Trabajo de fin de grado
Ingeniería de las tecnologías de Telecomunicación

Diseño y desarrollo de un bróker FHIR-jBPM

Autor: Juan Manuel Brazo Mora
Tutora: Isabel Román Martínez

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2024

Trabajo de fin de grado
Ingeniería de las tecnologías de Telecomunicación

Diseño y desarrollo de un bróker FHIR-jBPM

Autor:
Juan Manuel Brazo

Tutora:
Isabel Román Martínez
Profesora Colaboradora

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2024

Trabajo Fin de Carrera: Diseño y desarrollo de un bróker FHIR-jBPM

Autor: Juan Manuel Brazo Mora

Tutora: Isabel Román Martínez

El tribunal nombrado para juzgar el Trabajo arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2024

El Secretario del Tribunal

A mi familia

A mis maestros

AGRADECIMIENTOS

Quisiera expresas mi agradecimiento a todas las personas que me han apoyado y guiado durante la realización de toda la carrera.

A mis profesores de la Universidad de Sevilla, que me han proporcionado una sólida base académica fundamental para mi desarrollo profesional.

A mi tutora por guiarme y ayudarme durante toda la realización del proyecto.

A mi familia, por su apoyo incondicional durante todo el trayecto.

Juan Manuel Brazo Mora

Sevilla, 2024

RESUMEN

El objetivo principal de este proyecto es diseñar e implementar un bróker que permita la comunicación indirecta entre servidores FHIR y servidores KIE.

Utilizando el entorno Spring y siguiendo el patrón Modelo-Vista-Controlador, se ha desarrollado una aplicación que gestiona suscripciones a servidores FHIR, para recibir notificaciones de eventos de suscripción, y comunica a servidores KIE estos eventos mediante señales.

Se ha utilizado el servidor desarrollado por HAPI FHIR como servidor FHIR. Para el envío de solicitudes HTTP para realizar pruebas al servidor FHIR se ha utilizado el software Postman.

Como tecnología de soporte a BPM se ha usado jBPM, servidores KIE y Business Central que nos permite modelar procesos de negocio mediante la notación BPMN.

ABSTRACT

The main objective of this project is to design and implement a broker that enables indirect communication between FHIR servers and KIE servers.

Using the Spring framework and following the Model-View-Controller pattern, an application has been developed to manage subscriptions to FHIR servers, receive event notifications, and communicate these events to KIE servers through signals.

The FHIR server developed by HAPI FHIR has been used as the FHIR server. Postman software has been employed to send HTTP requests and conduct tests on the FHIR server.

For BPM support technology, jBPM, KIE servers, and Business Central have been used, allowing us to model business processes using BPMN notation.

ÍNDICE

ÍNDICE DE ILUSTRACIONES

1 INTRODUCCIÓN

1.1 Motivación

En el ámbito de la salud, la eficiencia en el manejo de datos y la automatización de procesos son clave para proporcionar un servicio de calidad. Las actividades desarrolladas en las organizaciones dan lugar a procesos, estos pueden estar formalmente definidos o no. El paradigma de gestión de procesos empresariales (BPM) [1] aboga por analizar y especificar formalmente los procesos y por el desarrollo de tecnologías que faciliten su ejecución, monitorización y análisis, en pro de su continua optimización. En el ámbito sanitario, los procesos asistenciales pueden requerir la participación de diferentes organizaciones y, en consecuencia, el apoyo de sistemas de información heterogéneos. Los estándares como FHIR (Fast Healtcare Interoperability Resources) [2] facilitan la interoperatividad y pueden suponer un importante apoyo para el desarrollo de procesos interorganizacionales. En este proyecto se pretende diseñar y desarrollar una solución que facilitará la ejecución de dichos procesos.

1.2 Objetivos

El objetivo principal de este Proyecto es diseñar y desarrollar una solución que facilite la comunicación indirecta entre servidores FHIR [2] y motores de procesos. De este modo los eventos de interés en servidores FHIR se recibirán en los motores de procesos. La solución será conforme a la especificación FHIR para el marco de trabajo de suscripciones basadas en tópicos [3] y usará las APIs de jBPM [4] para interaccionar con servidores KIE.

1.3 Organización del trabajo

Para este trabajo se han definido varias etapas:

- Estudio de las tecnologías: En esta fase se ha dedicado el tiempo a estudiar todas las tecnologías usadas en el Proyecto (Software, bibliotecas…).
- Diseño de la solución
- Codificación
- Pruebas

1.4 Organización del documento

A continuación, se detallan los puntos de los que se compone esta memoria:

- Introducción: Este capítulo introduce la motivación detrás del proyecto, los objetivos que se buscan alcanzar y cómo está organizado el documento.
- Estado de la Técnica: Se detallan las tecnologías usadas como BPM, jBPM y su integración con Spring Boot, y se detallan los aspectos relevantes de FHIR, incluyendo el framework de suscripción y los recursos utilizados.
- Trabajo Realizado: Se presenta una descripción detallada de la arquitectura de la solución desarrollada.
- Conclusiones y Líneas Futuras: Este capítulo ofrece un resumen de los logros del proyecto, evalúa el cumplimiento de los objetivos planteados, y sugiere posibles mejoras y áreas de investigación futura.
- Referencias: Incluye todas las fuentes de información y documentación que se utilizaron para la realización del proyecto.

2 TECNOLOGÍAS USADAS

En este apartado se describen estándares y tecnologías que se van a usar en el desarrollo de este trabajo de fin de grado.

2.1 FHIR

FHIR (Fast Healtcare Interoperability Resources) [2] , es un estándar de interoperabilidad en el campo de la salud desarrollado por HL7. El principal objetivo de FHIR es facilitar el intercambio de información clínica de forma rápida, eficiente y estandarizada. Esta información es accesible a través de APIs RESTful que nos permiten operaciones para crear, leer, actualizar, eliminar u otro tipo de operaciones personalizadas. Sobre esta base, FHIR define otros mecanismos de intercambio de datos (que pueden verse en la Ilustración 1) entre los que se encuentra el mecanismo de suscripción, utilizado en este trabajo.

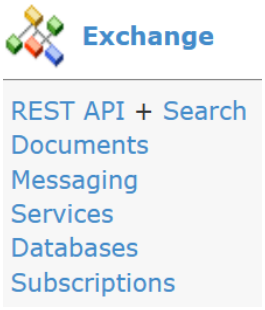

Ilustración 1: Mecanismos de intercambio de datos de FHIR.

FHIR define un total de 157 recursos de datos hasta la fecha. De todos los recursos nos interesa definir especialmente 3 de ellos:

-Subscription, SubscriptionStatus y SubscriptionTopic, usados para las suscripciones.

2.1.1 Mecanismo de suscripción

El mecanismo de suscripción [5] se utiliza para poder tener un modelo de comunicación que siga el patrón publicador/suscriptor entre servidores y clientes FHIR, de modo que los servidores FHIR puedan actuar como publicadores de eventos y los clientes FHIR como suscriptores de los mismos.

Para la comunicación, FHIR ofrece varias opciones de canales de comunicación:

- REST-Hook: Canal que permite el envío de mensajes de forma asíncrona. Para comunicarnos mediante este canal es necesario un endpoint, identificado con una URL, donde se envían las notificaciones, este endpoint debe atender peticiones HTTP. Este es el canal elegido en este proyecto.
- Websocket: Los Websocket permiten abrir una sesión de comunicación activa entre un usuario y un servidor. Este canal es útil en ciertos contextos ya que no requiere de un endpoint en el cliente.
- Email: Las notificaciones se envían a través de correos, esta opción tiene una alta latencia.
- FHIR Messaging: Mecanismo que permite la comunicación no RESTful entre servidores y clientes FHIR.
- Canales personalizados: FHIR permite la definición de canales personalizados en caso de que no sirva ninguno de los anteriores.

El mecanismo de suscripción tiene dos actores:

- Publicador: Es el servidor FHIR que hace de proveedor de eventos. Debe tener uno o varios SubscriptionTopics. Se encarga de gestionar las suscripciones y enviar las notificaciones.
- Suscriptor: Hace referencia a las aplicaciones que reciben los eventos a los cuales se han suscrito previamente. Estos eventos se reciben en el endpoint especificado en la suscripción, este endpoint es una URL donde el cliente escucha peticiones HTTP POST.

Para implementar el mecanismo de suscripción el servidor debe ser capaz de:
- Permitir acciones de lectura, escritura y búsqueda sobre el recurso Subscription.
- Permitir la operación $status (Utilizada para obtener información del estado de la suscripción) sobre el recurso Subscription
- Permitir la operación de lectura y búsqueda sobre el recurso SubscriptionTopic
- Proporcionar al menos un canal de comunicación
- Proporcionar al menos un método de entrega de datos.

En una suscripción están implicados los siguientes recursos FHIR:
- Recursos SubscriptionTopic que definen los datos (recursos, acciones y filtros) que se podrían notificar al suscriptor
- Recursos Subscription que definen las peticiones de los clientes de ser notificados de un tópico. Indican el canal por el que se notifica, los filtros establecidos y la forma en que se entregan los datos.
- Recursos SubscriptionStatus que definen el estado de una suscripción.

2.1.1.1 SubcriptionTopic

Un SubscriptionTopic [3] define un conjunto de eventos a los que un cliente puede suscribirse. En un SubscriptionTopic se puede indicar el recurso al que podemos suscribirnos y qué tipo de acción podría disparar una notificación, que puede ser crear, modificar o eliminar el recurso. También define un conjunto de filtros que se pueden aplicar en la suscripción como valores concretos o comparaciones de campos del recurso, esto nos permitiría, por ejemplo, suscribirnos a un tópico para que nos notificara la creación de un paciente cuya fecha de nacimiento sea posterior a una fecha. Contiene campos como:

1. url: identificador único del tópico, representado como una URI global y única
2. status: código que puede ser “draft, active, retired o unknown”
    a. Draft: Todavía en desarrollo, no está disponible para su uso
    b. Active: Tópico activo, disponible para su uso
    c. Retired: Tópico suspendido
    d. Unknown: Estado desconocido
3. resourceTrigger: Recurso que se utiliza como disparador. Puede ser cualquiera de los recursos de FHIR, se debe indicar la acción o acciones realizadas sobre dicho recurso que queremos que notifique (create, update o delete).
5. filterParameter: Se pueden crear filtros para una mayor especificidad en la definición de la suscripción

Un tópico debe especificar:
1. Qué causa la notificación, bien haciendo referencia a un evento que se produzca en el servidor o describiendo el cambio de estado de un recurso.
2. Qué filtros puede aplicar un cliente. Por ejemplo, elegir notificaciones de un paciente en concreto.
3. Qué recursos relacionados puede incluir el servidor en la notificación, por ejemplo, si se actualiza un recurso cita (“Encounter”), sería interesante incluir información del paciente con los recursos “observations”, “diagnosed conditions” …

2.1.1.2 Subscription

El recurso Subscription [6] establece la forma en la que se envían las notificaciones de los eventos relacionados con un SubscriptionTopic en un cliente específico, definiendo el contenido de la notificación y el canal por el cual se envían.

Ajustando Subscription.content los suscriptores pueden elegir la información que se le pasa en las notificaciones:

1. empty: solo se notifica, no recibe información
2. id-only: se notifica el id del recurso que se ha creado/modificado/eliminado
3. full-resource: se envía el recurso completo.
4. También se pueden elegir campos específicos del recurso.

El suscriptor debe elegir el canal por el que se le notifican los eventos. En este proyecto nos centraremos en el canal rest-hook, en el cual las notificaciones se envían mediante HTTP POST al Subscription.endpoint URL.

Para recibir notificaciones vía HTTP POST el cliente debe indicar en la suscripción el tipo de canal rest-hook en subscription-channel-type y un endpoint, en Subscription.endpoint, con la URL a la que debe acceder el servidor para notificar. El servidor realizará un POST con un "notification Bundle" a la URL del endpoint. Este Bundle es un contenedor para una colección de recursos.

A continuación, se muestra un diagrama donde se representan los intercambios de mensajes para establecer una suscripción mediante rest-hook.

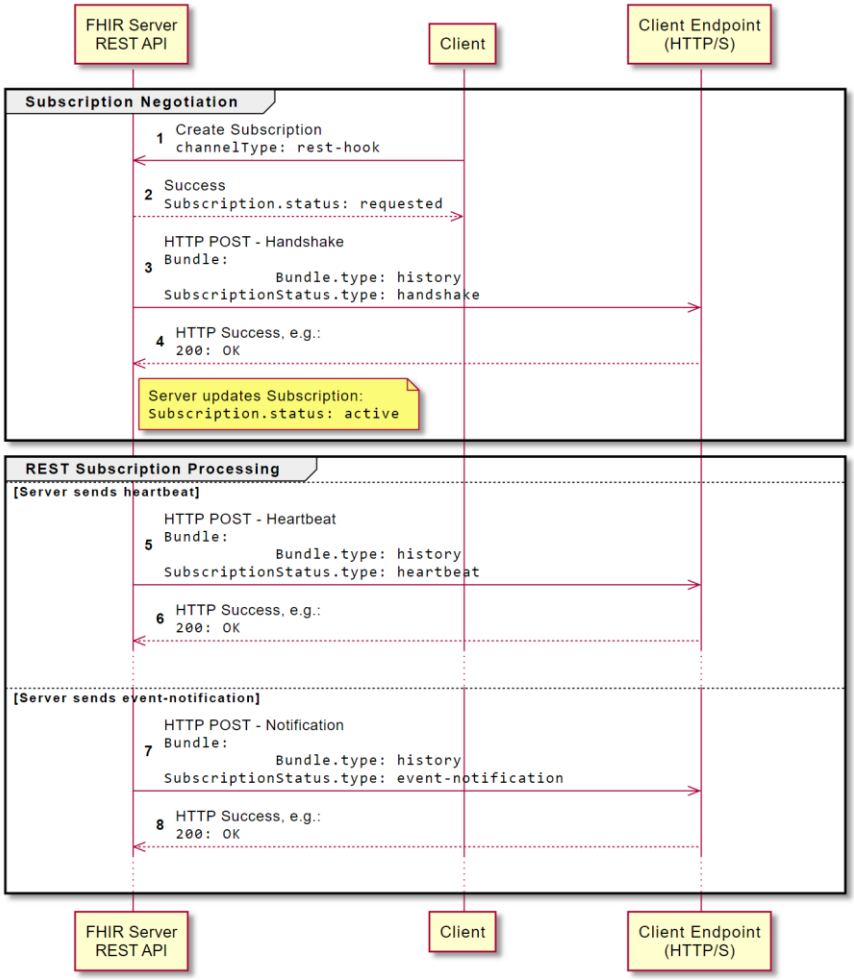

Ilustración 2: Intercambio de mensajes para establecer una suscripción y para enviar notificaciones.

