Trabajo Fin de Grado
en Ingeniería de las Tecnologías de Telecomunicación

Aplicación web para la gestión de tareas sanitarias en procesos BPMN

Autor: Juan Manuel Ostos Rabadán
Tutora: Dra. Isabel Román Martínez

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2025

Trabajo Fin de Grado
en Ingeniería de las Tecnologías de Telecomunicación

Aplicación web para la gestión de tareas sanitarias en procesos BPMN

Autor:
Juan Manuel Ostos Rabadán

Tutora:
Dra. Isabel Román Martínez
Profesora colaboradora

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2025

Trabajo Fin de Grado: Aplicación web para la gestión de tareas sanitarias en procesos BPMN

Autor: Juan Manuel Ostos Rabadán

Tutora: Dra. Isabel Román Martínez

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2025

El Secretario del Tribunal

A mi familia

A mi pareja

A mis amigos

A mi tutora

A mis maestros

A mis compañeros

Agradecimientos

Este apartado va dedicado a todas aquellas personas que me han ayudado tanto moral como intelectualmente. No solo en el proceso de realizar este trabajo de fin de grado, si no a lo largo de todo mi periodo como estudiante de grado en ingeniería telecomunicaciones.

En primer lugar, me gustaría agradecer a mi familia. En especial a mis padres por confiar y apoyarme siempre en lograr mis objetivos. Además de ofrecerme una buena educación, no solo académica sino en todos los aspectos.

También he de destacar a mi pareja por ser mi mayor apoyo, por estar siempre a mi lado incondicionalmente, por su paciencia, comprensión y por animarme en los momentos más difíciles. Eres la mejor, te quiero!!

A mis amigos de toda la vida por las quedadas y los buenos momentos.

A mi tutora del tfg por darme la oportunidad de trabajar con ella y guiarme en este proceso tan complicado, ya que no me resultó fácil al comienzo asimilar los conceptos del proyecto. Y en lo personal por darme una oportunidad para desarrollarme profesionalmente recomendándome para una empresa.

A mis profesores de la universidad por enseñarme, aunque no siempre lo pongan fácil.

A mis compañeros de grado, por el trabajo en equipo realizado en muchos proyectos a lo largo de la carrera y el gran sacrificio y esfuerzo que hemos dedicado estos años.

Resumen

El objetivo principal de este proyecto es mejorar las funcionalidades de un servicio de gestión de tareas sanitarias conforme al paradigma de gestión de procesos de negocio (BPM, Business Process Management). Este desarrollo es continuación de un TFG de partida. Y servirá asimismo para futuras versiones que amplíen opciones y funcionalidades desarrolladas y por desarrollar en este TFG.

La solución ha sido diseñada en el entorno jBPM y Spring usando las librerías de ambas tecnologías, además se ha usado Thymeleaf para la gestión de las plantillas de la aplicación, así como la librería HapiFHIR para la gestión de recursos sanitarios.

Abstract

The main objective of this project is to upgrade the functionalities of a healthcare task management service that adheres to the Business Process Management (BPM) paradigm. This is an ongoing development of a previous undergraduate project (TFG). In the same way, this service will serve as a foundational platform for versions that will extend the options and functionalities developed in this project and introduce new ones.

The solution has been designed using the jBPM and Spring environments, utilizing the libraries of both technologies. Additionally, Thymeleaf has been employed for template management within the application, and the FHIR library has been used to handle healthcare-related tasks.

Índice

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

Este trabajo de fin de grado tiene como objetivo continuar con el desarrollo de un servicio para gestionar instancias de tareas humanas [3] definidas en modelos de procesos aplicados al ámbito sanitario. Este servicio permitirá obtener, reclamar y completar dichas tareas mediante la presentación de cuestionarios y la recopilación de sus respuestas. La solución es independiente de los procesos específicos, lo que posibilita la gestión de tareas humanas pertenecientes a distintos motores de procesos. Además, se establece una vinculación entre las tareas humanas del proceso jBPM y los recursos FHIR (Task, Questionnaire y QuestionnaireResponse), lo que favorece la integración del sistema y facilita el seguimiento y análisis de las tareas humanas en el entorno de una organización sanitaria.

