

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

- **Descripción:**

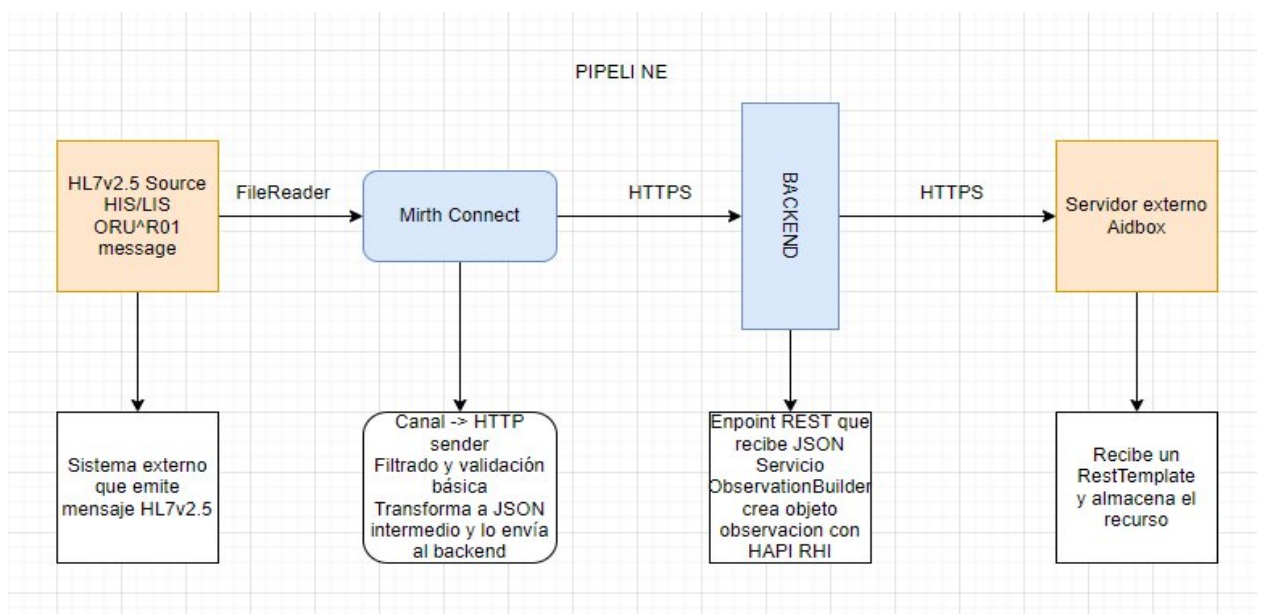
Este proyecto simula un flujo de interoperabilidad clínica donde se reciben mensajes HL7v2 (en este caso de uso: ORU^R01), el cual será procesado inicialmente por Mirth Connect y luego transformado a un recurso FHIR (Observation/Patient) mediante una aplicación en Spring Boot con la librería HAPI FHIR.

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 Connect

- 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 hacia 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 transporte (como MLLP) y realiza transformaciones de formato iniciales (en este caso de HL7v2 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 interoperabilidad y la interacción con APIs de alto nivel. Permite un desarrollo más robusto, escalable y testeable. Si se pretendiese hacer toda la transformación directa (HL7v2 a FHIR) solo en Mirth, el JavaScript del transformador sería extremadamente complejo y difícil 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

```

ORU^R01.hl7
C: > Users > juanf > OneDrive > Desktop > ORU^R01 > ORU^R01.hl7
1 MSH|^~&|LAB_APP|HOSPITAL_A|EMR_SYSTEM|HOSPITAL_B|20250602123000||ORU^R01|MSG00001|P|2.5
2 PID|1||12345678M^^^DNI^NI||REYES^JUAN||19850115|M||123 AV^^DE_ESPAÑA^MAD^90210^ESP||(34)123-4567||M|ES|123-45-6789||
3 PV1|1|I|MED^CARD^01||||2000^SEGURA^SANTIAGO^DR|||||A01|20250601080000|
4 ORC|RE|ORD123|RES456|CM|||||20250601090000|
5 OBR|1|ORD123|RES456|GLU^Glucose^L|||20250601093000|||||20250601100000||||LABTECH^MARY|||||
6 OBX|1|NM|GLUC^Glucose^L|1|105|mg/dL|99-110 mg/dL|H||F||20250601101500|LABTECH^MARY|
  