Este diagrama muestra el flujo de mensajes que intercambian el servidor FHIR y el cliente a la hora de establecer un canal de comunicación mediante REST-Hook y los posteriores mensajes para enviar una notificación:

1. El Cliente crea una Suscripción con el channelType configurado en rest-hook.
2. El servidor responde con un código de éxito y crea la suscripción con un estado de solicitado.
3. El servidor realiza un HTTP POST al punto final solicitado con una notificación de saludo.
4. El punto final del cliente acepta el POST y devuelve un código HTTP de éxito (por ejemplo, 200).
5. El servidor envía una notificación de tipo latido en cualquier momento (DEBE ser al menos una vez por periodo de latido).
6. El punto final del cliente acepta un latido a través de HTTP POST y devuelve un código HTTP de éxito (por ejemplo, 200).
7. Cuando ocurre alguno de los eventos disparadores, el servidor envía una notificación de tipo notificación.
8. El punto final del cliente acepta una notificación de evento a través de HTTP POST y devuelve un código HTTP de éxito (por ejemplo, 200).

2.1.1.3 SubscriptionStatus

Cuando ocurre un evento de notificación de suscripción [7] en un servidor FHIR se envía un "Bundle", esto es un contenedor de recursos FHIR con una o más entradas, la primera de estas entradas es siempre un recurso de tipo SubscriptionStatus.

El recurso SubscriptionStatus [8] contiene información relacionada con el estado de una suscripción en el momento en el que se envía la notificación al cliente. Esta información contiene los recursos Subscription y SubscriptionTopic relacionados con el evento de notificación, el número de secuencia del evento y el tipo de evento notificado, en este caso concreto nos interesa el tipo de evento "event-notification".

Este recurso no puede ser persistido ni referenciado por otros recursos.

2.2 BPM

La gestión de procesos de negocio (BPM) [1], es un paradigma diseñado para gestionar procesos de una organización o empresa. Estos procesos están diseñados para automatizar actividades empresariales, mejorar la eficiencia operativa y facilitar la toma de decisiones. Los procesos empresariales en el dominio sanitario pueden abarcar una amplia gama de actividades, desde la gestión de recursos humanos, de inventario o contabilidad hasta la gestión procesos asistenciales o de investigación, entre muchos otros.

2.3 Motor de procesos

Los motores de procesos son componentes clave en la gestión de procesos de negocio (BPM). Estos motores permiten la automatización, ejecución, monitoreo y optimización de procesos empresariales complejos. Utilizando la notación BPMN (Business Process Model and Notation), los motores de procesos facilitan la ejecución de flujos de trabajo que pueden integrar reglas de negocio, eventos y tareas humanas.

En este proyecto se utiliza el motor de procesos proporcionado por KIE (Knowledge Is Everything) [9].

2.4 JBPM

JBPM (Java Business Process Manager) [4] es una herramienta de código libre que ayuda a la creación de aplicaciones empresariales que pueden interaccionar con motores de procesos KIE. Soporta BPMN [10] (Business Process Model and Notation) que es una notación estandarizada que permite el modelado de procesos de negocio, en un formato de flujo de trabajo (*workflow*).

JBPM también nos permite gestionar y monitorizar una aplicación de negocio durante todo su ciclo de vida.

Una aplicación jBPM se compone de tres módulos:

- KJAR: Contiene el modelado de los procesos de negocio y los activos de la aplicación. Este modelado se puede realizar de forma sencilla desde Business Central (Que se explicará en el siguiente punto).
- Model: Este módulo define los POJOs (Plain Old Java Objects) compartidos entre los módulos KJAR y service. Estos POJOs definen los datos manejados por el sistema.
- Service: Es la aplicación principal construida con Spring Boot (Explicado más adelante). En él se define la lógica de la aplicación.

2.5 Business Central

Business Central [11] es un servicio web incluido en la suite de herramientas de jBPM (Java Business Process Management) de Red Hat, que facilita la gestión, modelado y monitoreo de procesos de negocio. Proporciona un entorno completo para diseñar, implementar y administrar procesos de negocio utilizando la notación BPMN (Business Process Model and Notation).

Algunas de las características clave de Business Central son:

- Modelado Gráfico: Permite a los usuarios diseñar procesos de negocio mediante una interfaz gráfica intuitiva que soporta BPMN 2.0.
- Gestión de Procesos: Facilita la administración de los procesos de negocio, incluyendo la versión y el despliegue de procesos en el entorno de producción.
- Monitoreo y Reportes: Proporciona herramientas para monitorear la ejecución de procesos en tiempo real, generar informes y analizar el rendimiento de los procesos.
- Reglas de Negocio: Integración con el motor de reglas Drools, permitiendo la definición y ejecución de reglas de negocio complejas dentro de los procesos.
- Gestión de Tareas: Administración de tareas humanas asociadas a los procesos de negocio, incluyendo la asignación, seguimiento y cumplimiento de tareas.
- Soporte para Formularios: Creación y gestión de formularios para capturar y presentar datos en los procesos de negocio.

En este proyecto usaremos Business Central para modelar los procesos de negocio que reciban las señales enviadas por nuestro bróker tras recibir notificaciones desde servidores FHIR.

2.6 Java Spring Framework

Java Spring Framework [12] es un marco de trabajo de código abierto que permite crear aplicaciones autónomas para Java. Spring Boot es una herramienta que simplifica el desarrollo de aplicaciones con Spring Framework mediante configuraciones automáticas.

Las aplicaciones Spring Boot se ejecutan de forma autónoma (Standalone), por lo que pueden ser ejecutadas directamente desde línea de comandos o empaquetadas en archivos JAR.

Estas aplicaciones utilizan las herramientas Maven o Gradle para la gestión de dependencias, lo que facilita la incorporación de bibliotecas y componentes a nuestra aplicación.

A través del fichero application.properties podemos indicar la configuración del servidor, configuraciones de bases de datos y propiedades personalizadas entre otras.

SpringBoot permite realizar anotaciones que simplifican el código, algunas de las usadas en el Proyecto son:

- @SpringBootApplication: Marca la clase principal de la aplicación y habilita la configuración automática de Spring Boot, el escaneo de componentes y la configuración de Spring.
- @RestController: Utilizada para simplificar la creación de controladores RESTful. Indica que la clase manejará solicitudes HTTP y que los métodos de la clase devolverán datos directamente en el cuerpo de la respuesta.
- @RequestMapping: Se usa para mapear solicitudes HTTP a métodos específicos dentro de un controlador.
- @PostMapping: deriva de la anotación RequestMapping e indica que el método va a manejar solicitudes de tipo POST.
- @Autowired: Permite la inyección automática de dependencias, dejando que Spring resuelva e incluya las dependencias necesarias para un componente de forma automática.
- @RequestBody: Indica que el parámetro está asociado al cuerpo de una solicitud web, se utiliza para leer datos de una solicitud en formato XML o JSON y convertirlos a un objeto Java.

jBPM se integra de manera sencilla con Spring Boot, facilitando la configuración y el despliegue de aplicaciones empresariales.

2.7 Postman

Postman [13] es un Software utilizado para hacer test de APIs creado en 2012 por Abhinav Asthana.

En este Proyecto utilizaremos Postman para realizar las peticiones HTTP necesarias a nuestro servidor FHIR para probar la aplicación.

2.8 Docker

Docker [14] es un conjunto de herramientas que facilita el uso de aplicaciones a través de la virtualización basada en contenedores.

Usaremos Docker para levantar servidores KIE con Business Central

2.9 HAPI FHIR

La biblioteca HAPI FHIR [15] es una implementación de la especificación HL7 FHIR que nos ofrece una API para Java que permite interactuar con recursos FHIR de forma simplificada. Esto nos permitirá manejar los recursos FHIR en nuestra aplicación.

También usaremos un servidor JPA que usa esta biblioteca para crear nuestra BBDD FHIR. Este servidor se encuentra en un repositorio de Github: https://github.com/hapifhir/hapi-fhir-jpaserver-starter

2.10 Java Persistence API (JPA)

JPA [16] es una API que proporciona un mecanismo para gestionar la persistencia de objetos en Java. La persistencia de datos es un medio por el que una aplicación puede almacenar y recuperar información desde un sistema de almacenamiento no volátil.

2.11 Thymeleaf

Thymeleaf [17] es un motor de plantillas utilizado principalmente para el desarrollo de aplicaciones web, permite escribir plantillas HTML y se integra fácilmente con el entorno Spring, facilitando la inyección de datos desde controladores directamente en las vistas.

3 TRABAJO DESARROLLADO

3.1 Requisitos

Para el desarrollo de esta aplicación se definen una serie de requisitos que debe cumplir:

- Desarrollar un cliente FHIR para manejar recursos relacionados con la gestión de suscripciones.
- Desarrollar endpoints REST con Spring Boot para manejar las notificaciones recibidas al ocurrir un evento determinado en un servidor FHIR, obteniendo los datos de interés. Estos endpoints se generarán de manera dinámica de forma que existan múltiples endpoints para poder manejar distintas notificaciones. Las configuraciones de los endpoints se persisten en un repositorio de entidades JPA. Existirá un endpoint por cada pareja de acción-recurso, se creará una entidad endpoint que almacena la acción, el recurso, el id de la entidad y el nombre de la señal que se enviará al servidor KIE. Por ejemplo, para una suscripción que notifique de la creación de un recurso FHIR de tipo ServiceRequest habrá una entidad endpoint con los siguientes datos:
    - Id:1.
    - Resource: ServiceRequest.
    - Interaction: create.
    - SignalName: create-ServiceRequest.
- Automatizar el envío de señales a servidores KIE, estas señales serán enviadas por la aplicación tras recibir una notificación en un endpoint. Estos servidores se podrán agregar de forma dinámica, almacenando su información (url y credenciales del servidor) en un repositorio de entidades JPA.

3.2 Arquitectura de la solución

En este apartado se desarrolla de manera detallada la solución implementada en este trabajo, describiendo el código de la aplicación, los procesos de negocios diseñados para realizar pruebas y las configuraciones de las herramientas utilizadas para realizar el proyecto.

La solución adoptada para cumplir los objetivos se basa en una arquitectura que utiliza Spring Boot, jBPM y FHIR de manera conjunta para dar soporte a la gestión de procesos de negocio en el ámbito de la salud.

La arquitectura elegida para este proyecto sigue el patrón Modelo-Vista-Controlador (MVC). Esta arquitectura es muy utilizada en el desarrollo de software basado en aplicaciones web ya que nos permite separar la lógica de la aplicación (Controlador), la parte de interfaz de usuario (Vista) y los aspectos de persistencia y representación de la información (Modelo). A continuación, se muestra un esquema del patrón MVC.

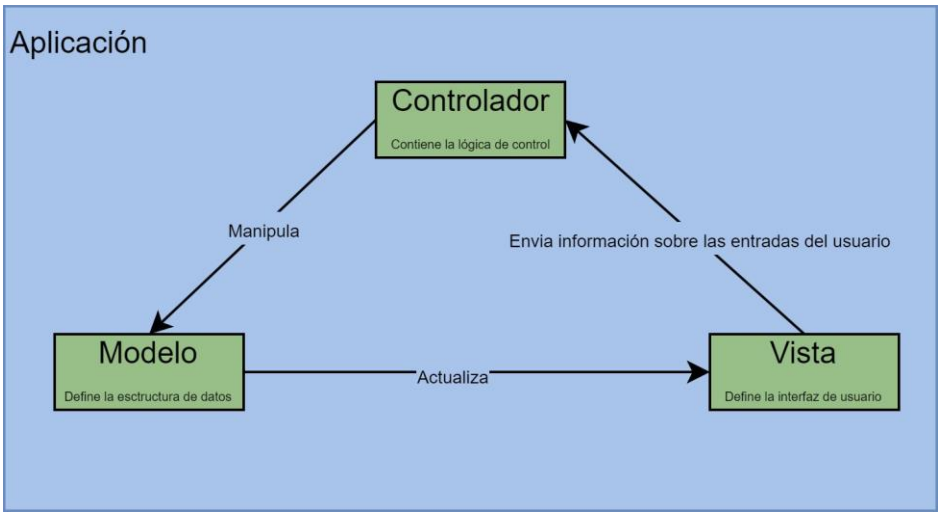

Ilustración 3: Patrón MVC [18]

3.3 Servidor FHIR

Para el servidor FHIR hemos utilizado el código creado por HAPI FHIR, este código lo podemos encontrar en el siguiente repositorio de GitHub: https://github.com/hapifhir/hapi-fhir-jpaserver-starter. Debemos configurar el servidor a través del fichero application.yaml para indicar la versión de FHIR y el Puerto donde queremos alojarlo:

Solo se incluyen las partes modificadas del mismo:

```
server:
# Establecemos el puerto del servidor
  port: 8080
hapi:
  fhir:
    ### Elegimos la versión de FHIR R5
    fhir_version: R5

    ### Habilitamos las suscripciones por resthook
    subscription:
      resthook_enabled: true
```

Código 1: Fragmento de application.yaml.