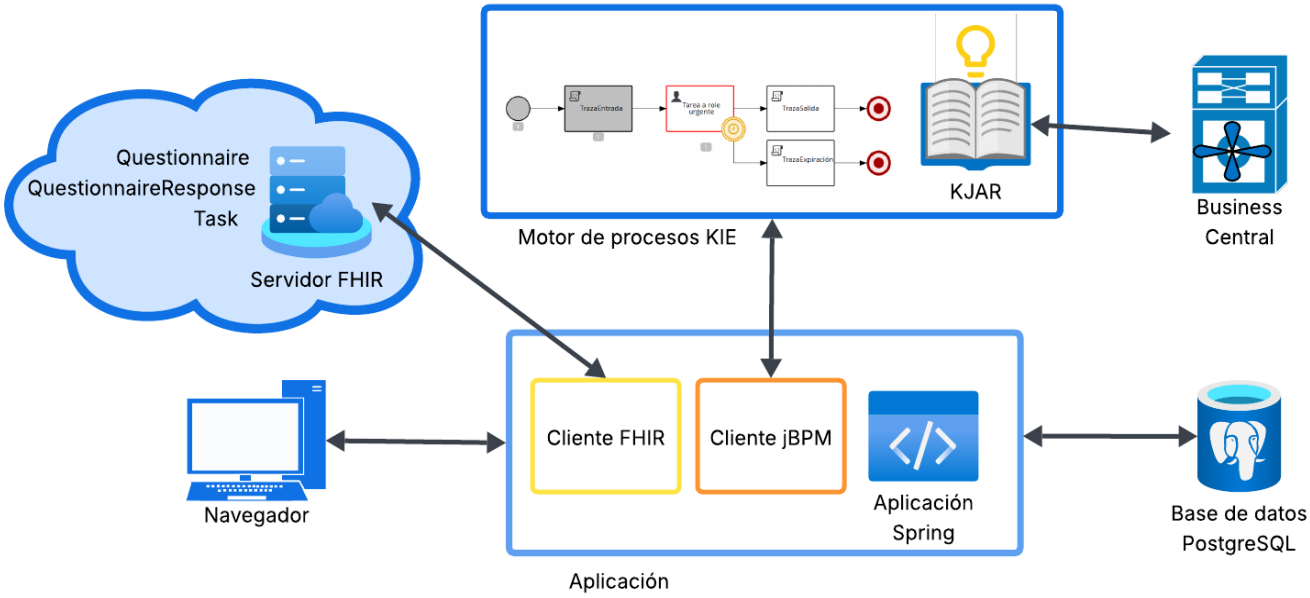

Ilustración 1. Estructura del servicio

1.1 Punto de partida

La solución de partida fue desarrollada por Marco Antonio Maldonado Orozco en su TFG. En él se propone una arquitectura de sistema basada en una aplicación Spring, así como la comunicación con el motor de procesos jBPM [26] y el servidor FHIR [25] para el manejo de recursos como tareas y cuestionarios que son necesarios para el desempeño de la aplicación. Para lograr este objetivo se tomaron varias decisiones estratégicas:

- **Aplicación independiente de los procesos:** La aplicación se desarrolló de tal manera que no necesita una entrada predeterminada ni está sujeta a ningún proceso concreto, es una aplicación que sirve para la gestión de tareas humanas definidas en cualquier proceso.

- **Conexión con los datos de la organización sanitaria:** Las instancias de las tareas humanas de los procesos se vinculan a un recurso Task de FHIR [11]. Para ello, se añade como parámetro de entrada a la tarea del proceso jBPM el id del recurso Task FHIR asociado. Esta vinculación debe ser unívoca, es decir, para cada tarea jBPM asignar una uri de tarea FHIR distinta. A su vez cada recurso Task de FHIR está vinculado a un recurso Questionnaire de FHIR [12], de tal forma que dentro del input del recurso Task hay una entrada cuyo nombre es “ClosingQuestionnaire” que

tiene como valor el id de recurso Questionnaire correspondiente, el cuál puede estar referenciado en varias tareas FHIR. A su vez, cuando la tarea se finalice se generará un recurso QuestionnaireResponse de FHIR [13] cuyo id se añadirá al recurso Task dentro del campo output. En este caso el id de respuesta sólo puede ser referenciado en la tarea FHIR correspondiente. Estas relaciones facilitan la integración con la información de la organización sanitaria.

- **Uso del marco de trabajo KIE:** Se ha seleccionado la suite de productos de KIE [27], que incluye jBPM. Este marco de trabajo proporciona las herramientas necesarias para gestionar procesos de negocio y tareas sanitarias de forma eficiente.