```

Figura 2 Evento HL7v2.5 de tipo ORU

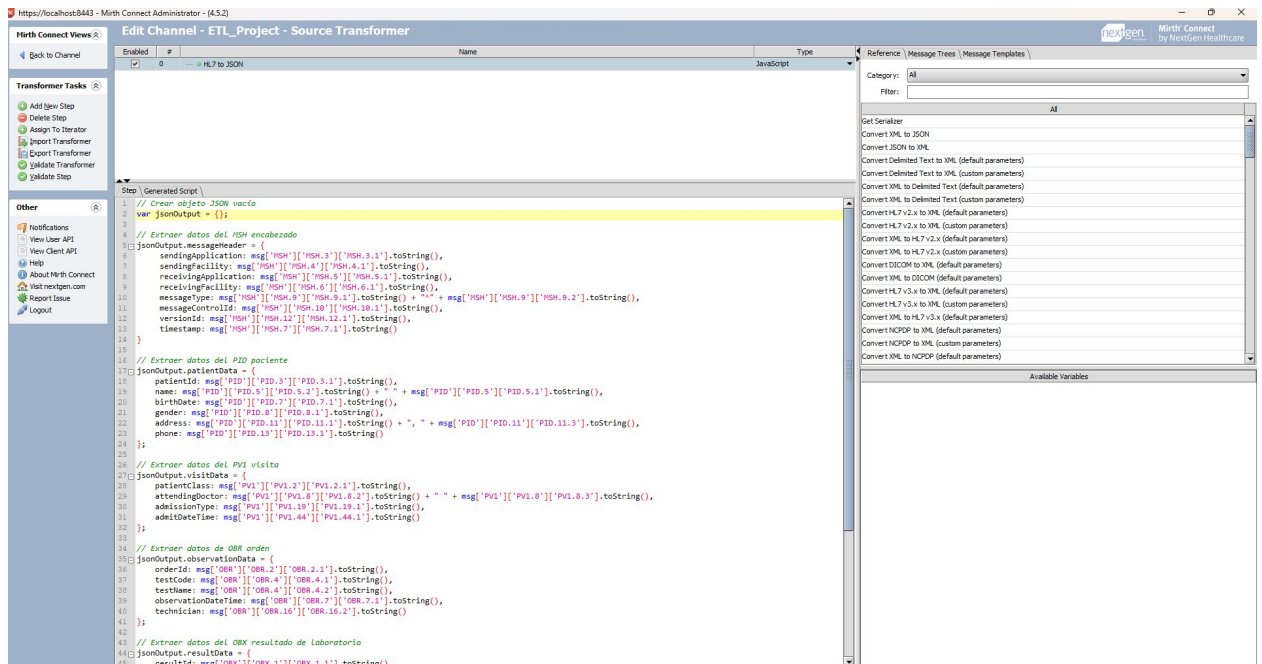


Figura 3 Transformador en JavaScript

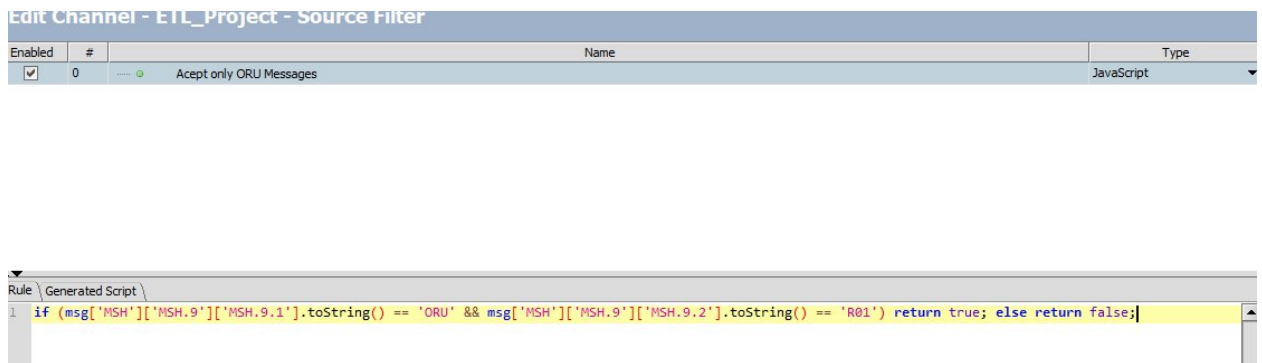


Figura 4 Filtro para solo aceptar mensajes de tipo ORU

Messages								
Mappings								
Raw Encoded Sent Response								
URL: http://localhost:8081/api/h1/7/receive								
METHOD: POST								
[HEADERS]								
[PARAMETERS]								
[CONTENT]								
{								
"messageHeader": {								
"sendingApplication": "LAB_APP",								
"sendingFacility": "HOSPITAL_A",								
"receivingApplication": "EMR_SYSTEM",								
"receivingFacility": "HOSPITAL_B",								
"messageType": "ORU-R01",								
"messageControlId": "MSG00001",								
"versionId": "2.5",								
"timestamp": "20250602123000"								
},								
"patientData": {								
"patientId": "123456789",								
"name": "JUAN REYES",								
"birthdate": "19850115",								
"gender": "M",								
"address": "123 AV. DE ESPAÑA",								
}								
}								

Figura 5 Canal desplegado y JSON transformado y enviado

```

    "issued": "2025-06-01T09:30:00.000+02:00",
    "valueQuantity": {
      "value": 105.0,
      "unit": "mg/dL",
      "system": "http://unitsofmeasure.org",
      "code": "mg/dL"
    },
    "interpretation": [ {
      "coding": [ {
        "system": "http://terminology.hl7.org/CodeSystem/v2-0078",
        "code": "H",
        "display": "H"
      } ]
    } ],
    "referenceRange": [ {
      "text": "99-110 mg/dL"
    } ]
  },
  "response": {
    "status": "201",