Una vez configurado nuestro servidor, nos desplazamos con la consola al directorio raíz del servidor y ejecutamos "*mvn spring-boot:run"*. Con esto ya tendríamos el servidor configurado para trabajar con FHIR R5 disponible en http://localhost:8080

3.4 Servidor KIE

Para usar servidores KIE vamos a utilizar la imagen de Docker que proporciona jBPM de su aplicación Business central. Desde el buscador de imágenes de Docker descargamos la imagen "jboss/jbpm-server-full" y podremos instanciar tantos servidores como queramos usando contenedores. Podemos hacerlo directamente desde línea de comandos, para ello debemos ejecutar el siguiente comando:

"docker run -p 8080:8080 -p 8001:8001 -d --name jbpm-server-full jboss/jbpm-server-full:latest"

Donde se deben indicar los puertos de escucha de la aplicación.

3.5 Aplicación bróker FHIR-KIE

Esta aplicación es la que hará de intermediaria entre servidores FHIR y servidores KIE. Para ello necesitamos que la aplicación tengas las siguientes funcionalidades:

1. Permitir la gestión de servidores KIE: La aplicación debe ofrecer al administrador una interfaz web para poder añadir o eliminar servidores KIE, estos servidores serán los que reciban las señales tras las notificaciones de un servidor FHIR.
2. Permitir la gestión de suscripciones a servidores FHIR: La aplicación debe ofrecer al administrador una interfaz web para que se puedan crear suscripciones en distintos servidores FHIR. Para ello la aplicación tiene que gestionar las conexiones a distintos servidores FHIR y los recursos Subscription y SubscriptionTopics de los servidores FHIR.
3. Gestionar las notificaciones recibidas por un servidor FHIR para enviar las señales a los servidores KIE.

La estructura de ficheros es la siguiente:

```
Resources/
├── Solicitudes servidor FHIR.postman_collection.json (Colección de peticiones HTTP de Postman)
├── application.yaml (Fichero de configuración del servidor FHIR modificado para las pruebas)
├── docker-compose.yaml (Fichero para ejecutar directamente con Docker el servidor FHIR y el servidor KIE)
├── fkbroker-kjar (Proyecto de business central con los activos usados para las pruebas)
├── create-ServiceRequest.bpmn (Activo de proceso de negocio para Business Central)
└── update-ServiceRequest.bpmn (Activo de proceso de negocio para Business Central)
fkbroker-service/
├── pom.xml (Dependencias de la aplicación)
└── src/main
    ├── java/us/dit/fkbroker/service
    │       ├── Application.java
    │       ├── conf/ (Configuraciones de la aplicación como aspectos de seguridad)
    │       ├── controllers/ (Controladores)
    │       ├── entities/ (Entidades)
    │       ├── repositories/ (Repositorios de Entidades)
    │       └── services/
    │           ├── fhir/ (Servicios relacionados con FHIR)
    │           └── kie/ (Servicios relacionados con KIE)
    └── resources/
        ├── application.properties (Fichero de configuración del broker)
        ├── static/ (archivos estáticos como CSS, JavaScript, imágenes)
        └── templates/ (archivos html con las vistas)
```

La aplicación está disponible en el siguiente repositorio de Github: https://github.com/tfg-projects-dit-us/FKBroker/tree/master. A continuación, se detalla el código desarrollado.

3.5.1 Gestión de servidores KIE

El bróker permite al administrador gestionar los servidores KIE a los que se enviarán las señales. Considerando la interacción del administrador con el bróker, los casos de uso elementales para la gestión de servidores KIE son:

1. El administrador va a la dirección http://ip_aplicación:puerto/kieServers , introduciendo la url directamente o a través del menú "Gestión de servidores KIE" que proporciona la interfaz web.
2. La aplicación consulta los servidores KIE ya configurados al repositorio de servidores KIE y los muestra como una tabla donde se indica la url, el usuario y la contraseña. A la derecha de cada fila existe un botón que permite borrar el servidor del repositorio.
3. Debajo de la tabla aparece un formulario para crear entidades KieServer donde se debe indicar url, usuario y contraseña del servidor que queremos añadir al repositorio de servidores.

3.5.1.1 Código

En primer lugar, crearemos la entidad KieServer.java y el repositorio de servidores KieServerRepository.java

- KieServer.java tiene tres atributos: url, usu y pwd. Los cuales hacen referencia a la url donde se envían las señales, el usuario y la contraseña del servidor respectivamente. Estos atributos tienen sus métodos set() y get().
- KieServerRepository.java: Interfaz para crear nuestro repositorio de KieServers, esta interfaz extiende la interfaz JpaRepository<KierServer, String>. Al configurarlo como un repositorio no hace falta desarrollar los métodos de esta clase, se encarga SpringBoot.

Una vez creadas la entidad y el repositorio, creamos el controlador KieController.java. Este controlador, atiende a las peticiones realizadas a la url http://ip_aplicación:puerto/kieServers, ofrece el constructor y tres métodos:

1. @GetMapping
   public String getKieServers(Model model): ofrece un listado de KieServers a la vista al recibir una petición HTTP get a la url.
2. @PostMapping("/add")
   public String addKieServer(@ModelAttribute KieServer kieServer): Permite crear entidades KieServer. Cuando rellenamos el formulario para añadir servidores KIE y se envía, se manda una solicitud HTTP post a http://ip_aplicación:puerto/kieServers/add con la información del servidor en el cuerpo del mensaje.
3. @PostMapping("/delete")
   public String deleteKieServer(@RequestParam String url): Permite borrar entidades KieServer. Cuando eliminamos un servidor KIE se envía una solicitud HTTP delete a http://ip_aplicación:puerto/kieServers/delete con la url del servidor que queremos borrar incluida en el cuerpo del mensaje.

Estas acciones las realiza llamando a métodos de la clase KieServerService.java, la cual está configurada como "service" en SpringBoot a través de anotaciones. Esta clase ofrece cuatro métodos:

1. public List<KieServer> getAllKieServers() : Obtiene todas las entidades KieServer del repositorio.
2. public KieServer saveKieServer(KieServer kieServer): Guarda un nuevo KieServer en el repositorio.
3. public void deleteKieServer(String url): Elimina un KieServer del repositorio.
4. public void sendSignalToAllKieServers(NotificationEP notificationEP, String mensaje): Esta función es la que se encarga de enviar una señal a los servidores KIE cuando llega una notificación FHIR, se verá más adelante.

Por último, la vista que recibe el usuario es KieServers.html. Esta es una plantilla desarrollada con Thymeleaf. A continuación, podemos ver el diseño de esta plantilla.

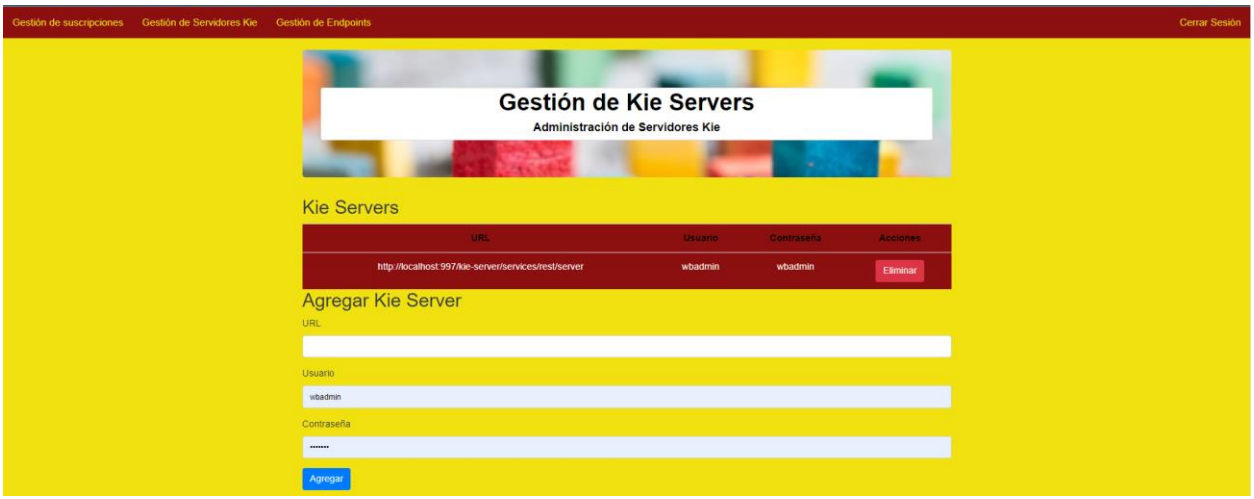

Ilustración 4: Vista de Interfaz de usuario para la gestión de servidores KIE

3.5.2 Gestión de suscripciones

Para la gestión de suscripciones, como se ha comentado anteriormente, la aplicación debe ofrecer una interfaz que permita al administrador conectarse a un servidor FHIR y gestionar las suscripciones en el mismo. Para ello el funcionamiento es el siguiente:

1. El administrador ingresa la <IP:Puerto> del servidor FHIR al que desea conectarse
2. La aplicación consulta al servidor los SubscriptionTopics y las Subscriptions relacionadas con nuestro broker existentes y se las muestra al administrador en forma de tabla, indicando los valores relevantes de cada uno de ellos; como el nombre, el id o los filtros.
3. En la parte inferior se encuentra un formulario donde el administrador puede elegir a qué SubscriptionTopic suscribirse, una vez elegido se mostrará un formulario para elegir el valor de los filtros que se deseen configurar.
4. La aplicación comprueba si existe una entidad NotificationEP en el repositorio de endpoints que contenga la pareja de interacción-recurso. Si ya existiera se le asignaría como endpoint a la suscripción http://ip_aplicación:puerto/endpoint/{id}, donde {id} es el id de la entidad NotificationEP. Si no existiera esa pareja, crearía la nueva entidad NotificationEP y la guardaría en el repositorio de endpoints, asignándole como id el id de la entidad creada. Este id nos permitirá identificar la entidad NotificationEP relacionada para asignarle el nombre a la señal que se envía a los servidores KIE al recibir una notificación.
5. La aplicación envía al servidor FHIR al que estamos conectados, mediante HTTP POST, la información de la suscripción en un JSON para que se guarde en el servidor.
6. Finalmente nos devolvería a la página donde se visualizan los SubscriptionTopics y Subscriptions, reflejándose la nueva suscripción creada

3.5.2.1 Código del repositorio de endpoints

En primer lugar, crearemos la entidad NotificationEP.java y el repositorio NotificationEPRepository.java.

- NotificationEP.java tiene cuatro atributos: id(clave), resource, interaction y signalName. Los cuales hacen referencia al id de la entidad (que se usará para el endpoint de la suscripción), el recurso relacionado con el SubscriptionTopic, la interacción relacionada con el SubscriptionTopic (create o update) y el nombre de la señal (que se forma con la pareja interaction-resource) respectivamente. Estos atributos tienen sus métodos set() y get().

- NotificationEPRepository.java: Interfaz para crear nuestro repositorio de NotificationEPs, esta interfaz extiende la interfaz JpaRepository<NotificationEP, Long>. Al configurarlo como un repositorio no hace falta desarrollar los métodos de esta clase, se encarga SpringBoot, a excepción del método "*Optional<NotificationEP> findByResourceAndInteraction (String resource, String interaction)*", el cual se encarga de comprobar si existe una entidad que contenga los campos "resource" e "interaction" indicados, en caso afirmativo la devuelve.

Una vez creadas la entidad y el repositorio, creamos el controlador NotificationEPController.java. Este controlador atiende a las peticiones realizadas a http://ip_aplicación:puerto/notificationEPs. Ofrece tres métodos:

1. @GetMapping
   public String getNotificationEPs(Model model): ofrece un listado de NotificationEPs a la vista al recibir una petición HTTP get a la url.
2. @PostMapping("/add")
   public String addNotificationEP(@ModelAttribute NotificationEP notificationEP): Permite crear entidades NotificationEP, aunque se generen de manera automática, nos puede interesar crear una de forma manual para asignar un nombre de señal distinto al que se genera por defecto. Al crearla de forma manual desde el formulario se envía una petición HTTP post a http://ip_aplicación:puerto/notificationEPs/add con los datos de la entidad a crear incluidos en el cuerpo del mensaje.
3. @PostMapping("/delete")
   public String deleteNotificationEP(@RequestParam Long id): Permite borrar entidades NotificationEP. Al eliminar una entidad NotificationEP se envía una solicitud HTTP delete a http://ip_aplicación:puerto/notificationEPs/delete con el id de la entidad que deseamos eliminar incluida en el cuerpo del mensaje.
4. @PostMapping("/edit")
   public String editNotificationEP (@RequestParam Long id, @RequestParam String signalName): Permite editar el valor signalName de una entidad NotificationEP especificada con el id. Cuando editamos el nombre de la señal se envía una solicitud HTTP post a http://ip_aplicación:puerto/notificationEPs/edit con el id de la entidad que deseamos editar y el nuevo nombre de la señal incluidos en el cuerpo del mensaje.

Estas acciones las realiza llamando a métodos de la clase NotificationEPService.java, la cual está configurada como "service" en SpringBoot a través de anotaciones. Esta clase ofrece cinco métodos:

1. public List<NotificationEP> getAllNotificationEPs(): Obtiene todas las entidades NotificationEP del repositorio.
2. public NotificationEP saveNotificationEP(NotificationEP notificationEP): Guarda una nueva entidad en el repositorio.
3. public void deleteNotificationEP(Long id): Elimina un NotificationEP del repositorio.
4. public Optional<NotificationEP> findNotificationEPByResourceAndInteraction(String resource, String interaction): Esta función es la que se encarga de comprobar si existe en el repositorio una entidad con los parámetros indicados.
5. public Optional<NotificationEP> findById(Long id): Busca en el repositorio una entidad por el id

Por último, queda la vista que recibe el administrador, la cual es notificationEPs.html. Esta es una plantilla desarrollada con Thymeleaf. En esta plantilla se ha configurado una tabla que muestra todas las entidades NotificationEP del repositorio del bróker y un formulario para crear NotificationEPs personalizados, donde debemos introducir el recurso, la interacción y el nombre de la señal. También podemos eliminar una entidad o editar su valor de signal name desde la propia interfaz web. En la siguiente Ilustración podemos ver el diseño de esta plantilla.