- **Vista generada dinámicamente:** Ofrece la creación y adaptación de plantillas genéricas para la representación automatizada de cuestionarios [12] FHIR como formularios usando Thymeleaf.

- **Gestión básica de tareas humanas:** El proyecto previo se centraba en el proceso principal de gestión de tareas humanas, esto incluía la asignación de tareas potencialmente asignables por el usuario, así como las acciones de comenzar, continuar y rechazar para tareas ya asignadas.

1.2 Objetivos

Se seguirá en la línea marcada por la versión anterior del proyecto. Es decir, usando tanto las tecnologías ya implementadas, como cumpliendo con la filosofía y los objetivos marcados anteriormente en el apartado previo. Dicho esto, se añadirán funcionalidades adicionales que supondrán un avance importante en su desarrollo final. Los objetivos principales definidos en esta versión son:

- **Despliegue de un entorno de desarrollo:** Con el objetivo de facilitar y agilizar el método de trabajo, además de mejorar la comprensión del proyecto, se ha dedicado esfuerzo en montar un entorno de desarrollo adecuado que nos permita trabajar cómodamente. Es por esto que se proporcionan los recursos necesarios para facilitar el despliegue de la solución en el entorno de desarrollo. Esto nos permitirá modificar y cargar los procesos del motor KIE [7] sin necesidad de reiniciar la aplicación y contar con un modo de depuración para controlar el flujo de ejecución del programa.

- **Centralización de tareas en múltiples motores de procesos:** Esto supondrá una mejora notable en la conectividad de la aplicación, que ahora soportará la conexión simultánea con los servidores KIE que se configuren, permitiendo recibir tareas desde varias fuentes de origen y gestionarlas desde una sola ubicación.

- **Desarrollo gradual:** La aplicación se ha desarrollado de manera gradual, abordando los distintos desafíos que se iban encontrando proporcionando soluciones integradas unas con otras que cohesionan en la aplicación completa. Esto incluye dificultades cómo adaptar los modelos de procesos, las pruebas de test, o el código existente de la aplicación.

- **Mejora en la visualización y presentación de tareas:** Facilitará la gestión a los usuarios , mejorando la visualización y organización de sus tareas a través de funciones de filtrado y ordenación según los diferentes campos de las tareas. También se añadirá nueva información ligada a las tareas humanas para poder gestionarlas con mayor eficacia y permitiendo priorizarlas según distintos criterios.

- **Gestión avanzada de tareas humanas:** El proyecto se centrará en la función principal de gestión de tareas humanas [3], a la que se añadirá la consulta de un historial de tareas ya completadas por el usuario, así como la acción de ver las respuestas al cuestionario asociado a la misma, usando plantillas generadas dinámicamente con Thymeleaf. Por otra parte se gestionará el tiempo de vida de una tarea no completada, es decir, cuándo expira.

- **Vinculación unívoca de tareas:** Se ha establecido una vinculación única entre las tareas humanas del proceso jBPM y los recursos FHIR (Task y QuestionnaireResponse), lo que favorece la integración del sistema y facilita el seguimiento y análisis de las tareas humanas en el entorno de la

organización sanitaria.

1.3 Descripción de los siguientes capítulos

El resto del documento estará dividido en los siguientes capítulos:

- 2. Estado de la tecnología: Explicación de las tecnologías usadas en este proyecto.
- 3. Trabajo realizado: Se describirá la solución desarrollada.
- 4. Conclusiones y trabajo futuro: Se resumirán los apartados anteriores dando una visión general y se proporcionará una recomendación sobre posibles mejoras posteriores.

2.ESTADO DE LA TECNOLOGÍA

En este capítulo comentaremos los paradigmas, estándares, y tecnología usados para poder diseñar, desarrollar e implementar este proyecto.

2.1 Java

Es un lenguaje de programación de alto nivel, orientado a objetos, que se utiliza ampliamente en el desarrollo de software gracias a su capacidad de ejecutarse en cualquier plataforma. [28]

En este proyecto usaremos la versión de Java 8, cuya documentación puede encontrarse en [29].

2.2 Paradigma BOM

Browser Object Model [31], es la forma en que los navegadores web manipulan y representan los elementos de una página web a través de un conjunto de objetos que se pueden acceder mediante JavaScript. Su enfoque radica en la interacción y manipulación de elementos en el navegador para crear aplicaciones web dinámicas e interactivas.

El proyecto se centra en priorizar la representación en el servidor usando plantillas con Thymeleaf [30], minimizando el JavaScript necesario para aumentar así el dinamismo de la aplicación y disminuyendo el uso del BOM lo máximo posible.