    "location": "/Observation/ec29fb30-e027-4231-901f-dfcd63affeb7/_history/251",
    "etag": "251",
    "lastModified": "2025-06-03T11:04:10.040799Z"
  }
}
]
5-06-03T13:04:10.099+02:00 INFO 24260 --- [myetl] [nio-8081-exec-6] com.myetl.service.FhirProcessorService : Recurso /Patient/fa6cfef2-24f0-466d-a3f3-cf4864f64646/_history/249 procesado con ID final: null
5-06-03T13:04:10.099+02:00 INFO 24260 --- [myetl] [nio-8081-exec-6] com.myetl.service.FhirProcessorService : Recurso /Observation/ec29fb30-e027-4231-901f-dfcd63affeb7/_history/251 procesado con ID final: null
5-06-03T13:04:10.099+02:00 INFO 24260 --- [myetl] [nio-8081-exec-6] com.myetl.service.FhirProcessorService : Procesamiento de DTO ORU R01 completado exitosamente.

```

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

GET /fhir/Observation

Send

Status: 200

Headers

```

resourceType: Bundle
type: searchset
meta:
  versionId: '0'
total: 39
link:
  - relation: first
    url: http://localhost:8086/fhir/Observation?page=1
  - relation: self
    url: http://localhost:8086/fhir/Observation?page=1
entry:
  - resource:
      category:
        coding:
          - code: laboratory
            system: http://terminology.hl7.org/CodeSystem/observation-category
            display: Laboratory
      referenceRange:
        - text: 99-110 mg/dL
      meta:
        lastUpdated: '2025-06-03T10:51:07.428458Z'
        versionId: '10'
        extension:
          - url: ex:createdAt

```

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