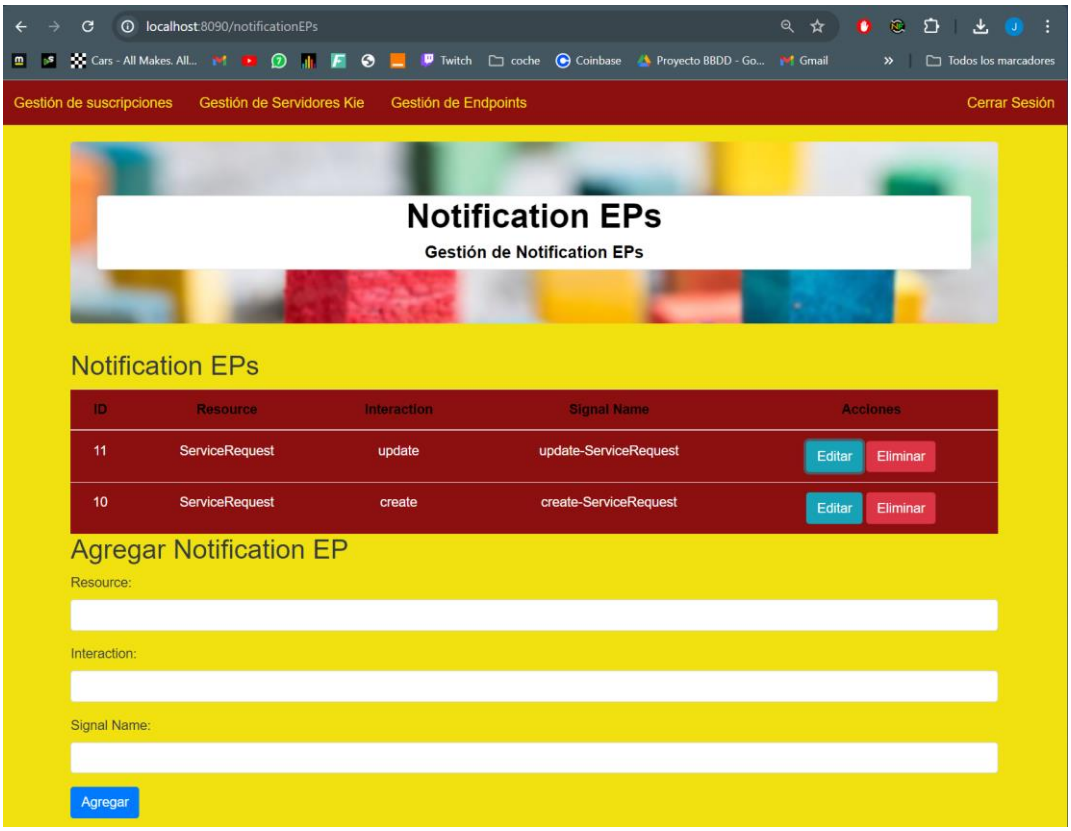

Ilustración 5: Interfaz de notificationEPs.html

3.5.2.2 Código de suscripciones

En primer lugar, creamos el controlador SubscriptionController.java, que se encargará de gestionar todo lo relacionado con peticiones de suscripciones. Ofrece cinco métodos:

1. @GetMapping("/")
   public String getHomePage(Model model): ofrece al usuario la vista donde podrá introducir el servidor FHIR al que desea conectarse al recibir una solicitud HTTP get en la url http://ip_aplicación:puerto.
2. @GetMapping("/subscriptions")
   public String getSubscriptionPage(Model model, @RequestParam String fhirUrl): ofrece un listado de los tópicos y suscripciones creadas en el servidor FHIR especificado en la url, por ejemplo "http://ip_aplicación:puerto/subscriptions?fhirUrl=localhost:8080"
3. @PostMapping("/create-subscription")
   public String createSubscription(@RequestParam String topicUrl, @RequestParam String payload, @RequestParam String fhirUrl, Model model): Al recibir una solicitud HTTP post en http://ip_aplicación:puerto/create-subscription recibe en el cuerpo del mensaje la url de un tópico, comprueba los filtros que tiene y comprueba si existe una entidad NotificationEP con la pareja interacción-recurso para crear un nuevo endpoint o asignarle uno existente. Redirecciona a la página donde especificamos el valor de los filtros.
4. @PostMapping("/submit-filters")
   public String submitFilters(@RequestParam Map<String, String> requestParams, @RequestParam String fhirUrl, @RequestParam String endpoint, Model model): Recibe en el cuerpo del mensaje un listado de los filtros disponibles del SubscriptionTopic, una vez seleccionado el valor de los filtros, llama a la función para crear la suscripción en el servidor.

5. @PostMapping("/delete-subscription")
   public String deleteSubscription(@RequestParam String subscriptionId, @RequestParam String fhirUrl): Elimina una suscripción del servidor al recibir una solicitud HTTP delete en http://ip_aplicación:puerto/delete-subscription, el id de la suscripción que deseamos eliminar se incluye en el cuerpo del mensaje.

Este controlador hace uso de la clase SubscriptionService.java para realizar las operaciones, la cual a su vez hace uso de la clase FhirClient.java para realizar todas las operaciones relacionadas con recursos FHIR.

FhirClient se encarga de realizar las peticiones a los servidores FHIR, recibir y enviar los recursos y gestionarlos para entregar la información solicitada a la aplicación. Para realizar estas operaciones se hace uso de la biblioteca de HAPI FHIR [19] que ofrece métodos para manejar la información de un recurso FHIR. La clase FhirClient.java ofrece los siguientes métodos:

1. public List<SubscriptionTopicDetails> getSubscriptionTopics(String fhirUrl): Obtiene una lista de SubscriptionTopics desde un servidor FHIR.
2. public List<SubscriptionDetails> getSubscriptions(String fhirUrl): Obtiene una lista de suscripciones desde un servidor FHIR. Solo muestra las suscripciones relacionadas con nuestro bróker, para ello filtra las suscripciones y devuelve solo las que tengan como endpoint la dirección en la que se ejecuta el bróker.
3. public String getTopicTitle(String referencia): Obtiene el título de un SubscriptionTopic a partir de su referencia.
4. public String getTopicResource(String referencia): Obtiene el recurso asociado a un SubscriptionTopic a partir de su referencia.
5. public String getTopicInteraction(String referencia): Obtiene la interacción asociada a un SubscriptionTopic a partir de su referencia.
6. public List<SubscriptionTopicDetails.FilterDetail> getFilters(String topicUrl, String fhirUrl): Obtiene los detalles de los filtros de un SubscriptionTopic desde un servidor FHIR.
7. public List<String> getSubscriptionTopicIds(String fhirUrl): Obtiene una lista de IDs de SubscriptionTopics desde un servidor FHIR.
8. public void createSubscription(String topicUrl, String payload, List<Filter> filters, String fhirUrl, String endpoint): Crea una nueva suscripción en el servidor FHIR.
9. public void deleteSubscription(String subscriptionId, String fhirUrl): Elimina una suscripción en el servidor FHIR indicado.
10. public String getNotificationResourceId(String json): Obtiene la URL completa del recurso asociado a una notificación a partir de su JSON.

Por último, las plantillas de thymeleaf utilizadas para esta parte son:

- index.html, que muestra un formulario para introducir la IP:Puerto del servidor FHIR al que deseamos conectarnos.
- subscriptions-manager.html, que utiliza las plantillas subscriptions.html, subscriptionsTopics.html y createSubscription.html para mostrar las suscripciones, los tópicos y el formulario para crear suscripciones respectivamente
- subscription-form-html que muestra el formulario para poder modificar los filtros de una suscripción.

A continuación, se muestran las vistas de estas plantillas:

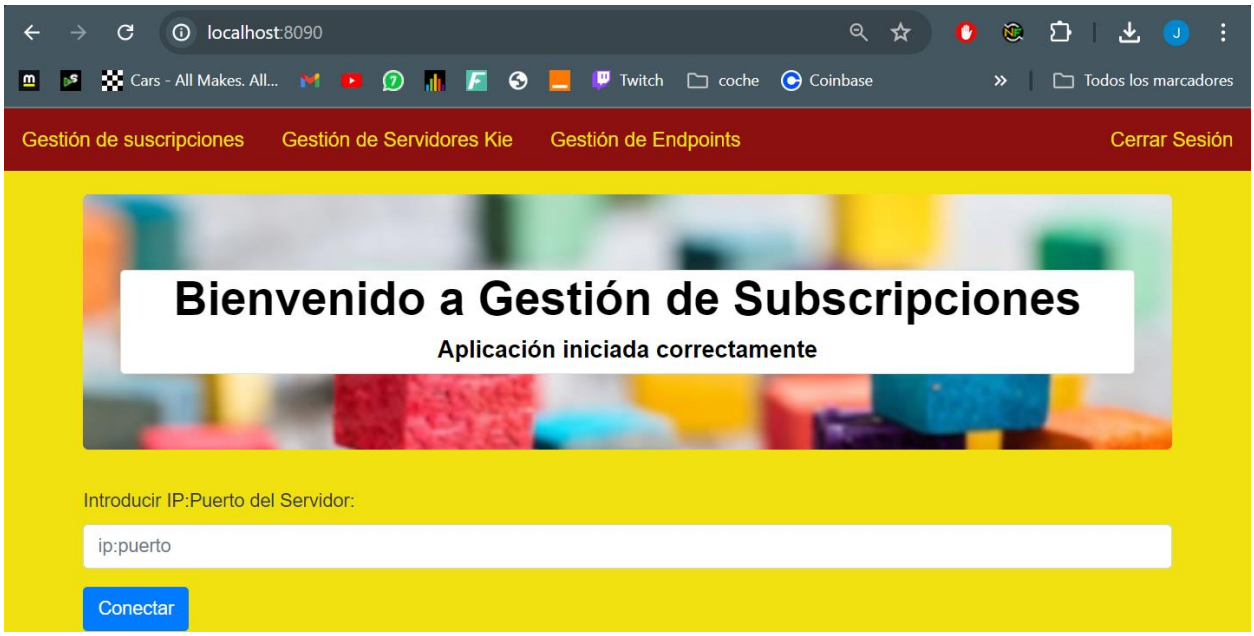

Ilustración 6: Vista de index.html

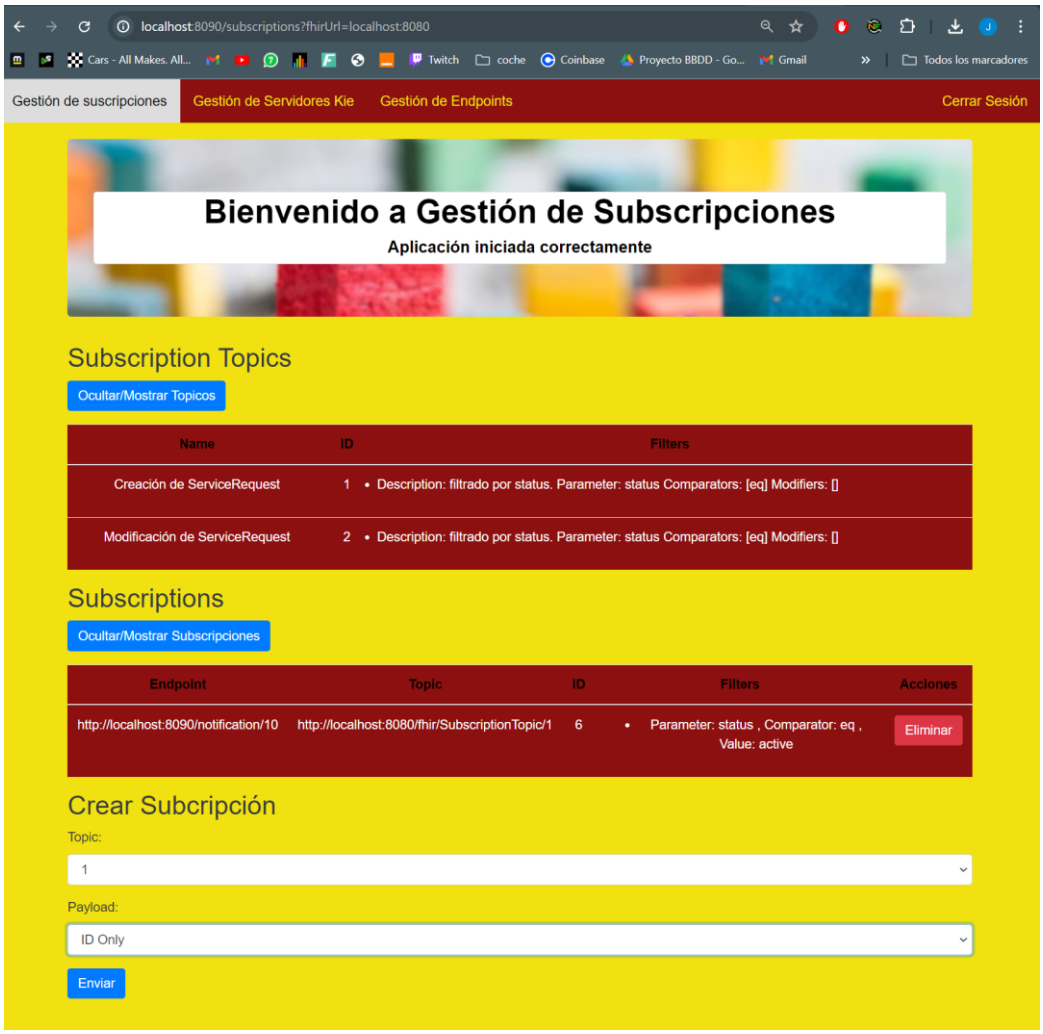

Ilustración 7: Vista de subscriptions-manager.html

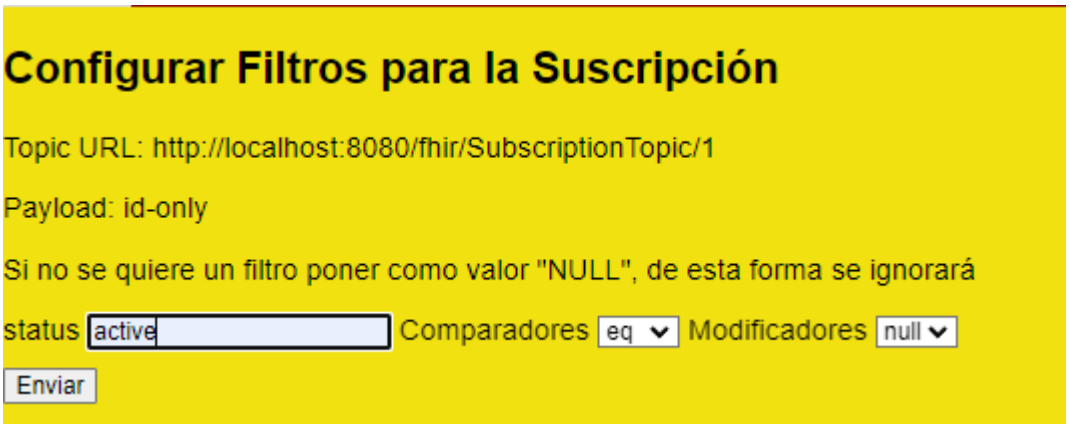

Ilustración 8: Vista de subscription-form.html

3.5.2.3 Código de notificaciones

Para la gestión de notificaciones se ha desarrollado el controlador NotificationController.java. Este controlador maneja las peticiones HTTP enviadas a http://ip_aplicación:puerto/notification. Ofrece un único método:

@PostMapping("/{id}")
public ResponseEntity<String> sendNotification(@PathVariable Long id, @RequestBody String json)

Este método será llamado al recibir una petición HTTP post a http://ip_aplicación:puerto/notification/id. Donde id será el identificador de una entidad NotificationEP. Esta url es el endpoint que se le asigna previamente a una suscripción.

Al recibir la notificación se busca en el repositorio de endpoints la entidad NotificationEP con el id proporcionado y se obtiene la url del recurso por el cual se nos ha notificado, posteriormente se llama al método de KieServerService, sendSignalToAllKieServers(notificationEP, idRecurso) donde se proporciona la entidad del enpoint y el id del recurso por el que se nos ha notificado.

Este método busca todos los servidores KIE almacenados en el repositorio KieServerRepository y envía a cada uno de ellos una señal cuyo nombre extrae del atributo signalName de la entidad NotificationEP y cuyo mensaje será la url del recurso implicado en la notificación.

3.5.3 Inicio de sesión en el bróker

Para poder acceder al bróker se tiene que iniciar sesión. Cuando se inicia la aplicación, al navegar a cualquier url de nuestra aplicación, esta te redirige a la url http://IP:Puerto/login. Donde se deben introducir las credenciales. Esto se realiza en la clase DefaultWebSecurityConfig.java:

@Bean
SecurityFilterChain filterChain(HttpSecurity http) throws Exception: Configura la cadena de filtros de seguridad de Spring Security utilizando HttpSecurity.

Los usuarios se encuentran configurados en:

@Bean
UserDetailsService userDetailsService (BCryptPasswordEncoder encoder): Configura las credenciales de usuario del bróker y sus roles en memoria.

3.6 Pruebas realizadas

A continuación, se muestran detalladamente los pasos a seguir para ejecutar la aplicación y ver su funcionamiento:

3.6.1 Servidor FHIR

Para ejecutar el servidor FHIR, debemos configurarlo en application.yaml para que funcione con la versión de FHIR R5 y para que acepte las suscripciones mediante REST-Hook, este archivo de configuración podemos encontrarlo en el repositorio de Github donde se encuentra subido el trabajo https://github.com/tfg-projects-dit-us/FKBroker/blob/master/Resources/application.yaml. Nos desplazamos al directorio raíz del servidor y ejecutamos mvn spring-boot:run. Una vez arrancado podemos acceder a su interfaz web en https://localhost:8080/fhir, donde se muestra la siguiente interfaz:

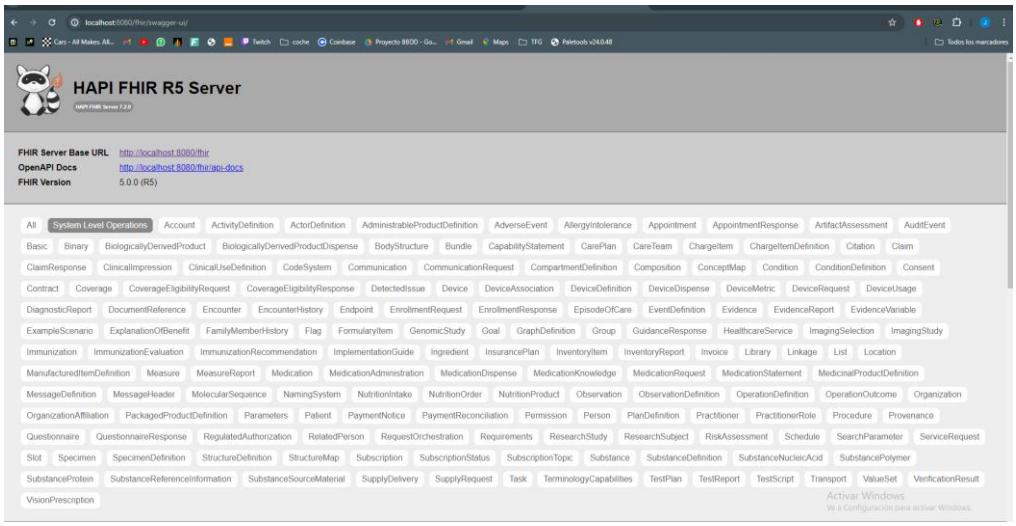

Ilustración 9: Página principal de servidor FHIR.

En esta página se especifica la url del servidor y la versión de FHIR que se utiliza. En la parte inferior se muestra un listado de todos los recursos FHIR disponibles en el servidor.

Con el servidor en funcionamiento procedemos a crear los SubscriptionTopics. Usaremos la herramienta Postman para enviar las peticiones al servidor. Vamos a crear dos tópicos, uno para la creación de ServiceRequest y otro para la modificación de ServiceRequest, añadiremos como filtro el status del recurso.

En el siguiente repositorio de Github [20], podemos encontrar una colección de Postman con todas las peticiones HTTP, estas se encuentran enumeradas en orden de ejecución. Debemos importarla desde la ventana superior izquierda → file → import.

Primero creamos los JSON que se van a enviar:

```
{
  "resourceType": "SubscriptionTopic",
  "id": "1",
  "url": "http://localhost:8080/fhir/SubscriptionTopic/1",
  "title": "Creación de ServiceRequest",
  "status": "active",
  "description": "Creación de ServiceRequest",
  "resourceTrigger": [{
    "description": "Creación de ServiceRequest",
    "resource": "ServiceRequest",
    "supportedInteraction": ["create"]}],
  "canFilterBy": [{
    "description": "filtrado por status.",
    "resource": "ServiceRequest",
    "filterParameter": "status",
    "comparator": ["eq"]
  }]
}
```

Código 2: JSON de un SubscriptionTopic para la creación de un ServiceRequest. [20]

```
{
  "resourceType": "SubscriptionTopic",
  "id": "2",
  "url" : "http://localhost:8080/fhir/SubscriptionTopic/2",
  "title": "Modificación de ServiceRequest",
  "status": "active",
  "description": "Modificación de ServiceRequest",
  "resourceTrigger": [{
    "description": "Update de ServiceRequest",
    "resource": "ServiceRequest",
    "supportedInteraction": ["update"]}],
  "canFilterBy": [{
    "description": "filtrado por status.",
    "resource": "Encounter",
    "filterParameter": "status",
    "comparator": ["eq"]
  }]
}
```

Código 3: JSON de un SubscriptionTopic para la modificación de un ServiceRequest. [20]

En Postman debemos configurar la dirección del servidor y el método HTTP, que en este caso sería una petición POST a http://localhost:8080/fhir/SubscriptionTopic. En la cabecera se debe poner que Content-Type es application/fhir-json y en el cuerpo del mensaje indicamos que es un JSON y ponemos los indicados anteriormente. Una vez enviadas las peticiones se habrán creado los dos tópicos en el servidor. Podemos comprobarlo a través del navegador:

Ilustración 10: Recursos SubscriptionTopic del servidor FHIR.

Con estos pasos ya tendríamos el servidor FHIR configurado.

3.6.2 Servidor KIE

Para instanciar el servidor KIE, podemos abrir Docker y seleccionar en el menú de imágenes la correspondiente al business-central de jBPM. Tenemos que indicar los puertos TCP de escucha de la aplicación, en este caso hemos escogido los puertos 999 y 997:

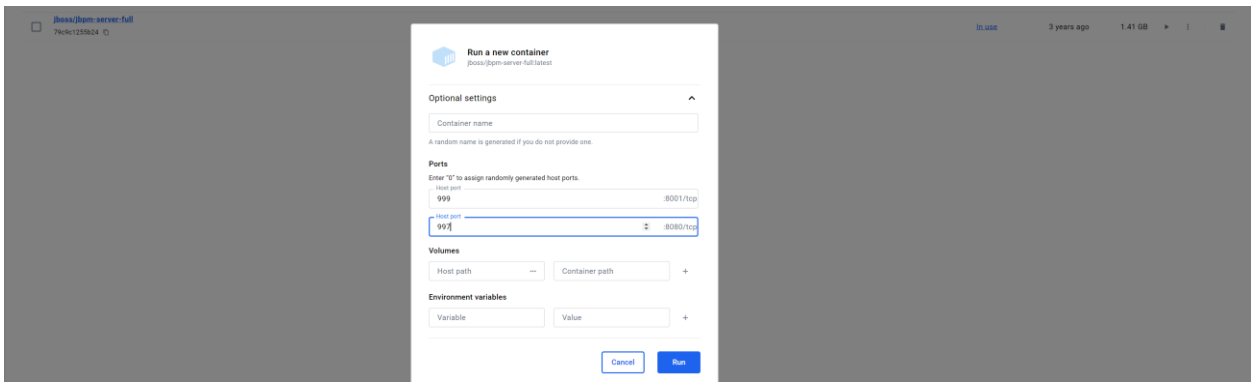

Ilustración 11: Configuración de servidor KIE en Docker.

También podemos iniciarlo directamente desde línea de comandos con el siguiente comando:

"docker run -p 997:8080 -p 999:8001 -d --name jbpm-server-full jboss/jbpm-server-full:latest"

Una vez arrancado el servicio, podemos acceder a business central por el navegador en la url http://localhost:997/business-central, donde se muestra una página para iniciar sesión, debemos introducir usuario: "wbadmin" , contraseña: "wbadmin".

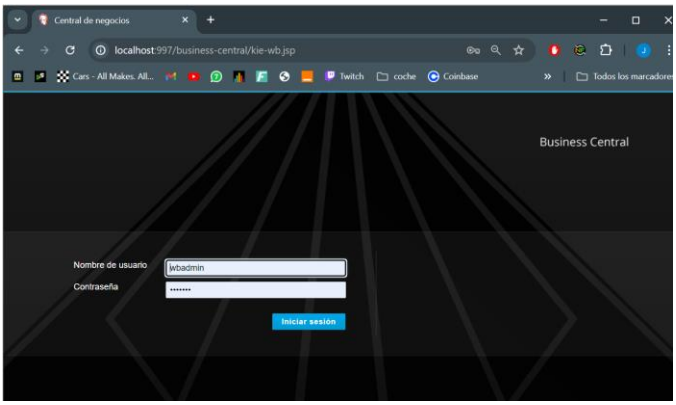

Ilustración 12: Página de login de Business central

Una vez dentro creamos un Proyecto, en este caso lo hemos llamado "Signals".

Es importante indicar que en este ejemplo se van a crear dos endpoints de manera automática al crear las suscripciones, uno para la creación de los ServiceRequest, al cual se le asignará el nombre de señal "create-ServiceRequest" y otro para la modificación de ServiceRequest, al cual se le asignará el nombre de señal "update-ServiceRequest". Por lo que tenemos que crear dos procesos de negocio de forma que se instancien al recibir estas señales, es decir, un proceso se instanciará al recibir una señal cuyo nombre sea "create-ServiceRequest" y el otro al recibir una señal cuyo nombre sea "update-ServiceRequest".

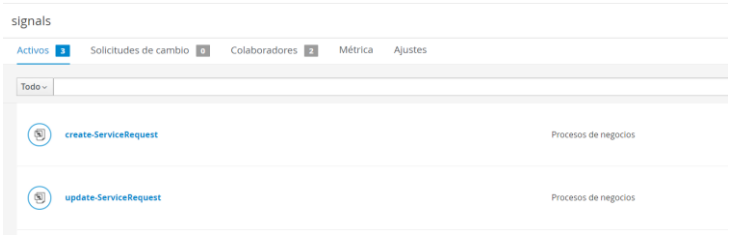

Ilustración 13: Procesos de negocio.

La configuración de los procesos es la siguiente:

- Creamos el modelo del proceso, en este caso, el evento de inicio será de tipo "Señal de inicio", después se realizará una tarea automatizada que imprima una cadena de texto en la consola y finaliza el proceso.