2.3 Thymeleaf

Thymeleaf [30] es un motor de plantillas moderno y flexible para Java, diseñado para trabajar tanto en entornos web como no web. Fue creado para permitir a los desarrolladores de aplicaciones Java renderizar vistas de una manera más sencilla y eficiente, con soporte para diversas tecnologías de vista y plantillas.

Las principales características de Thymeleaf son: sintaxis natural, integración con Spring, capacidad de procesamiento en cliente y servidor, motor de plantillas basado en XML/HTML, dialectos personalizados e internacionalización. Se prioriza frente a JavaScript para aumentar el dinamismo de la aplicación.

2.4 FHIR

FHIR [32] (Fast Healthcare Interoperability Resources) es un conjunto de reglas y especificaciones para el intercambio de información de atención médica. Está diseñado para ser flexible y adaptable, facilitando la interoperabilidad entre diferentes sistemas de salud, permitiéndoles comunicarse y compartir datos de manera segura y eficiente.

En este proyecto vamos a usar principalmente 3 recursos FHIR, estos son:

- **Task:** [11] Proporciona una forma de gestionar y rastrear la ejecución de diversas actividades o flujos de trabajo dentro del ámbito sanitario. Las tareas representan acciones individuales a realizar, como tareas administrativas, procedimientos clínicos o incluso flujos de trabajo complejos que involucran múltiples pasos. Además, este recurso lleva un seguimiento del estado de progreso de la actividad, indicando si se ha iniciado o debería iniciarse, y finalmente si se ha finalizado con éxito o no.

  Las tareas pueden tener inputs y outputs que será necesario saber manejar. Los primeros son elementos que deben estar presente en la tarea para poder ejecutarse y completarse. Los outputs representan resultados finales producidos por la tarea.

- **Questionnaire:** [12] Se utiliza para definir un conjunto de preguntas destinadas a recopilar información. Incluye la estructura y el contenido del cuestionario, especificando los tipos de preguntas y las posibles respuestas.

  Las preguntas y el tipo de datos esperado están presentes en el campo Item y pueden tener una

estructura básica o tener subpreguntas. Respecto a los Items hay tres atributos claves que debemos conocer:

- **LinkId**: se refiere al Id único de la pregunta dentro del cuestionario.
- **Required**: indica obligatoriedad de responder a la pregunta para completar el cuestionario.
- **Repeats:** Indica si la pregunta puede tener más de una respuesta. Cada Item siempre va acompañado de un título donde se muestra la pregunta.

- **QuestionnaireResponse:** [13][14] Representa los datos recopilados al completar un cuestionario. Captura las respuestas dadas a cada pregunta en el recurso Questionnaire correspondiente. En el campo Item del cuestionario vienen presentes los títulos de las preguntas, y las respuestas en el atributo Answer.

  Tanto el recurso Questionnaire como el QuestionnaireResponse manejan una serie de tipos de datos definidos por FHIR [9][10]:

  - **Coding:** tipo complejo de datos formado principalmente por una variable code (lista controlada de cadenas predefinidas) y un display (texto asociado al code)**.**
  - **Quantity:** tipo complejo de datos formado principalmente por un valor de tipo decimal y una unidad que es una variable tipo code.
  - **Reference:** tipo complejo de datos. Formado principalmente por el campo reference que almacena una URI, id, extensión o referencia a un objeto FHIR.
  - **String:** tipo simple de datos. Cadena de texto.
  - **URI:** tipo simple de datos. Un enlace o referencia que siga el protocolo RFC 3986.
  - **Boolean:** tipo simple de datos. True/false.
  - **Integer:** tipo simple de datos. Número entero con signo.
  - **Decimal:** tipo simple de datos. Número decimal con signo.
  - **Date:** tipo simple de datos. Una fecha en formato ISO8601.
  - **Time:** tipo simple de datos. Una hora del día en formato ISO8601.
  - **DateTime:** tipo simple de datos. Una fecha y hora del día en formato ISO8601.

2.5 BPM

La gestión de procesos de negocio (Business Process Management, BPM) es un paradigma que persigue analizar y mejorar de forma continuada cada proceso de negocio de una organización [33].

Esta metodología da soporte a:

- Analistas de negocio: Se pueden definir de forma clara y concisa los procesos de trabajo de la organización.

- Desarrolladores: Es posible desarrollar una solución a los procesos de trabajo a partir de la definición clara y concisa de los analistas.

