INTEGRACIÓN ETL HL7v2.5 y conversión a FHIR mediante Mirth Connect + Java y Carga en FHIR Server Aidbox.

- **Descripción:**

Este proyecto simula un flujo de interoperabilidad clínica donde se reciben mensajes HL7v2 (en este caso de usi: ORU^R01), el cual será procesado inicialmente por Mirth Connect y luego transformado a un recurso FHIR (Observation/Patient) mediante una aplicación en Spring Boot con la librería HAPI FHIR.

Los recursos generados se envían a un servidor Aidbox FHIR para su almacenamiento y posterior consulta.

El objetivo es demostrar conocimientos en estándares eHealth, integración HL7 / FHIR, desarrollo backend en Java y uso de servidor FHIR Aidbox.

- **Stack utilizado:**

Componente	Herramienta
HL7v2	Mirth Connect
Transformación	Java + HAPI FHIR
Validación	HAPIFhirValidator
Cliente FHIR	HAPIIGenericClient
Servidor FHIR	Aidbox Community
IDE	IntelliJ con Maven

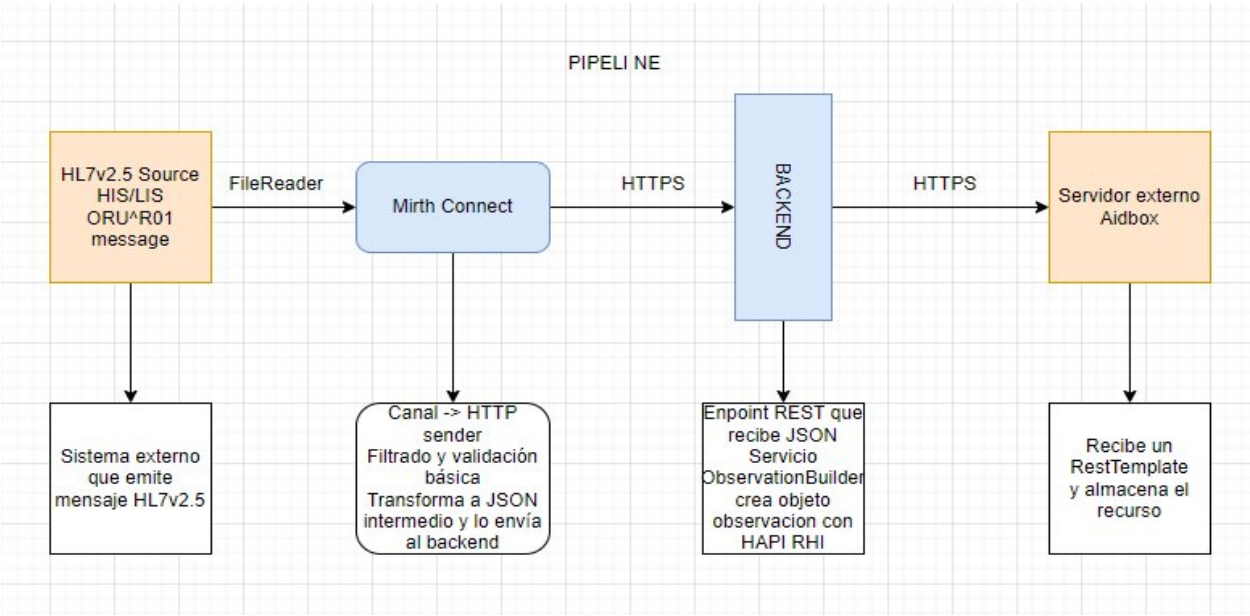


Figura 1 Diagrama de componentes y arquitectura a alto nivel del sistema

Explicación del Componentes:

1) Entrada HL7v2.5 en Mirth Connct

- El canal recibe un ORU^R01 por FIleReader
- Utiliza un JavaScript transformer para parsear el mensaje y construir un JSON
- Envía el Json por Web Service

2) Aplicación en Spring Boot con HAPI FHIR:

- Se recibe el JSON desde Mirth.
- Crear instancias de FHIR con HAPI FHIR.
- Validar el recurso generado antes de enviar a Aidbox
- Si no es válido, se logea el error.
- Si la validación es exitosa IGenericClient para POST hacía Aidbox.