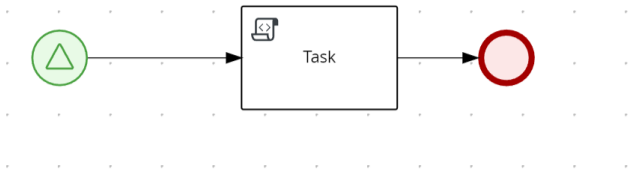

Ilustración 14: Modelo de proceso de negocio

- Creamos una variable del proceso, de tipo String, a la cual llamaremos urlRecurso.
- En la configuración de la señal de inicio indicamos que esta señal será “create-ServiceRequest” en uno de los procesos y “update-ServiceRequest en el otro y le indicamos que el mensaje que reciba se debe almacenar en la variable urlRecurso que hemos creado previamente.
- Configuramos la tarea para que ejecute un script que imprima por pantalla la variable urlRecurso.

Estos procesos de negocio se encuentran en el repositorio de Github https://github.com/tfg-projects-dit-us/FKBroker/tree/master/Resources, se pueden importar directamente desde la ventana de activos del proyecto. También se encuentra el proyecto de business central entero en la carpeta fkbroker-kjar de este repositorio.

Una vez configurados los dos procesos, desde la pantalla principal del Proyecto le damos a implementar y ya tendríamos el servidor KIE funcionando a la espera de recibir señales.

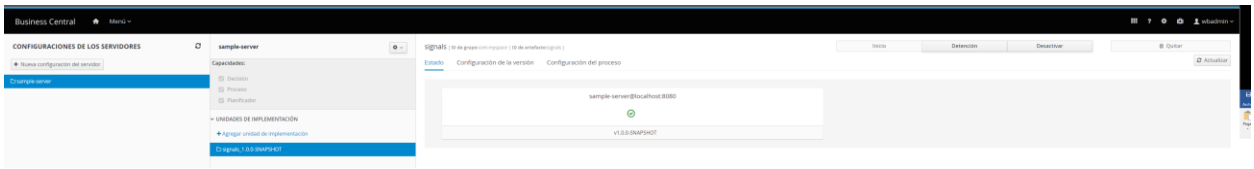

Ilustración 15: Servidor KIE desplegado.

3.6.3 Aplicación Bróker

Para ejecutar nuestra aplicación bróker debemos ir al directorio raíz de nuestra aplicación y ejecutar *“mvn spring-boot:run”*. Una vez iniciada la aplicación podemos acceder a ella mediante el navegador por la url http://localhost:8090, donde se mostrará una página para iniciar sesión, debemos introducir las credenciales usuario: wbadmin y contraseña: wbadmin.

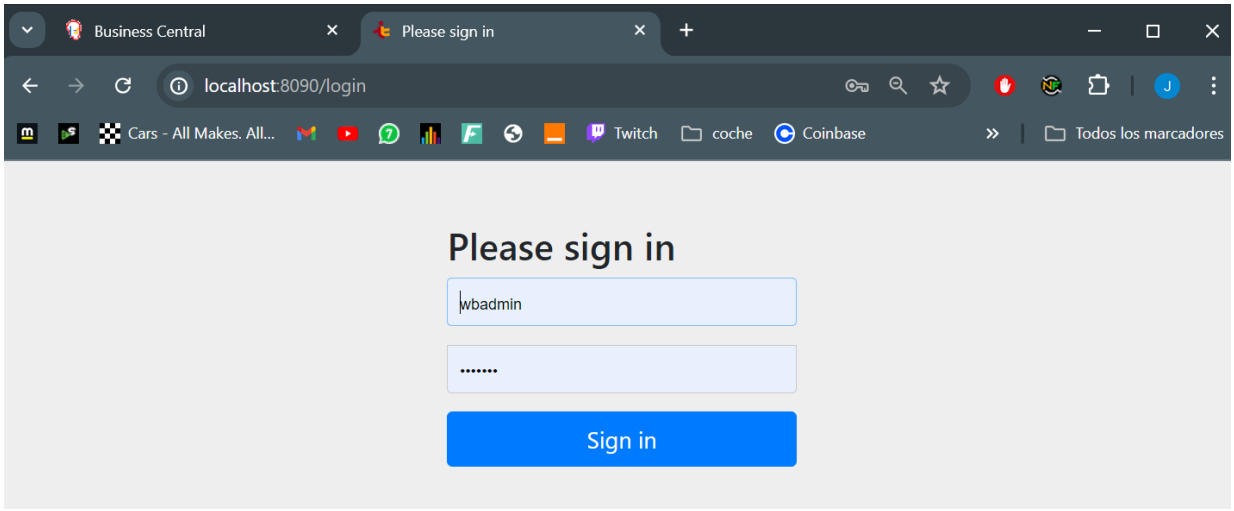

Ilustración 16: Página de inicio de sesión de la aplicación.

Una vez introducidas las credenciales se nos muestra la página principal, donde aparece un formulario para introducir la IP:Puerto del servidor FHIR al que deseamos conectarnos y una barra superior con tres menús, el primero es la página mostrada, el Segundo para gestionar los servidores KIE y el tercero para gestionar los endpoints:

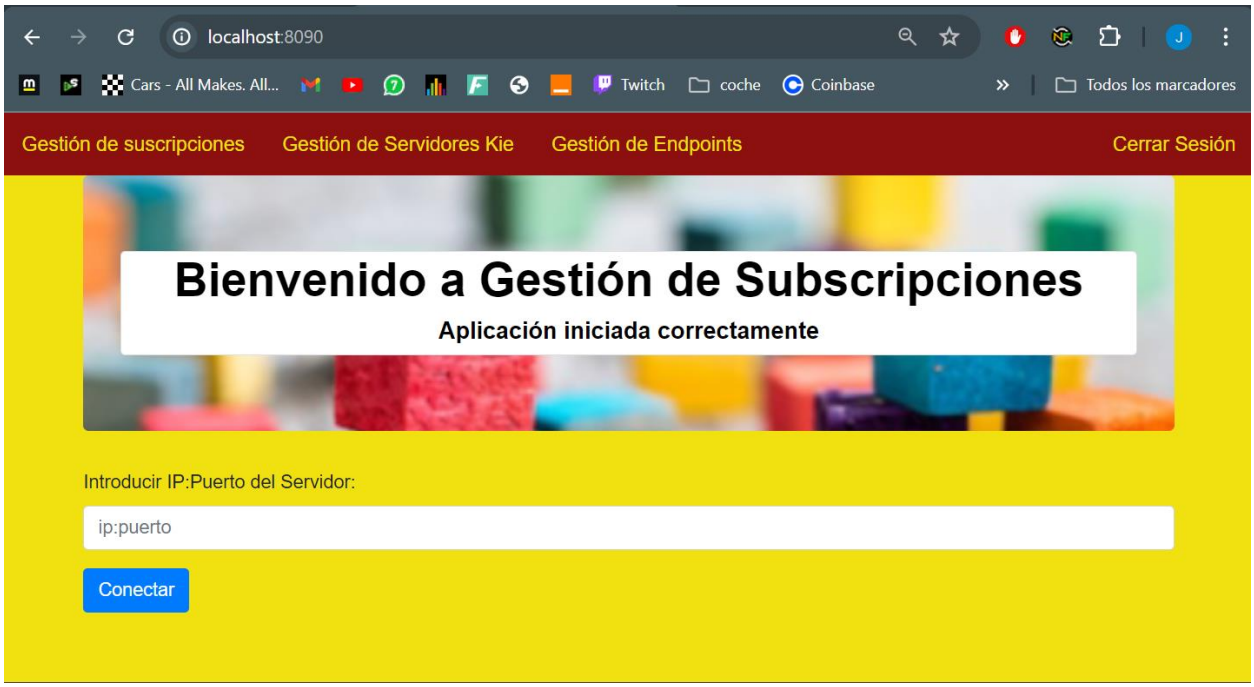

Ilustración 17: Página principal del bróker.

Antes de gestionar las suscripciones, vamos a añadir en el listado de servidores KIE el que acabamos de instanciar con Docker, para ello nos vamos al menú "Gestión de Servidores KIE" y se mostrará la siguiente página:

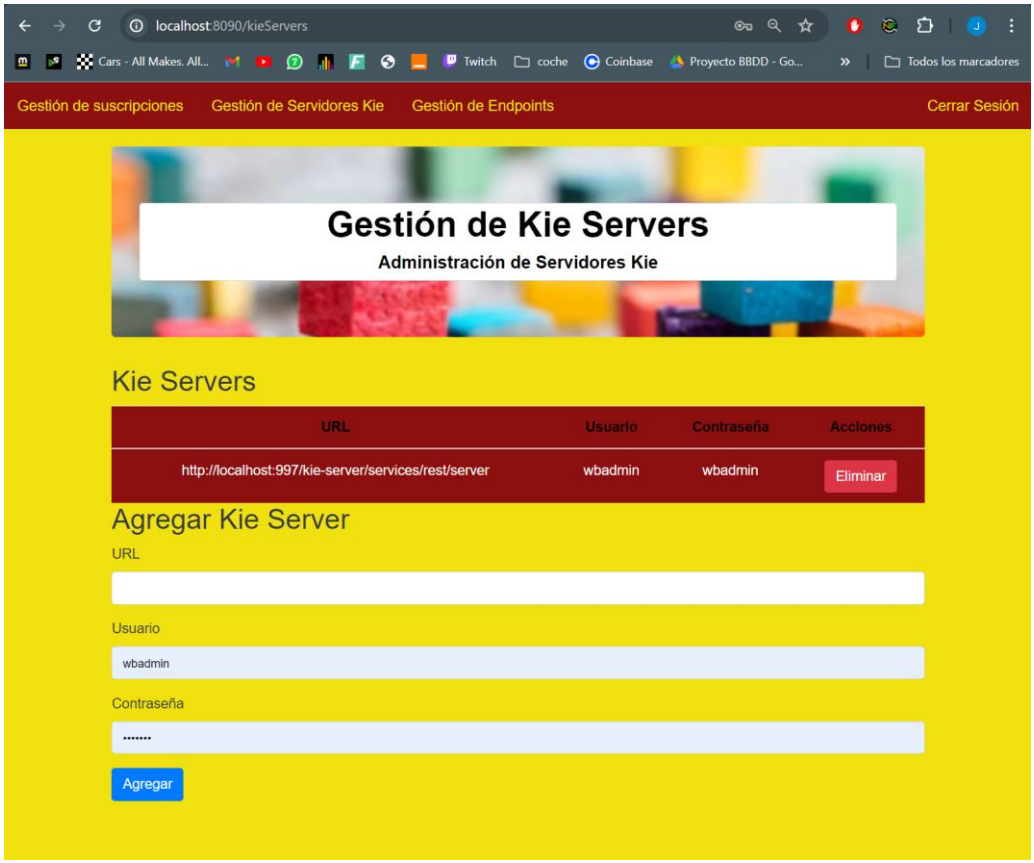

Ilustración 18: Gestión de servidores KIE.

En esta página vemos un listado de servidores KIE, que en un principio se mostrará vacía, y un formulario para añadir servidores KIE, donde debemos introducir la url donde se deben enviar las señales y las credenciales del servidor, al pulsar el botón de agregar se creará una nueva instancia KieServer y se mostrará en la lista. En este caso hemos agregado uno, pero podrían añadirse tantos como quisiéramos.

Una vez añadido el servidor KIE volvemos al menú "Gestión de suscripciones", donde se mostraba el formulario y lo rellenamos con la IP:Puerto de nuestro servidor FHIR, en este caso es localhost:8080.

Ilustración 19: Conexión a servidor FHIR.

Cuando pulsemos conectar se nos mostrará una página donde podemos ver tres campos: Un listado de SubscriptionTopics, donde se indica el nombre, el id y los filtros disponibles; un listado de suscripciones, donde se indica el endpoint, el tópico, el id de la suscripción y los valores de los filtros configurados; y, por último, un formulario para crear suscripciones, donde se debe indicar el id del tópico al que deseamos suscribirnos:

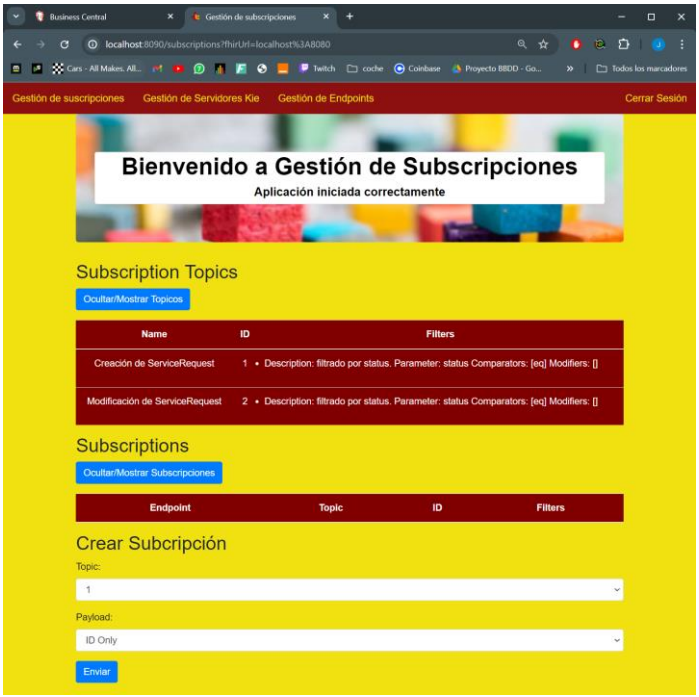

Ilustración 20: Página de gestión de suscripciones.