- Usuarios finales: Los usuarios finales podrán analizar y mejorar de forma continuada los procesos definidos y desarrollados.

Gracias a esta metodología se podrán diseñar mejores soluciones a los problemas de negocio que puedan tener las organizaciones.

Además, la lógica de negocio no está codificada, se usa la filosofía low-code [34] para que se pueda definir la lógica de negocio sin tener que codificar y de esta forma sea fácilmente entendible por todos los implicados.

2.5.1 Low-code

El low-code [34] es un enfoque del desarrollo de software que agiliza el desarrollo de aplicaciones usando la cantidad mínima posible de código. Las plataformas low-code proporcionan un entorno de desarrollo visual que permite a los usuarios crear aplicaciones de manera rápida y eficiente utilizando herramientas gráficas y configuraciones en lugar de codificación tradicional.

Esta filosofía es algo que BPM busca para poder definir toda la lógica de los procesos de forma gráfica.

2.5.2 Procesos de negocio

Un proceso de negocio [36], define el flujo de las tareas o conjunto de actividades desarrolladas para alcanzar un objetivo específico. Reflejan la forma de trabajar en la organización, tareas realizadas, entradas y salidas, relaciones, dependencias, etc.

Gracias a BPM podemos evaluar los procesos existentes para encontrar formas de mejorar su eficiencia y reducir errores y costes.

2.5.3 Activos de negocio

Los activos de negocio [37] son los distintos elementos y componentes que forman parte de la definición, implementación y ejecución de los procesos de negocio. Estos activos sirven para modelar, gestionar y automatizar los procesos de negocio dentro de una organización.

Algunos ejemplos de activos de negocio son:

- **Modelos de procesos:** Representaciones gráficas de los flujos de trabajo de los procesos de negocio. Estos diagramas definen las actividades, eventos, puertas de decisión y otros elementos que componen un proceso.
- **Definiciones de Tareas Humanas [3]**: Descripciones de las tareas que requieren intervención humana, incluyendo asignaciones, plazos, formularios asociados y reglas de notificación. Estas son las tareas en las que centramos este proyecto.
- **Formularios y plantillas:** Elementos utilizados para la entrada de datos por parte de los usuarios, como formularios web, plantillas de correo electrónico, etc.
- **Variables y datos de proceso [54]:** Información específica del proceso que se utiliza para controlar su flujo y comportamiento.

2.6 Aplicación de negocio

Una aplicación de negocio [37][38] es una solución que facilita la ejecución de los procesos de interés en una organización y que da soporte tanto a la ejecución de las tareas (automáticas o manuales) como al seguimiento y análisis de dichos procesos. En lugar de codificar la lógica de los procesos en la aplicación, se expresa a través de modelos, lo que facilita su análisis, diseño y evolución. En este proyecto, hemos utilizado las herramientas proporcionadas por KIE para lograr estos objetivos.

Una aplicación de negocio se divide en tres componentes fundamentales:

- **Tipos de datos:** Representa las entidades de información del negocio.
- **Modelo de procesos:** Representa los flujos de trabajo de los procesos de negocio.
- **Aplicación:** Se programan las tareas individuales y se gestionan las tareas humanas. Las principales virtudes de una aplicación de negocio son:
  - **Reutilización:** Se pueden usar los tipos de datos definidos en distintas aplicaciones de negocio.
  - **Evolución independiente:** Se puede modificar la aplicación sin necesidad de modificar el modelo del proceso y manteniendo su funcionalidad.
  - **Adaptabilidad:** Los modelos del proceso pueden cambiar y, gracias a herramientas como las propuestas en KIE, se pueden realizar estos cambios de forma sencilla para adaptarse a las necesidades del negocio
  - **Seguimiento del proceso:** una aplicación de negocio permite hacer un seguimiento del proceso para su mejora continua.

Los modelos de procesos por sí solos no son suficiente, necesitan un motor de procesos que interprete estos modelos, en este proyecto se ha usado la tecnología proporcionada por KIE para este propósito.

2.7 KIE

KIE (Knowledge Is Everything) [35][27] es una plataforma de código abierto que proporciona un conjunto de herramientas y componentes para la gestión del conocimiento y la automatización de decisiones en entornos empresariales. Sirve como ecosistema para distintos proyectos de código abierto enfocados en la automatización de procesos de negocio.

