Universidad de Las Américas

Ingeniería de Software

Integración de Sistemas

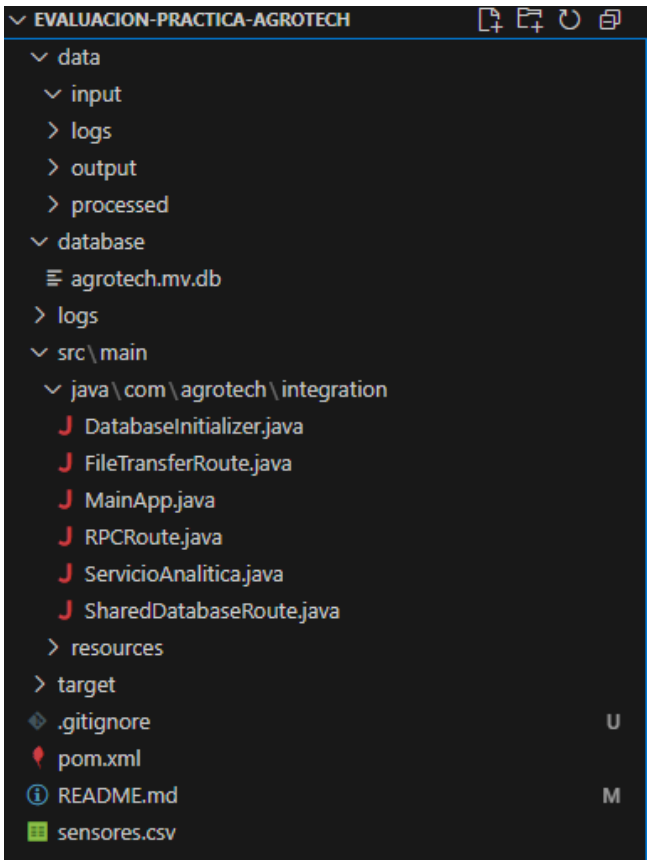
Nombre: Daniel A. Vizcarra

Fecha: 30/10/2025

Evaluación Práctica – Sistema de Integración Agrotech Solutions

1. EVIDENCIAS DE FUNCIONAMIENTO

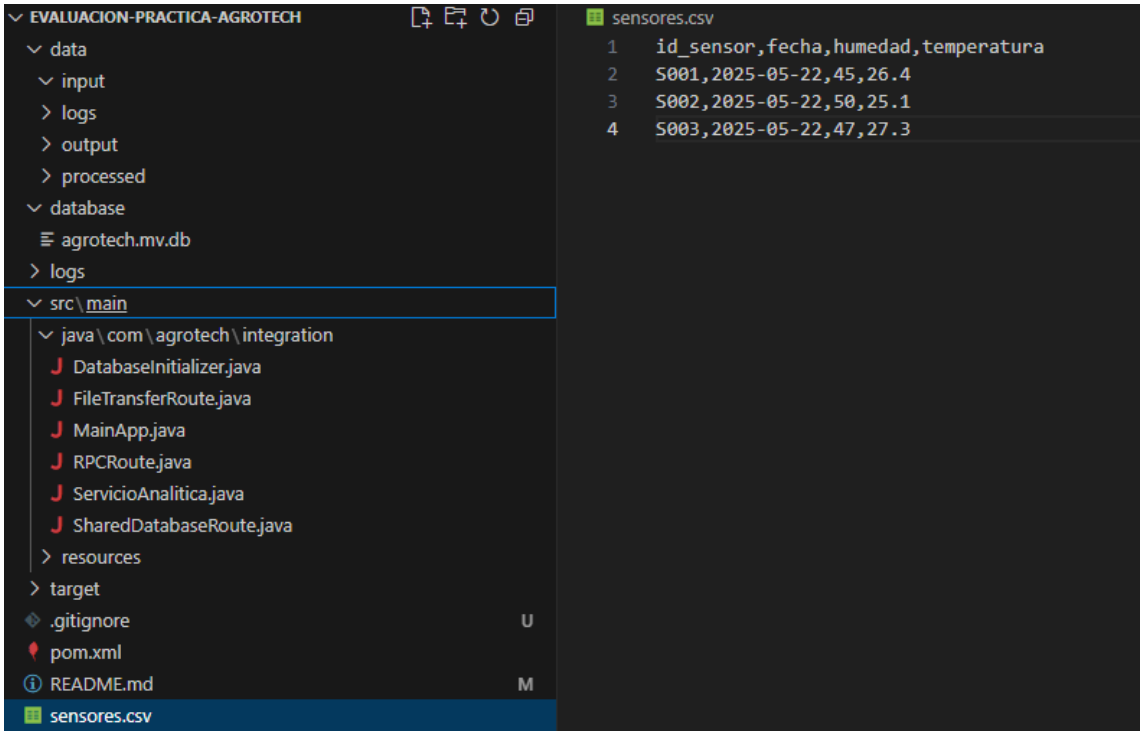
1.1. Estructura del Proyecto



Estructura completa del proyecto mostrando:

- Carpetas data/ (input, output, processed, logs)
- Carpeta database/
- Carpeta src/ con todas las clases Java
- Archivos pom.xml, README.md, .gitignore, sensores.csv

1.2. Archivo CSV de Entrada



Contenido del archivo sensores.csv ubicado en la raíz:

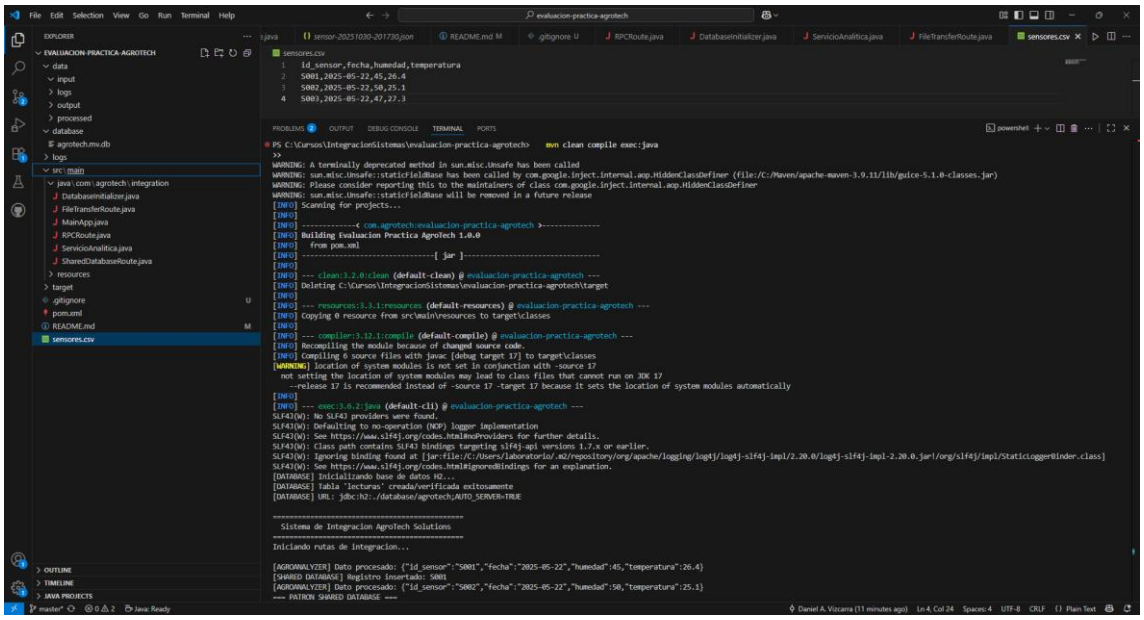
id_sensor,fecha,humedad,temperatura

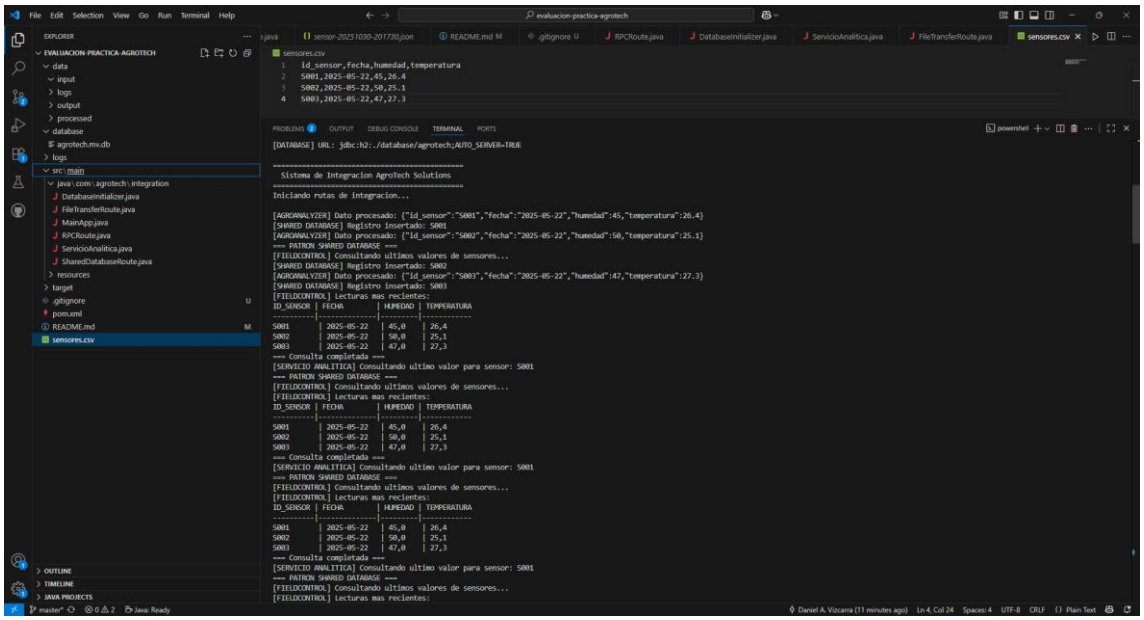
S001,2025-05-22,45,26.4

S002,2025-05-22,50,25.1

S003,2025-05-22,47,27.3

1.3. Ejecución Completa del Sistema





- Inicialización de la base de datos H2
- Procesamiento del archivo CSV
- Conversión a JSON
- Inserción en base de datos
- Consultas periódicas (Shared Database)
- Llamadas RPC entre cliente y servidor

1.4. Patrón 1: File Transfer

Logs:

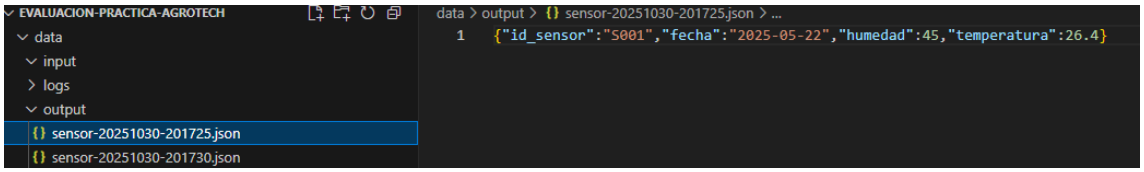
```
==========================================
  Sistema de Integracion AgroTech Solutions
==========================================
Iniciando rutas de integracion...

[AGROANALYZER] Dato procesado: {"id_sensor":"S001","fecha":"2025-05-22","humedad":45,"temperatura":26.4}
[SHARED DATABASE] Registro insertado: S001
[AGROANALYZER] Dato procesado: {"id_sensor":"S002","fecha":"2025-05-22","humedad":50,"temperatura":25.1}
=== PATRON SHARED DATABASE ===
[FIELDCONTROL] Consultando ultimos valores de sensores...
[SHARED DATABASE] Registro insertado: S002
[AGROANALYZER] Dato procesado: {"id_sensor":"S003","fecha":"2025-05-22","humedad":47,"temperatura":27.3}
[SHARED DATABASE] Registro insertado: S003
```

