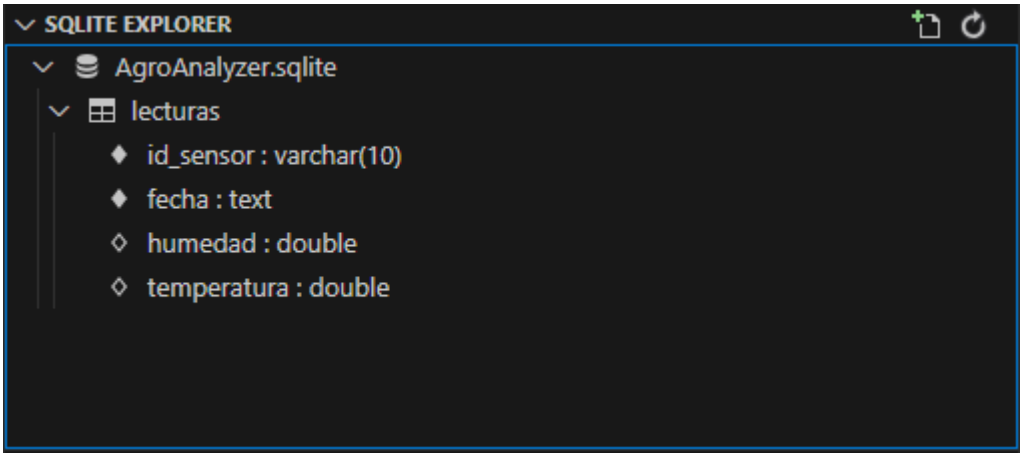


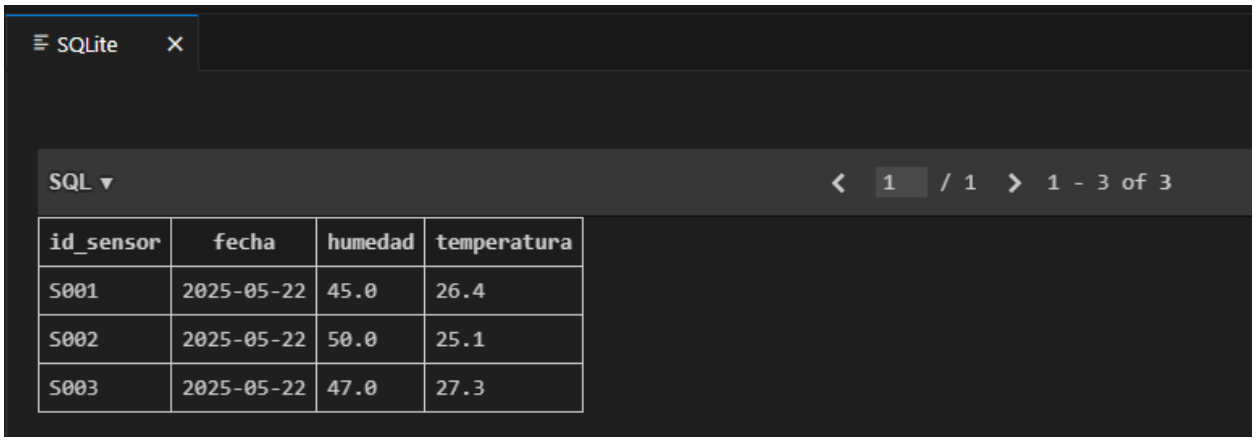
- Convertir los datos a JSON y enviarlos al módulo AgroAnalyzer.