Algunos proyectos dentro de este ecosistema son Kogito, Drools y jBPM. Entre todas las funcionalidades que nos provee KIE, se incluye el motor de procesos que interpreta modelos de procesos.

2.7.1 jBPM

jBPM (Java Business Process Management) [39] es un conjunto de herramientas de código abierto que sirve para construir aplicaciones de negocio y facilitar el control de los flujos de procesos de negocio.

jBPM puede usarse como un servidor independiente (por ejemplo, mediante KIE Server) o embebido dentro de una aplicación personalizada. Es compatible con varios frameworks, y en nuestro caso, lo integraremos con el framework Spring Boot.

Los procesos de negocio se pueden representar en modelos que muestran el detalle de las tareas a realizar para conseguir un objetivo a partir de unos datos de entrada, además de generar unos datos de salida.

Los procesos se pueden representar de forma estándar, con BPMN [40] una notación normalizada. jBPM incluye estas definiciones y otros activos de empresa en un contenedor que denomina kjar. Por tanto, un kjar es un contenedor de activos de negocio que puede desplegarse en un servidor KIE, que hará de motor de los procesos y reglas incluidos. Para crear este contenedor haremos uso del Business Central, que permite la definición de activos de negocio y la construcción del kjar.

Para implementar la lógica empresarial, jBPM aprovecha las capacidades de varios marcos de trabajo, como procesos comerciales, reglas comerciales y restricciones de planificación, pero también persistencia, mensajería, transacciones, etc.

2.7.1.1 Kjar

Kjar en jBPM [41] es un archivo JAR que contiene todos los activos de negocio necesarios para ejecutar procesos de negocio en la plataforma KIE. Estos activos incluyen definiciones de procesos, reglas de negocio, formularios de usuarios y otros recursos relacionados con la ejecución de procesos en el entorno jBPM.

Tener todo esto en un mismo archivo JAR facilita su distribución, implementación y gestión.

2.7.1.2 Definición de un proceso en jBPM

Una definición [52] de proceso es un archivo BPMN 2.0 que sirve como contenedor para un proceso y su diagrama BPMN. La definición del proceso muestra toda la información disponible sobre el proceso de negocio, como los subprocesos asociados o el número de usuarios y grupos que participan en la definición seleccionada.

2.7.1.3 Business Central

Business Central [51], es lo que se conoce como un WorkBench, que nos ofrece distintas herramientas para trabajar con jBPM. Por ejemplo, nos permite editar, ejecutar y controlar el flujo de los procesos para poder probar si el diseño es correcto. Las funcionalidades que permite son:

- **Creación de proyectos:** donde los usuarios podrán diseñar procesos de negocio y otros activos de negocio.
- **Control de los servidores:** permite controlar los motores y desplegar los Kjar en ellos.

- **Gestión de acceso a los procesos:** permite gestionar a qué procesos puede acceder cada usuario o grupo de usuarios.

- **Seguimiento de tareas asignadas:** nos proporciona la opción de seguir las tareas que tienen asignadas los usuarios, en motores controlados por BC.

Business Central soporta la filosofía low-code [34]. Permite a los usuarios modelar procesos, reglas de negocio y formularios a través de interfaces gráficas intuitivas, reduciendo la necesidad de codificación manual.

2.7.1.4 Instancia de proceso en jBPM

La instancia de proceso [53] en jBPM es la ejecución de un proceso. Cuando se crea una instancia de un proceso, se crea un token para marcar el camino de la ejecución. Este token se llama token raíz de la instancia de proceso y se posiciona en el nodo inicial de la definición del proceso. Conforme se van ejecutando las tareas, el token avanza, hasta que llega al final del mismo. Es posible tener múltiples instancias independientes de un mismo proceso.

2.7.1.5 Servidor de ejecución KIE

El servidor de ejecución KIE [7][42] es un motor de procesos y reglas, que interpreta los modelos y puede generar instancias de procesos y evaluar reglas de negocio cuando sea necesario. Se integra con Business Central para la ejecución de procesos y tareas.

Dentro del servidor se ejecutan de forma autónoma contenedores. Que son instancias almacenadas en memoria de un kjar. Permitiendo la declaración y uso de los procesos del modelo cargado en el contenedor. También posee un controlador REST, responsable de manejar los recursos gestionados por el servidor KIE, (instancias de procesos, tareas humanas, etc …).

2.8 SpringBoot

SpringBoot [43] es un framework de código abierto para el desarrollo de aplicaciones Java. Está diseñado para simplificar el proceso de creación y despliegue de aplicaciones basadas en Spring.