Archivo CSV original:

sensores.csv

```
1  id_sensor,fecha,humedad,temperatura
2  S001,2025-05-22,45,26.4
3  S002,2025-05-22,50,25.1
4  S003,2025-05-22,47,27.3
```

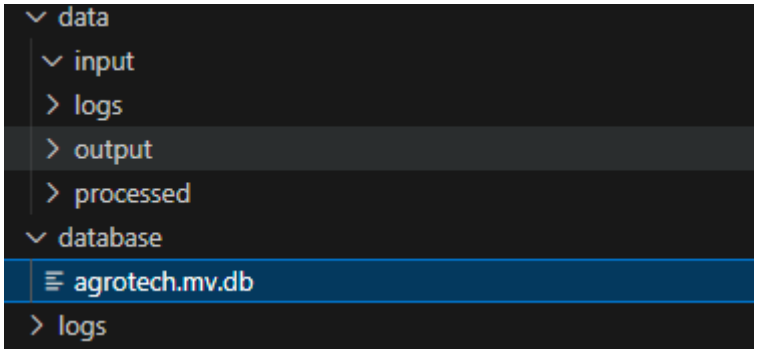
Archivos JSON generados (data/output/):



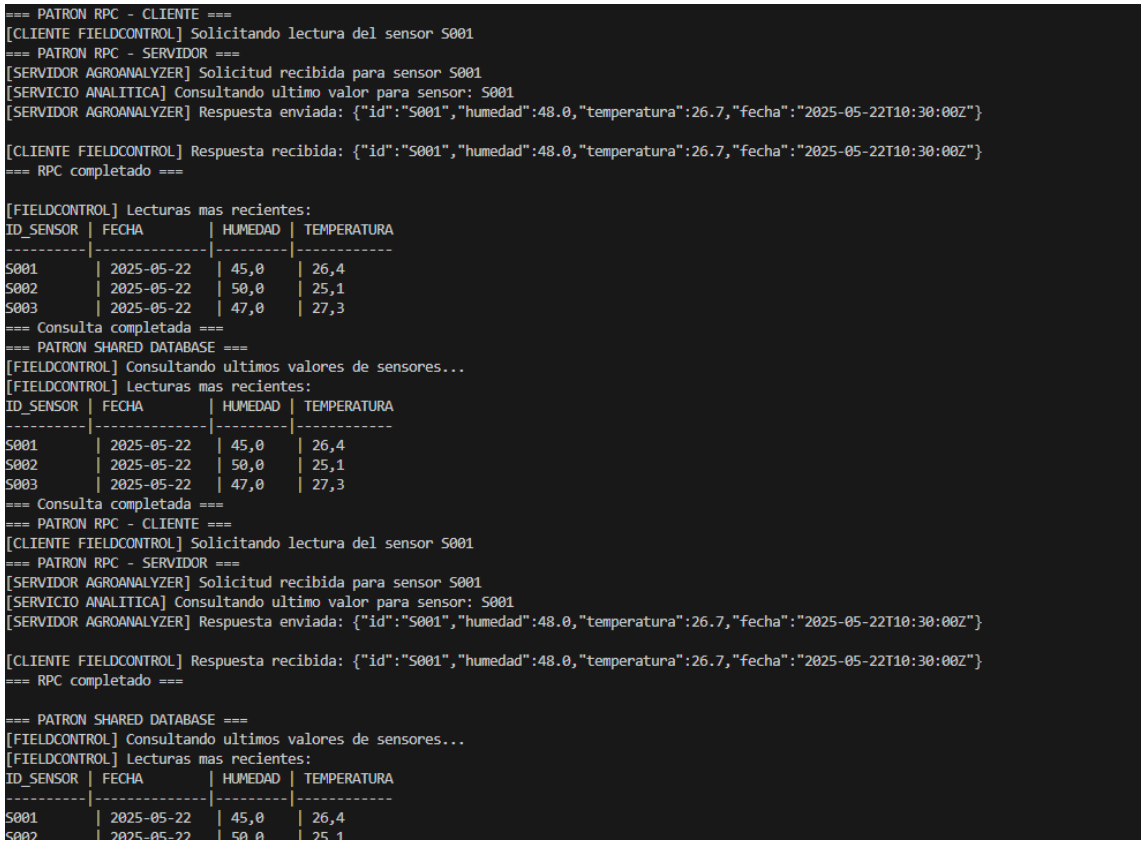
1.5. Patrón 2: Shared Database

```
=== PATRON SHARED DATABASE ===
[FIELDCONTROL] Consultando ultimos valores de sensores...
[SHARED DATABASE] Registro insertado: S002
[AGROANALYZER] Dato procesado: {"id_sensor":"S003","fecha":"2025-05-22","humedad":47,"temperatura":27.3}
[SHARED DATABASE] Registro insertado: S003
[FIELDCONTROL] Lecturas mas recientes:
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
=== Consulta completada ===
```