A continuación, vamos a crear una suscripción para cada tópico, para ello debemos seleccionar el id del tópico y pulsar en "enviar". Esto nos llevará a un Segundo formulario, donde se muestran los filtros disponibles para que podamos configurarlos. En caso de que no quisiéramos configurar un filtro en concreto tenemos que darle el valor "NULL".

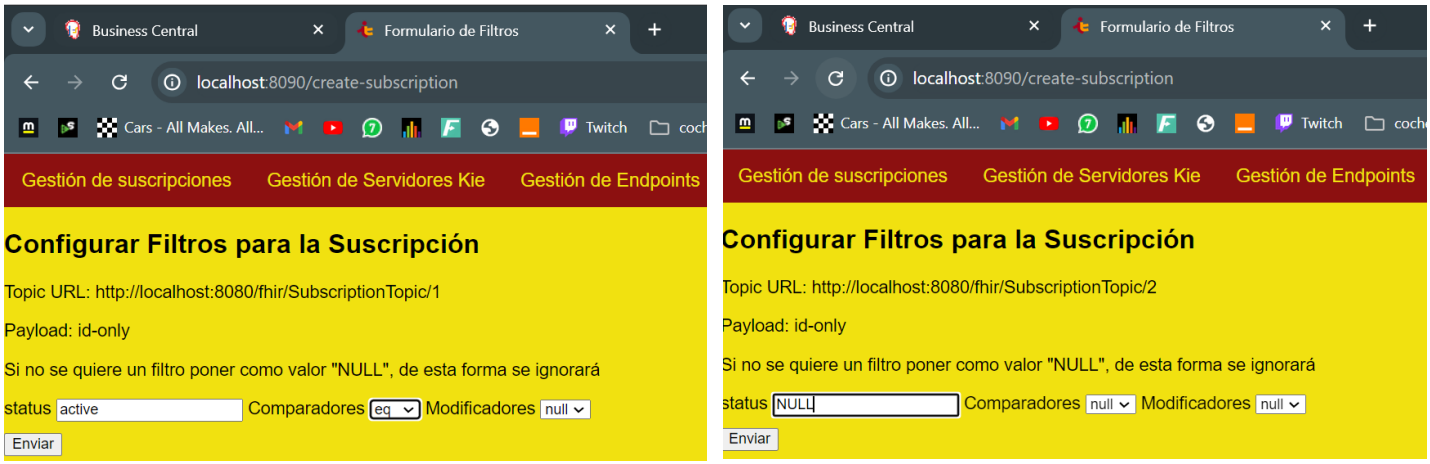

Ilustración 21: Configuración de filtros.

Una vez creadas las suscripciones se nos redirige a la página de gestión de suscripciones, donde podemos ver que se han creado las dos, también podemos comprobar que en la suscripción al tópico 2, al indicar el valor del filtro como NULL, este no se ha añadido.

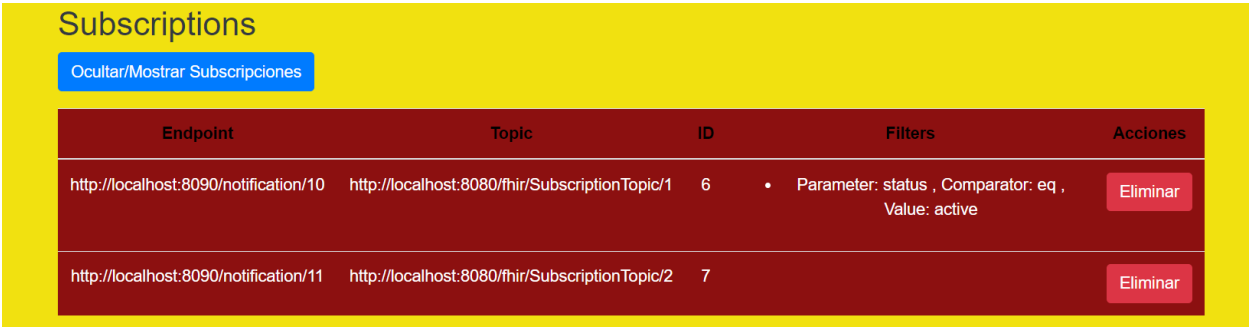

Ilustración 22: Suscripciones creadas.

Podemos apreciar en la imagen anterior que los endpoints son diferentes, esto se debe a lo explicado anteriormente, se ha creado un endpoint para create-ServiceRequest (/notification/10) y otro para update-ServiceRequest (/notification/11). Estos valores se corresponden con el id de la entidad NotificationEP creada, no empiezan en 1 porque se han estado hacienda pruebas anteriores y los IDs se asignan de manera lineal. Podemos comprobar que estos endpoints se han generado correctamente accediendo al menú “Gestión de endpoints”:

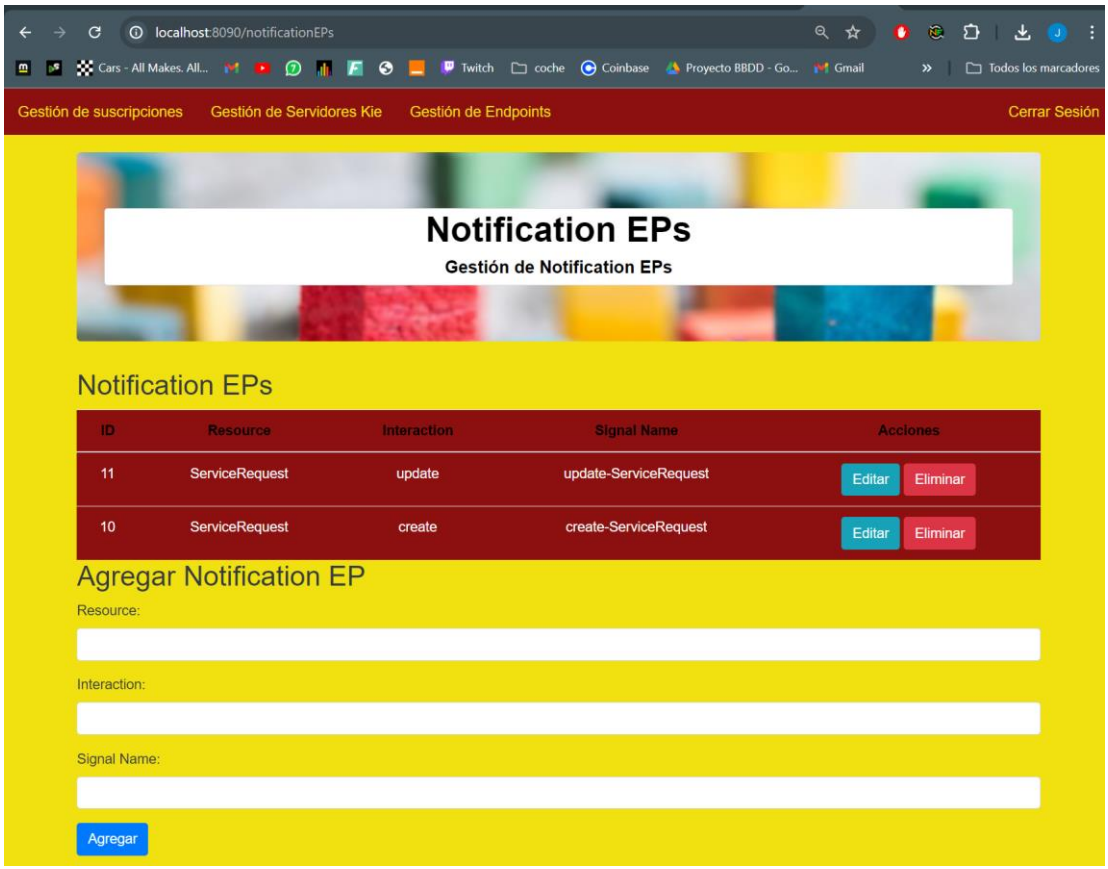

Ilustración 23: Gestión de endpoints

Podemos ver que existe un formulario para generar endpoints personalizados, por ejemplo, si quisiéramos que el nombre de la señal que se envía al crear un Paciente fuera "nuevo paciente" tendríamos que rellenar el formulario indicando que el recurso sería "Patient", la interacción "create" y el Signal Name "nuevo paciente". De esta forma, al crear una suscripción a un tópico que notificase de la creación de pacientes, se le asignaría el endpoint creado, en este caso tendría id 12 y el endpoint sería "http://localhost:8090/notification/12" y al recibir la notificación no enviaría la señal con nombre "create-Patient" si no como "nuevo paciente".

En el botón editar, podemos cambiar el nombre de la señal que se enviaría a los servidores KIE:

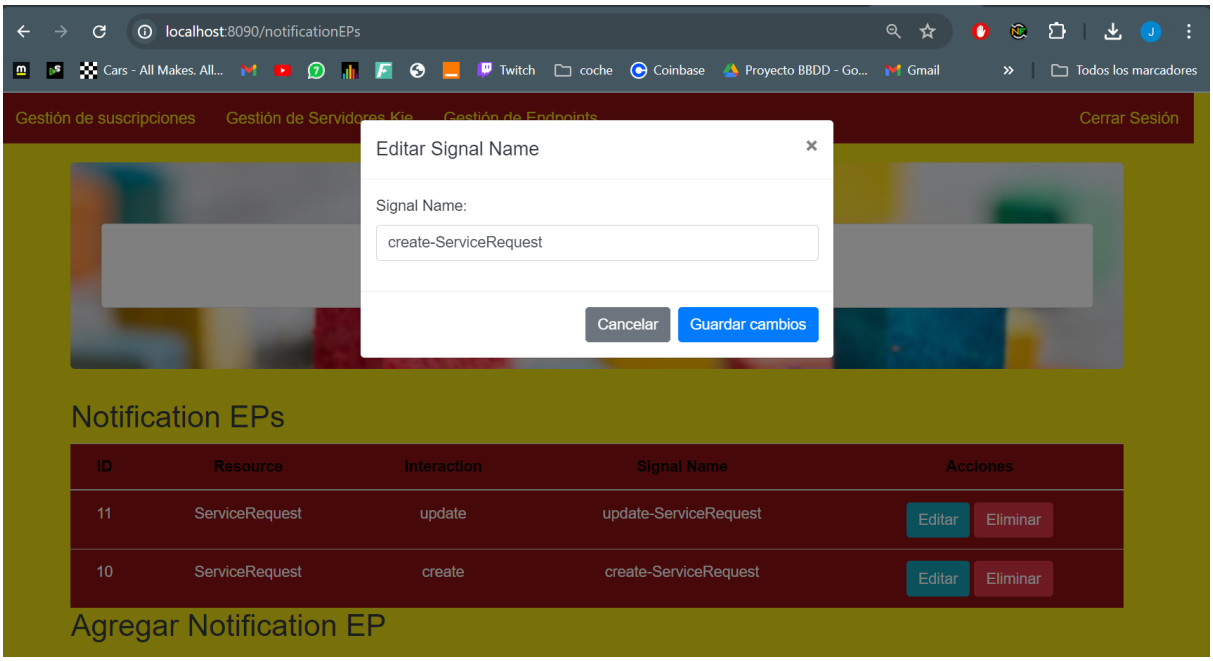

Ilustración 24: Edición de Signal Name

Con esto tendríamos configurado el bróker para recibir notificaciones en los endpoints creados para posteriormente enviar señales a los servidores KIE configurados.

3.6.4 Test

Por último, para comprobar el funcionamiento de la aplicación debemos comprobar que al crear y modificar un recurso ServiceRequest se reciben las notificaciones y se envían las señales al servidor KIE. Para ello, vamos a crear un ServiceRequest y comprobaremos en la aplicación que se recibe y se envía la señal. Esta comprobación la haremos por logs generados por la aplicación.

Creamos un ServiceRequest en el servidor FHIR a través de Postman, debemos enviar un JSON con la información del recurso a http://localhost:8080/fhir/ServiceRequest. Estas peticiones también se encuentran incluidas en la colección de Postman subida a Github.

```
{
  "resourceType": "ServiceRequest",
  "id": "5",
  "status": "active",
  "intent": "order",
  "code": {
    "coding": [
      {
        "system": "http://loinc.org",
        "code": "12345-6",
        "display": "Example Service Request"
      }
    ]
  }
}
```

Código 4: JSON de un ServiceRequest. [20]

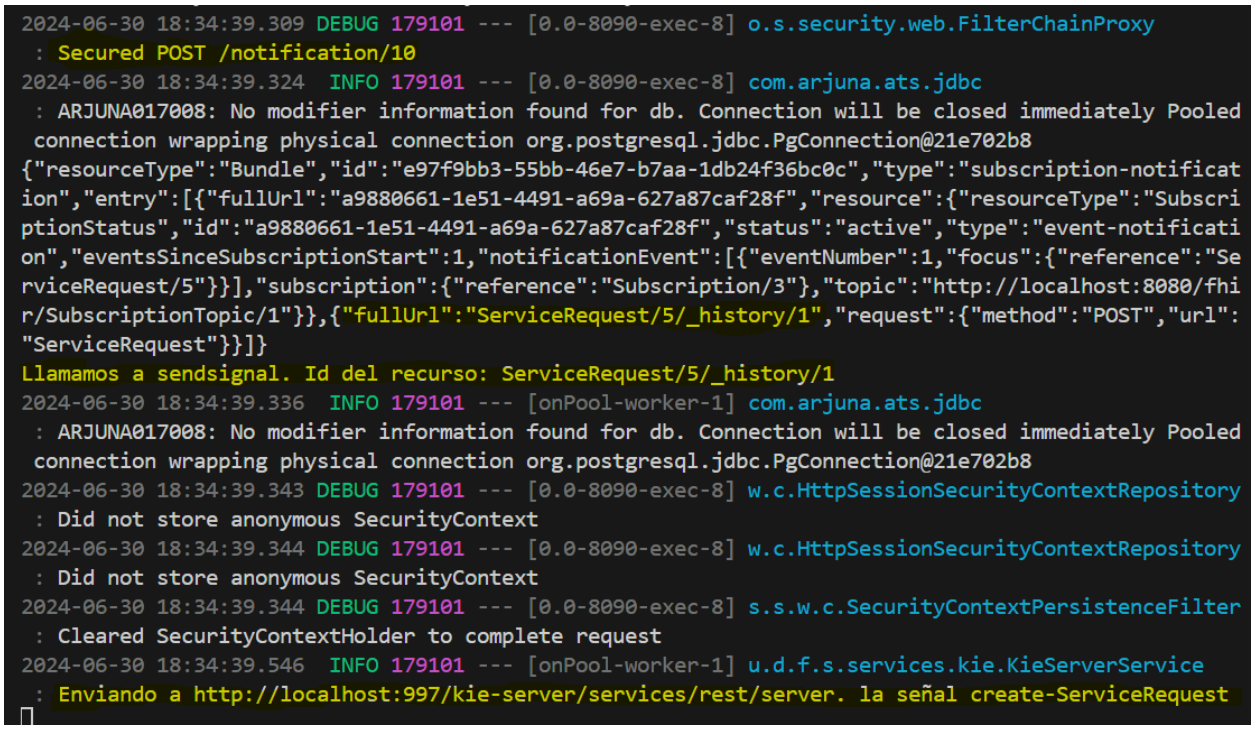

Ilustración 25: Mensajes de la consola del bróker al recibir una notificación

Podemos comprobar que se ha recibido un post en "/notification/10" el cual era el endpoint asignado a la creación de un ServiceRequest, se muestra el JSON y podemos comprobar que hemos obtenido la url del recurso, la cual se incluía en la notificación. Posteriormente vemos como se envía la señal "create-ServiceRequest al servidor KIE que añadimos previamente.