3) Servidor Aidbox

- Se utiliza como repositorio para mostrar los recursos creados.

Ciclo ETL (Extract, Transform, Load):

- **Extract (E):**Mirth Connect ingesta el archivo HL7v2.
- **Transform (T):**Mirth con JavaScript transformando a HL7v2 a JSON, y luego envío a Spring Boot transformando JSON a recursos FHIR y validándolos.
- **Load (L):**Tu Spring Boot enviando los recursos FHIR a Aidbox.

FLUJO DE INTEROPERABILIDAD COMPLETO

1) Origen (HL7v2 File en mi Desktop)

El sistema genera un mensaje HL7v2 de tipo ORU (en este casos un LIS).

2) Paso 1:

- Mirth Connect (ETL, - Extract y Transform inicial)
- Extract: El FileReader de Mirth recibe de origen el mensaje HL7v2.5.
- Initial Transform: El JavaScript Transformer de Mirth toma el mensaje HL7vx y lo convierte en un JSON plano, Mirth es excelente para parsear HL7vx a otros formatos.
- Load: El Web Service Sender de Mirth envía el JSON al backend de Spring Boot.

3) Paso 2: Spring Boot (ETL - Transformación secundaria, Validacion y Carga Final)

- Recibe: En MirthController se recibe el JSON de Mirth.

- Transformación secundaria: El FhirProcessorService toma ese JSON y lo mapea a recurso FHIR, utilizando las clases y utilidades de HAPI FHIR. Aquí es donde se aplica la lógica de negocio para construir objetos FHIR complejos.
- Validación: El FhirProcessorService utiliza FhirValidator de HAPI FHIR para asegurar que los recursos FHIR generados cumplan con el estándar (y, opcionalmente un recurso perfilado específicamente).
- Final: FhirProcessorService utiliza IGenericCliente de HAPI FHIR para enviar (POST) los recursos FHIR validados al servidor de FHIR, en este caso Aidbox.

4) Destino: Aidbox FHIR server.

Aidbox almacena los recursos FHIR haciendo que estén disponibles para consulta e intercambio.

¿ Por qué se dividen responsabilidades?

Esta división es una buena práctica en la integración de sistemas de salud.

- **Mirth Connect >** Su fortaleza es la orquestación y el mapeo de bajo nivel entre formatos de mensajería, maneja los protocolos de transprote (como MLLP) y realiza transformaciones de formato iniciales (en este caso de HL7vx a JSON).
- **Aplicación en Spring Boot (con HAPI FHIR) >** Recae la lógica de negocio y las validaciones más complejas así como el mapeo detallado al modelo específico de datos (FHIR). La validación otorga conformidad con estándares de interoperablidad y la interacción con APIs de alto nivel. Permite un desarrollo más robusto, escalable y testeable. Si se pretendiese hacer toda la transformación directa (HL7vx a FHIR) solo en Mirht, el JavaScript del transformador sería extremadamente complejo y dificil de mantener, testear y escalar. Si se intentara que Spring Boot leyera directamente el archivo HL7v2, perderías la capacidad de Mirth para manejar la ingesta de archivos y los protocolos de mensajería.