La principal ventaja de Spring es su flexibilidad, facilitando la integración con otros frameworks, como se puede observar en este proyecto en el que integramos tanto el framework de Spring, que nos ayuda con la configuración y la estructura, como el framework de jBPM. De esta forma podemos realizar una aplicación Spring que permita trabajar con la filosofía de BPM gestionando los procesos y las tareas de esos procesos.

2.8.1 Starter de Spring para jBPM

Los Starter de Spring son módulos o dependencias preconfiguradas que facilitan la integración de cualquier tecnología, por ejemplo, jBPM, en una aplicación basada en Spring Boot. Estos starters incluyen todas las dependencias necesarias y la configuración básica para integrarlas en la aplicación Spring.

Spring Initializr [44] genera la estructura del proyecto y añade las dependencias automáticamente, así los desarrolladores pueden comenzar rápidamente con la tecnología elegida dentro de una aplicación Spring.

La ventaja de integrar jBPM en Spring Boot es que podemos inyectar las dependencias de jBPM en nuestra aplicación y usar sus librerías de forma sencilla gracias a las etiquetas de Spring @Injection y @Autowire, además de poder usar todas las demás ventajas propias de Spring Boot como framework.

2.9 Maven

Maven [45] es una herramienta de automatización de construcción de código que se utiliza principalmente para proyectos Java. Los objetivos de Maven son:

- Facilitar la gestión de dependencias.

- Proporcionar un sistema uniforme de construcción de proyecto.

- Proporcionar información de calidad del proyecto.
- Fomentar el desarrollo de buenas prácticas.

Maven es una tecnología fundamental que da soporte a toda la idea de aplicación de negocio, ya que se busca la mayor independencia posible entre las partes y la reutilización de cada parte.

Cada parte de la aplicación de negocio comentada anteriormente se maneja como un artefacto de Maven, es decir, los modelos de datos, los modelos de los procesos y la aplicación o servicio son artefactos independientes, que luego a la hora de compilar se componen formando una única aplicación.

Toda la información relativa a las dependencias de Maven usadas en la aplicación está dentro del fichero pom.xml.

2.10 Git

Git [46] es un sistema de control de versiones distribuido y de código abierto que permite tener un registro de los cambios realizados en el código.

Git es distribuido, esto quiere decir que los desarrolladores manejan una copia local del repositorio. Gracias a esto no se depende de un servidor centralizado para confirmar los cambios, lo que hace a Git más robusto y descentralizado.

Git permite la creación de ramas independientes en el repositorio. Cada nueva funcionalidad puede desarrollarse en su propia rama, lo que permite a los desarrolladores trabajar en paralelo sin interferir en el código base principal, las modificaciones hechas en una rama no afectan al resto del proyecto hasta que se fusionan, lo que ayuda a mantener el código base estable. En resumen, es una herramienta esencial para el desarrollo de proyectos.

En este proyecto git es relevante para soportar dos tareas:

- Business Central usa git para la gestión de los proyectos kjar. De esta forma tenemos que clonar ese proyecto en nuestro local para poder trabajar con el kjar.
- El proyecto git está ubicado en un repositorio remoto que se aloja en la web Github: https://github.com/tfg-projects-dit-us/Healthcare-Tasks/ .

Para trabajar con este proyecto se realizó un fork del repositorio, se creó una nueva rama `jmrefactor` donde se desarrollaron los cambios, y se integraron estas modificaciones al repositorio original haciendo un pull-request con destino a la misma rama.