Base de datos creada:



1.6. Patrón 3: RPC Simulado



El RPC hace llamadas cada 20 segundos.

2. REFLEXIÓN INDIVIDUAL

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

FASE 1: SensData a AgroAnalyzer - FILE TRANSFER

Apliqué File Transfer porque SensData exporta datos cada hora en CSV, lo que indica procesamiento batch sin urgencia. Las lecturas de sensores agrícolas no requieren respuesta inmediata, y este patrón proporciona desacoplamiento total entre sistemas. Ambos pueden estar en diferentes tecnologías y ejecutarse en momentos distintos sin afectarse mutuamente.La implementación monitorea la carpeta input cada 5 segundos, lee el CSV, lo convierte a JSON, inserta en la base de datos y elimina el archivo procesado. Es ideal porque permite acumular múltiples lecturas en un archivo, es fácil de auditar, tolera desconexiones temporales y procesa eficientemente grandes volúmenes.

FASE 2: AgroAnalyzer y FieldControl - SHARED DATABASE

Elegí Shared Database porque FieldControl necesita consultar datos históricos y tendencias para decisiones de riego, no solo valores actuales. La base de datos permite consultas complejas como promedios de 24 horas o filtros por temperatura que serían imposibles con archivos.Además, múltiples sistemas pueden leer y escribir simultáneamente. AgroAnalyzer inserta datos procesados mientras FieldControl consulta cada 15 segundos. La base de datos

garantiza consistencia mediante transacciones ACID y actúa como fuente única de verdad para todos los sistemas.

FASE 3: FieldControl a AgroAnalyzer – RPC

Implementé RPC porque FieldControl controla bombas de riego que requieren decisiones inmediatas. Cuando consulta el último valor de un sensor crítico, necesita la respuesta ahora para activar o detener el riego. Un retraso podría desperdiciar agua o dañar cultivos.La comunicación síncrona es apropiada aquí porque el flujo no tiene sentido sin la respuesta. La operación es simple: obtener el último valor de un sensor específico. No requiere procesamiento complejo ni justifica la complejidad de un message broker. RPC con timeout de 2 segundos proporciona respuesta inmediata y maneja errores directamente.

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

Acoplamiento de esquema

Todos los sistemas dependen de la misma estructura de tabla. Agregar una columna como ubicacion_gps requiere coordinar cambios en AgroAnalyzer y FieldControl simultáneamente. Esto dificulta evolución independiente y hace cambios más lentos. Un error en el esquema afecta a todos los sistemas.

Bloqueos y contención

Cuando AgroAnalyzer inserta 1000 registros y FieldControl intenta leer para decisión urgente de riego, FieldControl queda bloqueado esperando. Esto causa retrasos críticos. También hay riesgo de deadlocks cuando sistemas acceden a datos en orden diferente.