8. Base de datos compartida (AgroAnalyzer ↔ FieldControl).

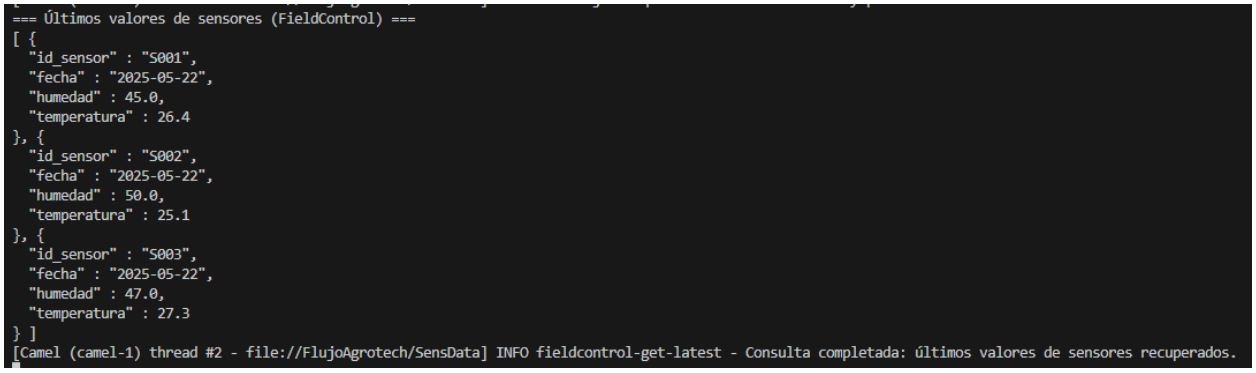
- AgroAnalyzer debe guardar los datos procesados en una base de datos SQLite.





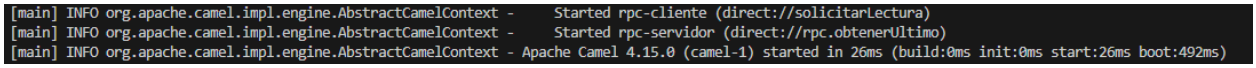
id_sensor	fecha	humedad	temperatura
S001	2025-05-22	45.0	26.4
S002	2025-05-22	50.0	25.1
S003	2025-05-22	47.0	27.3

- FieldControl debe leer directamente la base de datos para obtener los valores más recientes de cada sensor.



9. Remote Procedure Call (RPC simulado con Apache Camel).

Simula una llamada remota síncrona entre FieldControl y AgroAnalyzer.



10. Reflexión individual.

a. ¿Qué patrón aplicaste en cada fase del flujo y por qué?

En la primera fase apliqué el patrón File Transfer, porque era la forma más sencilla de mover los archivos CSV generados por SensData hacia AgroAnalyzer. En la segunda fase utilicé el patrón Shared Database, ya que ambos sistemas necesitaban acceder a la misma información de lecturas sin depender de archivos intermedios. Finalmente, en la tercera fase implementé un RPC simulado, para permitir una comunicación directa y síncrona entre FieldControl y AgroAnalyzer, simulando una consulta en tiempo real.

b. ¿Qué riesgos observas al usar una base de datos compartida?

El principal riesgo es que varios sistemas pueden modificar los datos al mismo tiempo, causando inconsistencias. También puede haber problemas de rendimiento si muchos procesos acceden a la base simultáneamente, y se pierde el aislamiento entre sistemas, lo que dificulta el mantenimiento y la escalabilidad.

c. ¿Cómo ayuda el RPC simulado a representar un flujo síncrono?

El RPC simulado permite que un sistema haga una solicitud y espere la respuesta antes de continuar, reproduciendo el comportamiento de una llamada remota real. Esto muestra cómo se pueden coordinar dos módulos en tiempo real sin depender de archivos o bases de datos intermedias.

d. ¿Qué limitaciones tienen los patrones clásicos frente a arquitecturas modernas?

Los patrones clásicos son efectivos para integrar sistemas pequeños, pero no escalan bien cuando hay muchos servicios o grandes volúmenes de datos. Además, suelen generar dependencias fuertes y poca flexibilidad. En cambio, las arquitecturas modernas como microservicios o event-driven ofrecen más independencia, escalabilidad y mejor manejo de concurrencia y errores.

Código fuente:

URL: https://github.com/orodriguez-dev/evaluacion-practica-agrotech.git

Conclusión

El desarrollo del prototipo de integración para AgroTech Solutions permitió aplicar de forma práctica los patrones clásicos de integración empresarial: Transferencia de Archivos, Base de Datos Compartida y RPC simulado. Gracias a esto, los tres sistemas —SensData, AgroAnalyzer y FieldControl— lograron comunicarse de manera automatizada y confiable. Este ejercicio ayudó a comprender cómo los flujos síncronos y asincrónicos pueden coordinar la transmisión y el análisis de datos dentro de una arquitectura distribuida.

Recomendación

Se recomienda en el futuro migrar hacia una arquitectura basada en servicios (SOA o API REST) para mejorar la escalabilidad y el mantenimiento del sistema. Además, sería ideal implementar validaciones y manejo de errores más robustos dentro de los flujos de integración, garantizando mayor resiliencia ante fallas en la comunicación o datos incompletos.