EL PROYECTO EN IMÁGENES

Figura 2 Evento HL7v2.5 de tipo ORU

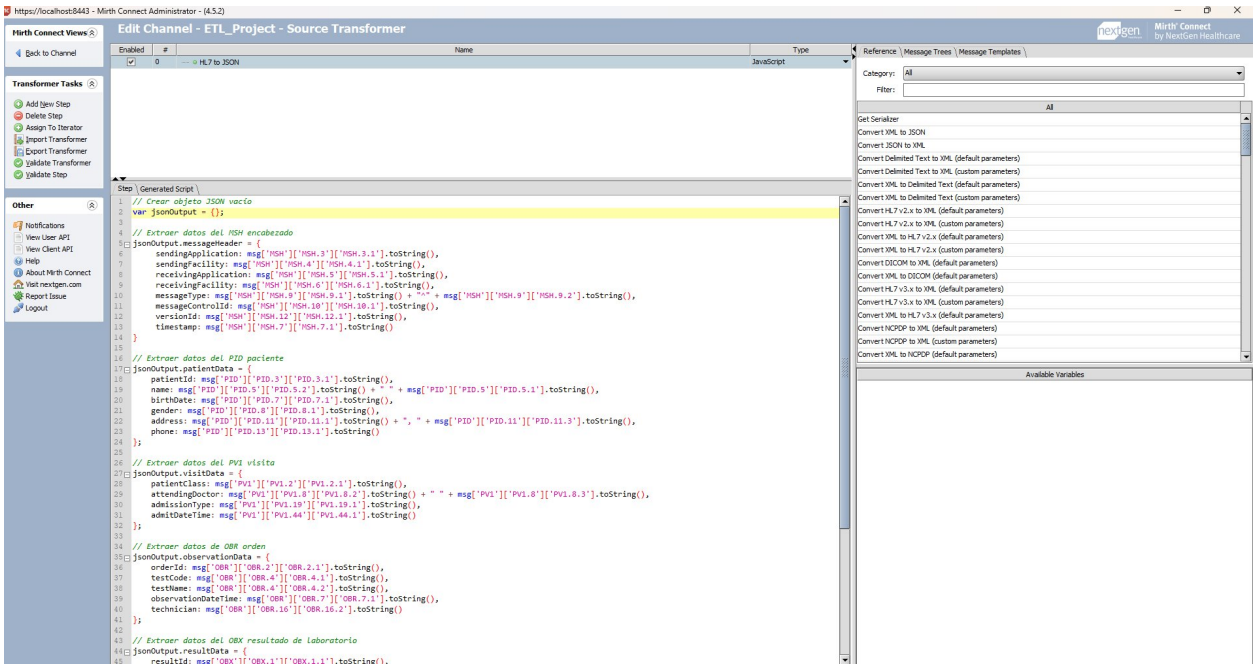
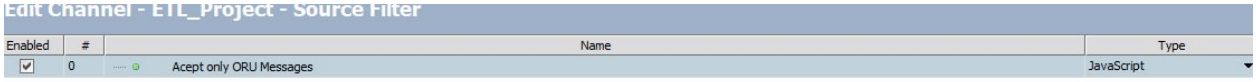