Escalabilidad limitada

La base de datos única es cuello de botella. Con 10000 sensores generando millones de lecturas diarias, H2 no soportaría la carga. No se puede escalar horizontalmente porque todos comparten la misma instancia.

Single point of failure

Si la base de datos falla, todos los sistemas fallan. SensData no puede guardar lecturas, AgroAnalyzer no procesa datos, FieldControl no controla riego. Un simple mantenimiento requiere detener toda la operación.

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

Bloqueo del cliente

Cuando fieldcontrol ejecuta to direct rpc obtenerultimo, se detiene completamente y espera respuesta. La ejecución no continúa hasta que agroanalyzer procese y retorne resultado. En los logs se ve siempre: solicitud cliente, procesamiento servidor, respuesta cliente, en orden estricto.

Acoplamiento temporal

Cliente y servidor deben estar activos simultáneamente. Si agroanalyzer está caído, fieldcontrol obtiene error inmediato. No hay cola ni almacenamiento temporal. La comunicación es directa como en rpc tradicional.

Respuesta garantizada

El cliente siempre obtiene respuesta o excepción. No existe mensaje enviado sin confirmación. El timeout de 2000 milisegundos asegura que si no hay respuesta en 2 segundos, el cliente detecta el problema.

Orden predecible

Las operaciones ocurren en secuencia estricta: log solicitud, llamada rpc que bloquea, log respuesta. No hay paralelismo. Esta predictibilidad es característica de flujos síncronos.

Comunicación directa

El endpoint direct simula conexión en memoria sin brokers ni colas. Replica naturaleza punto a punto de rpc donde cliente conoce exactamente a qué servidor llama.

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

Limitaciones File Transfer

Alta latencia por polling cada 5 segundos. En producción podría ser cada hora, dejando datos obsoletos. Arquitecturas modernas usan Kafka donde eventos se publican inmediatamente con latencia de milisegundos.

No escala con grandes volúmenes. Con 10000 sensores, los CSV serían enormes y lentos. Soluciones modernas dividen eventos en particiones que múltiples consumidores procesan en paralelo.

Sin garantías de entrega. Archivo perdido significa datos perdidos permanentemente. Plataformas modernas garantizan entrega con acknowledgment y reintentos automáticos.

Monitoreo limitado. Difícil rastrear archivos o detectar fallos silenciosos. Arquitecturas modernas usan distributed tracing con Jaeger para seguir cada evento.

Limitaciones Shared Database

- Acoplamiento de esquema. Cambiar tabla requiere coordinar todos los sistemas. En microservicios, cada servicio tiene su base de datos y evoluciona independientemente mediante APIs versionadas.
- No escala horizontalmente. Una sola base de datos es cuello de botella. Arquitecturas modernas usan CQRS con event store distribuido y múltiples réplicas de lectura.
- Bloqueos empeoran con concurrencia. Soluciones modernas evitan compartir estado mutable, prefiriendo eventos inmutables.
- Single point of failure. Arquitecturas modernas tienen múltiples instancias distribuidas geográficamente con replicación automática.
- Seguridad granular difícil. Arquitecturas modernas usan API Gateways con OAuth2 y políticas detalladas por endpoint.

Limitaciones RPC

- Acoplamiento fuerte. Cambios en servidor rompen clientes. APIs REST modernas usan versionado en URL para compatibilidad hacia atrás.

- Sin tolerancia a fallos. Servidor caído significa cliente fallado. Arquitecturas modernas usan circuit breakers que retornan respuestas desde cache permitiendo degradación graceful.
- Difícil de escalar. Arquitecturas modernas usan service mesh con load balancing automático y service discovery.
- Observabilidad pobre. Arquitecturas modernas con OpenTelemetry proporcionan trazas distribuidas mostrando exactamente dónde falla cada llamada.

Link del repositorio GIT:
https://github.com/daniel-vizcarra/evaluacion-practica-agrotech.git