Desde la interfaz web del servidor KIE podemos ver las instancias de proceso y comprobar que se ha creado una instancia del proceso "create-ServiceRequest".

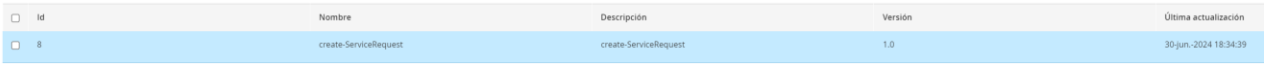

Ilustración 26: Instancias de procesos

Haciendo clic en la instancia podemos ver información del proceso, donde comprobaremos que la url del recurso se ha guardado satisfactoriamente:

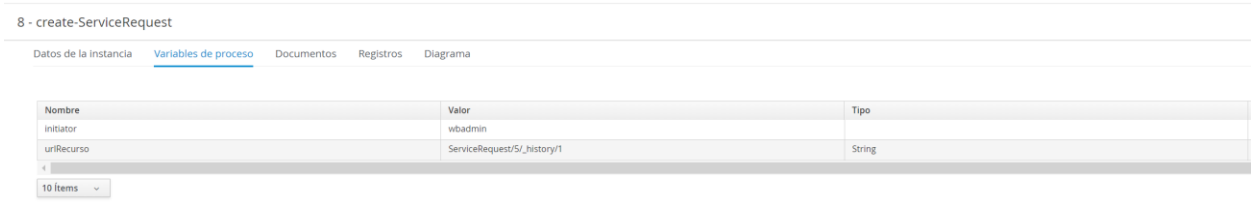

Ilustración 27: Variables del proceso create-ServiceRequest

Ahora procedemos a modificar el recurso ServiceRequest creado anteriormente para comprobar que se envía la otra señal, para ello enviaremos una petición PUT a http://localhost:8080/fhir/ServiceRequest/5 enviando en el cuerpo del mensaje el JSON pero con alguna variación, por ejemplo cambiaremos el estado del recurso de "active" a "draft":

```
{
  "resourceType": "ServiceRequest",
  "id": "5",
  "status": "draft",
  "intent": "order",
  "code": {
    "coding": [
      {
        "system": "http://loinc.org",
        "code": "12345-6",
        "display": "Example Service Request"
      }
    ]
  }
}
```

Código 5: JSON de un ServiceRequest. [20]

Una vez enviada la petición comprobamos nuevamente la consola y vemos que se recibe un POST en "/notification/11", se puede apreciar que en este caso el id es "ServiceRequest/5/_history/2" ya que se ha modificado el recurso y este cambia de versión:

```
2024-06-30 18:47:14.139 DEBUG 179101 --- [.0-8090-exec-10] o.s.security.web.FilterChainProxy
 : Secured POST /notification/11
2024-06-30 18:47:14.141  INFO 179101 --- [.0-8090-exec-10] com.arjuna.ats.jdbc
 : ARJUNA017008: No modifier information found for db. Connection will be closed immediately Pooled
 connection wrapping physical connection org.postgresql.jdbc.PgConnection@21e702b8
{"resourceType":"Bundle","id":"5bff6a12-6c49-4a0a-88ce-b9a8fae92c2b","type":"subscription-notificat
ion","entry":[{"fullUrl":"bef85830-902b-4dbd-a31f-b3a1d5e28001","resource":{"resourceType":"Subscri
ptionStatus","id":"bef85830-902b-4dbd-a31f-b3a1d5e28001","status":"active","type":"event-notificati
on","eventsSinceSubscriptionStart":1,"notificationEvent":[{"eventNumber":1,"focus":{"reference":"Se
rviceRequest/5"}}],"subscription":{"reference":"Subscription/4"},"topic":"http://localhost:8080/fhi
r/SubscriptionTopic/2"}},{"fullUrl":"ServiceRequest/5/_history/2","request":{"method":"PUT","url":"
ServiceRequest/5"}}]}
Llamamos a sendsignal. Id del recurso: ServiceRequest/5/_history/2
2024-06-30 18:47:14.144  INFO 179101 --- [onPool-worker-2] com.arjuna.ats.jdbc
 : ARJUNA017008: No modifier information found for db. Connection will be closed immediately Pooled
 connection wrapping physical connection org.postgresql.jdbc.PgConnection@21e702b8
2024-06-30 18:47:14.144 DEBUG 179101 --- [.0-8090-exec-10] w.c.HttpSessionSecurityContextRepository
 : Did not store anonymous SecurityContext
2024-06-30 18:47:14.145 DEBUG 179101 --- [.0-8090-exec-10] w.c.HttpSessionSecurityContextRepository
 : Did not store anonymous SecurityContext
2024-06-30 18:47:14.145 DEBUG 179101 --- [.0-8090-exec-10] s.s.w.c.SecurityContextPersistenceFilter
 : Cleared SecurityContextHolder to complete request
2024-06-30 18:47:14.170  INFO 179101 --- [onPool-worker-2] u.d.f.s.services.kie.KieServerService
 : Enviando a http://localhost:997/kie-server/services/rest/server. la señal update-ServiceRequest
```

Ilustración 28: Mensajes de la consola del bróker al recibir una notificación

Por último, comprobamos que se ha creado la instancia del proceso "update-ServiceRequest" en nuestro servidor KIE, donde el valor de nuestra variable del proceso será "ServiceRequest/5/_history/2":

Id	Nombre	Descripción	Versión	Última actualización
9	update-ServiceRequest	update-ServiceRequest	1.0	30-jun.-2024 18:47:14

Ilustración 29: Instancias de procesos

4 CONCLUSIONES Y LÍNEAS FUTURAS

4.1 Resumen de los logros y resultados del trabajo

El presente trabajo ha permitido el diseño e implementación de un bróker que facilita la comunicación indirecta entre servidores FHIR y servidores KIE, usando tecnologías como Spring Framework y siguiendo el patrón Modelo-Vista-Controlador. A lo largo del desarrollo del proyecto, se lograron los siguientes objetivos:

1. Integración de Tecnologías: Se logró integrar exitosamente FHIR con KIE utilizando un bróker intermedio. Esta integración permite manejar suscripciones en servidores FHIR y enviar notificaciones a servidores KIE mediante señales.
2. Automatización de Procesos: La solución desarrollada permite automatizar la recepción y procesamiento de eventos de suscripción en un servidor FHIR en tiempo real, lo que optimiza la gestión de procesos y mejora la eficiencia operativa en entornos de salud.
3. Uso de Estándares: La utilización de estándares como FHIR para la interoperabilidad y BPMN para el modelado de procesos asegura que la solución sea compatible con otras aplicaciones y sistemas, permitiendo que esta sea reutilizable y escalable.
4. Pruebas y Validación: Mediante el uso de herramientas como Postman y la implementación de pruebas exhaustivas, se garantizó el correcto funcionamiento del bróker y su capacidad para gestionar suscripciones y notificaciones de manera eficaz.

4.2 Líneas Futuras

A pesar de los logros alcanzados, el proyecto abre varias oportunidades de mejora y expansión en el futuro:

1. Mejoras en la Seguridad: Aumentar la seguridad en la transmisión de datos y en la autenticación de usuarios es crucial. Implementar mecanismos avanzados de cifrado y autenticación multifactorial puede proporcionar una mayor protección a la información manejada.
2. Integrar un servidor de identidad: Actualmente los usuarios se crean en memoria al iniciar el servidor, es necesario para garantizar una mayor seguridad que estos usuarios se encuentren en un servidor externo y este sea consultado por el bróker cuando un usuario intente iniciar sesión.
3. Optimización del Rendimiento: Aunque la solución es funcional, optimizar el rendimiento del bróker para manejar un mayor volumen de solicitudes y notificaciones es fundamental.
4. Procesamiento de SubscriptionStatus más completo: Actualmente el bróker procesa una notificación asumiendo que esta es de tipo “notification-event”. Extender la gestión de los notification-bundles recibidos en los endpoints para atender otro tipo de notificaciones, como “handshake” o “hearthbeat”.

REFERENCIAS

[1] IBM, «Business Process Management,» [En línea]. Available: https://www.ibm.com/topics/business-process-management. [Último acceso: 02 06 2024].

[2] HL7, «FHIR Release 5,» [En línea]. Available: https://hl7.org/fhir/R5/modules.html. [Último acceso: 02 06 2024].

[3] HL7, «FHIR SubscriptionTopic resource,» [En línea]. Available: https://hl7.org/fhir/R5/subscriptiontopic.html. [Último acceso: 01 07 2024].

[4] HL7, «jBPM Overview,» [En línea]. Available: https://docs.jbpm.org/7.74.1.Final/jbpm-docs/html_single/#_jbpmoverview. [Último acceso: 02 06 2024].

[5] HL7, «FHIR Subscriptions,» [En línea]. Available: https://hl7.org/fhir/R5/subscriptions.html. [Último acceso: 02 06 2024].

[6] HL7, «FHIR Subscription resource,» [En línea]. Available: https://hl7.org/fhir/R5/subscription.html. [Último acceso: 30 06 2024].

[7] HL7, «FHIR Subscription notification,» [En línea]. Available: https://hl7.org/fhir/R5/bundle.html#subscription-notification. [Último acceso: 30 06 2024].

[8] HL7, «FHIR SubscriptionStatus resource,» [En línea]. Available: https://hl7.org/fhir/R5/subscriptionstatus.html. [Último acceso: 01 07 2024].

[9] Red Hat, «KIE Execution Server,» [En línea]. Available: https://docs.jbpm.org/7.74.1.Final/jbpm-docs/html_single/#_ch.kie.server. [Último acceso: 02 07 2024].

[10] Red Hat, «What is BPMN 2.0,» [En línea]. Available: https://docs.jbpm.org/7.74.1.Final/jbpm-docs/html_single/#_what_is_bpmn_2_0. [Último acceso: 02 06 2024].

[11] Red Hat, «Business Central Overview,» [En línea]. Available: https://docs.jboss.org/jbpm/release/7.55.0.Final/jbpm-docs/html_single/#_business_central. [Último acceso: 02 07 2024].

[12] VMware, «Spring Framework Overview,» [En línea]. Available: https://docs.spring.io/spring-framework/reference/overview.html. [Último acceso: 15 06 2024].

[13] Postman, «Postman Overview,» [En línea]. Available: https://learning.postman.com/docs/introduction/overview/. [Último acceso: 03 07 2024].

[14] Docker, «Docker manuals,» [En línea]. Available: https://docs.docker.com/manuals/. [Último acceso: 03 07 2024].

[15] HAPI FHIR, «Getting Started with HAPI FHIR,» [En línea]. Available: https://hapifhir.io/hapi-fhir/docs/getting_started/introduction.html. [Último acceso: 26 06 2024].

[16] IBM, «JPA,» [En línea]. Available: https://www.ibm.com/docs/en/was-liberty/nd?topic=liberty-java-persistence-api-jpa. [Último acceso: 03 07 2024].

[17] Thymeleaf, «Thymeleaf,» [En línea]. Available: https://www.thymeleaf.org/. [Último acceso: 02 07 2024].

[18] Mozilla, «MVC - Glosario,» [En línea]. Available: https://developer.mozilla.org/es/docs/Glossary/MVC. [Último acceso: 30 06 2024].

[19] HAPI FHIR, «HAPI FHIR JPA Server API doc,» [En línea]. Available: https://hapifhir.io/hapi-fhir/apidocs/hapi-fhir-jpaserver-subscription/index.html. [Último acceso: 02 07 2024].

[20] J. M. B. M. Mora, «Github, colección de request HTTP,» 2024. [En línea]. Available: https://github.com/tfg-projects-dit-us/FKBroker/blob/master/Resources/Solicitudes%20servidor%20FHIR.postman_collection.json.

GLOSARIO

API: Aplicaction Programan Interface

BPM: Business Process Management

BPMN: Business Process Model and Notation

FHIR: Fast Healthcare Interoperability Resources

HAPI: HL7 Application Programming Interface

HTTP: Hypertext Transfer Protocol

jBPM: Java Business Process Management

JPA: Java Persistance API

KIE: Knowledge is Everything

MVC: Model-View-Controller

URL: Uniform Resource Locator