Figura 3 Transformador en JavaScript



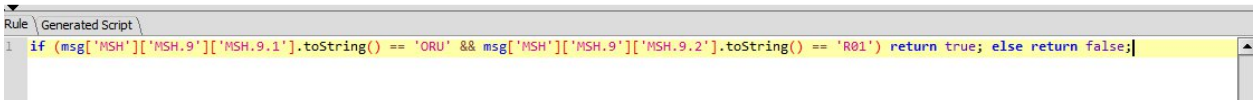


Figura 4Filtro para solo aceptar mensajes de tipo ORU

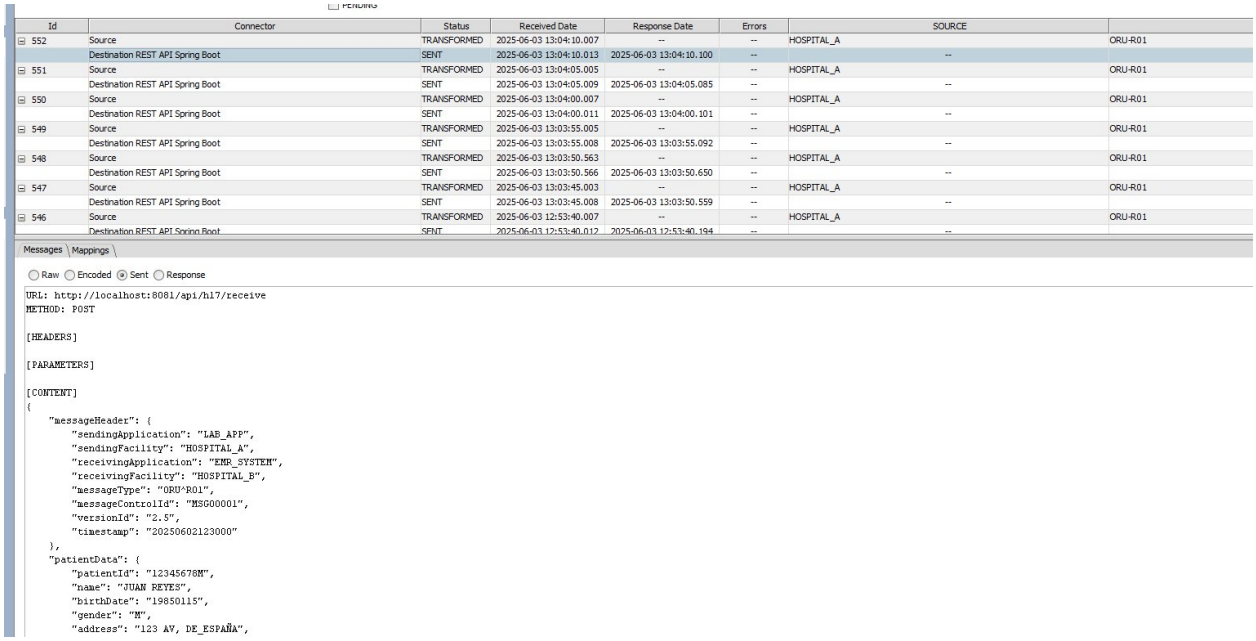


Figura 5Canal desplegado y JSON transformado y enviado

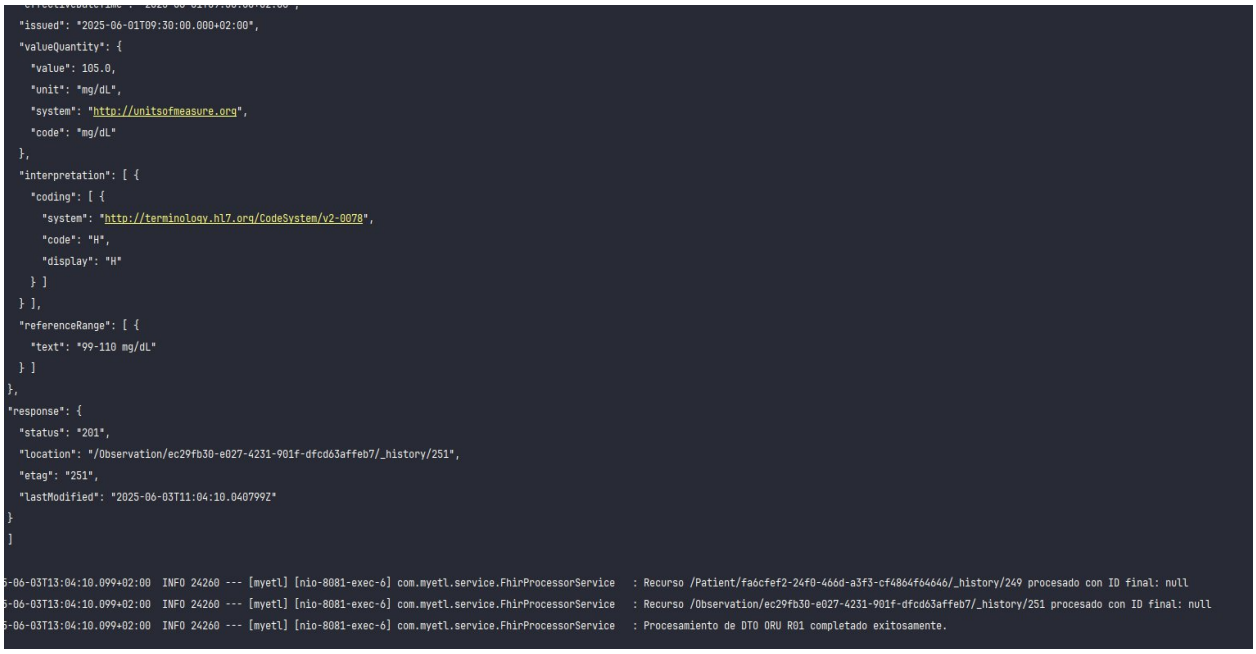


Figura 6Logger Spring muestra el JSON recibido y mapeado/validado a FHIR

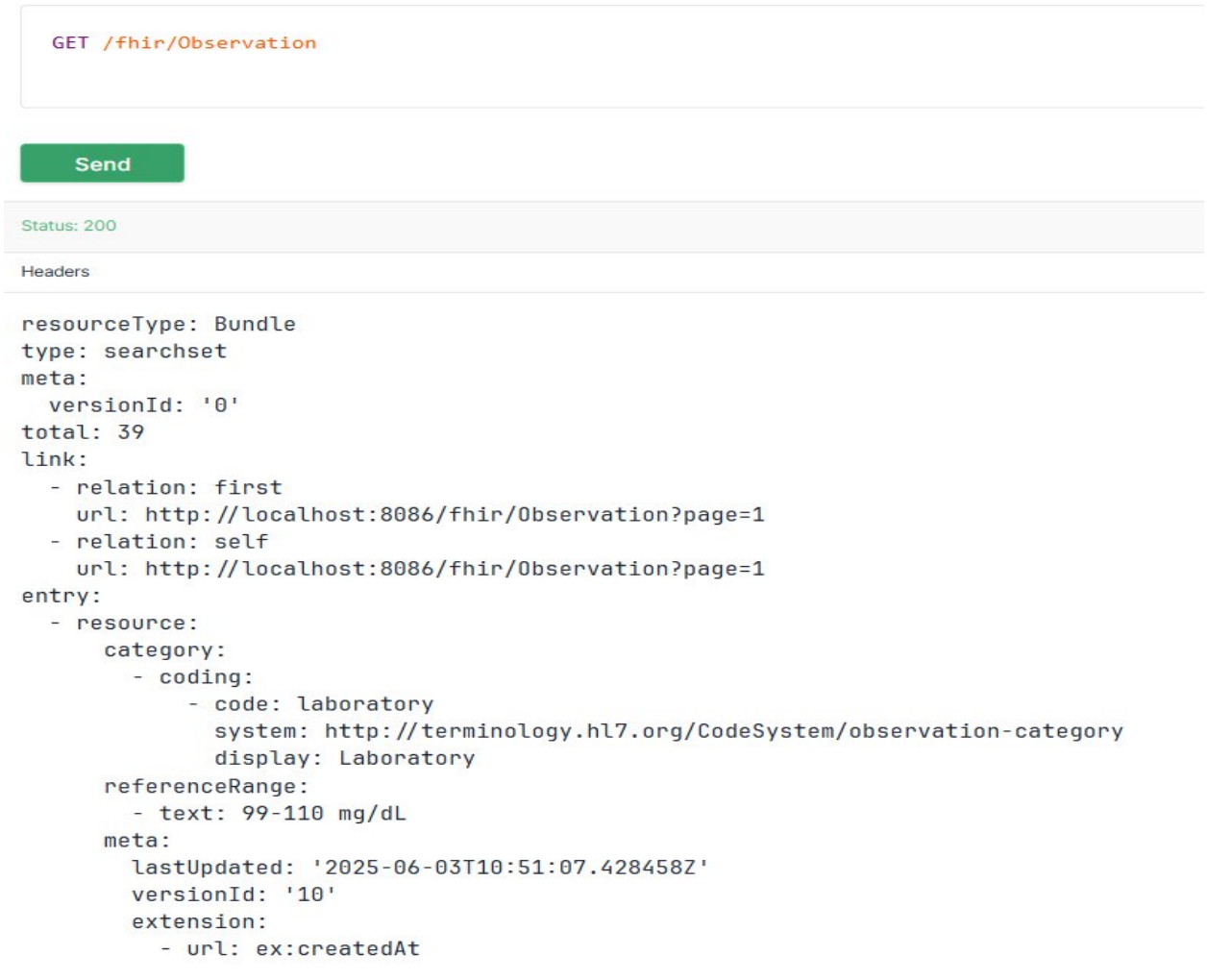


Figura 7GUI de Aidbox con el bundle del recurso Patient + Observation