2.10.1 Exportación de un proyecto de Business Central a nuestro local

1. Se va a los ajustes de nuestro proyecto:

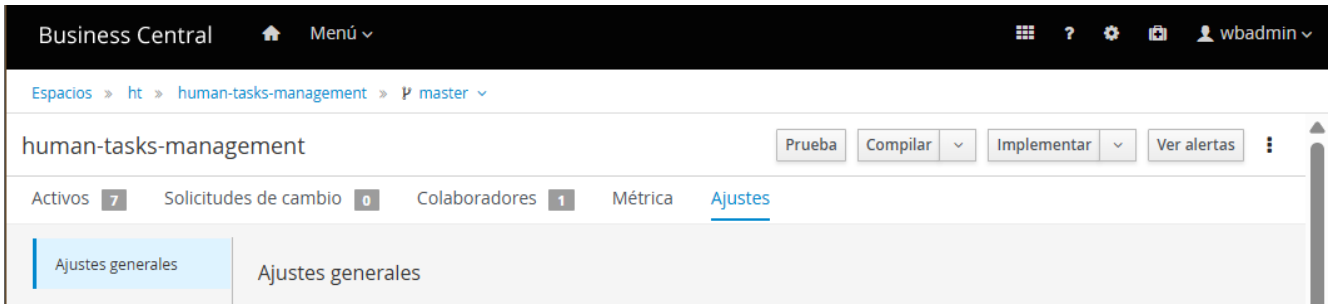

Ilustración 2. Acceso al menú ajustes de un proyecto de Business Central

2. En los ajustes generales elegimos la URL de http y la copiamos:

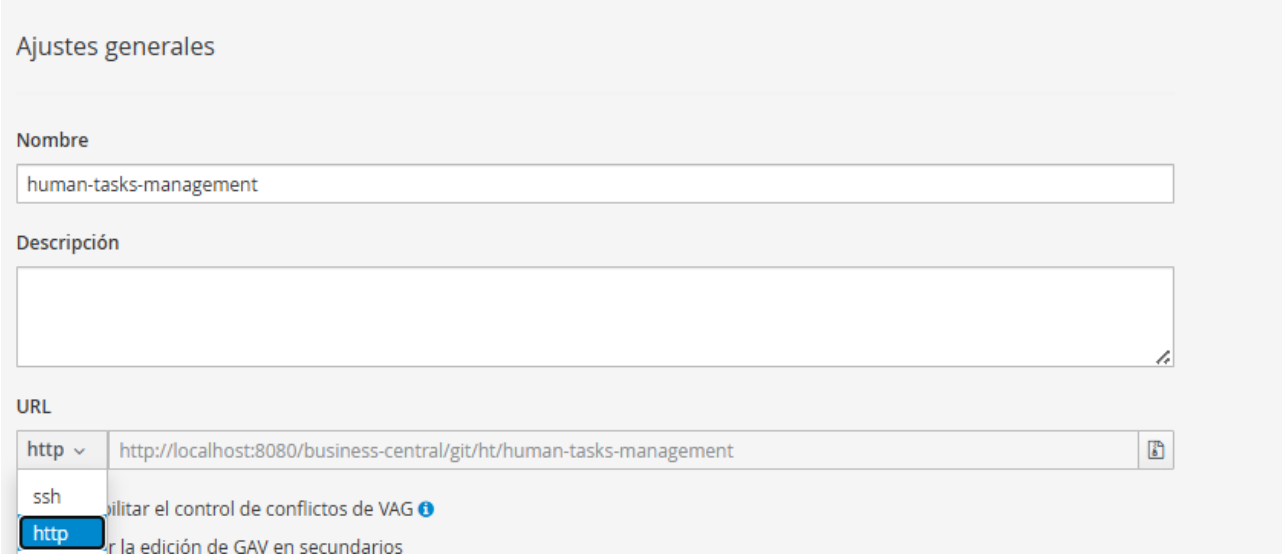

Ilustración 3. Ajustes generales de un proyecto de Business Central

3. Desde el terminal de nuestro IDE, en este caso Eclipse, nos vamos a la carpeta raíz del proyecto y hacemos un git clone de la URL copiada en el paso anterior:

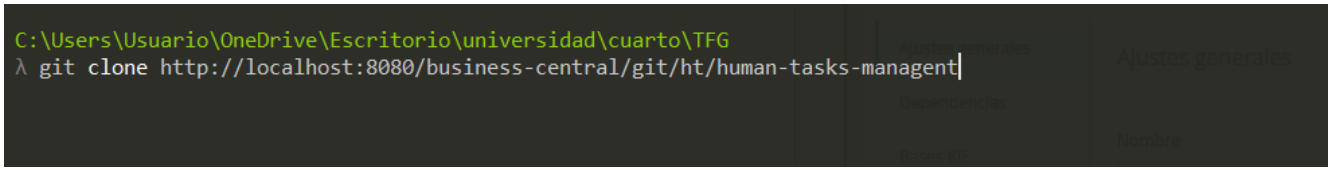

Ilustración 4. Comando git clone

Con esto tendríamos el kjar en nuestro local.

2.10.2 Importación de un proyecto de un repositorio remoto a Business Central

1. En el menú de Proyectos, Seleccionamos el espacio donde queremos importar el proyecto, en caso de no tener ningún espacio lo podemos crear pulsando en "Agregar Espacio":

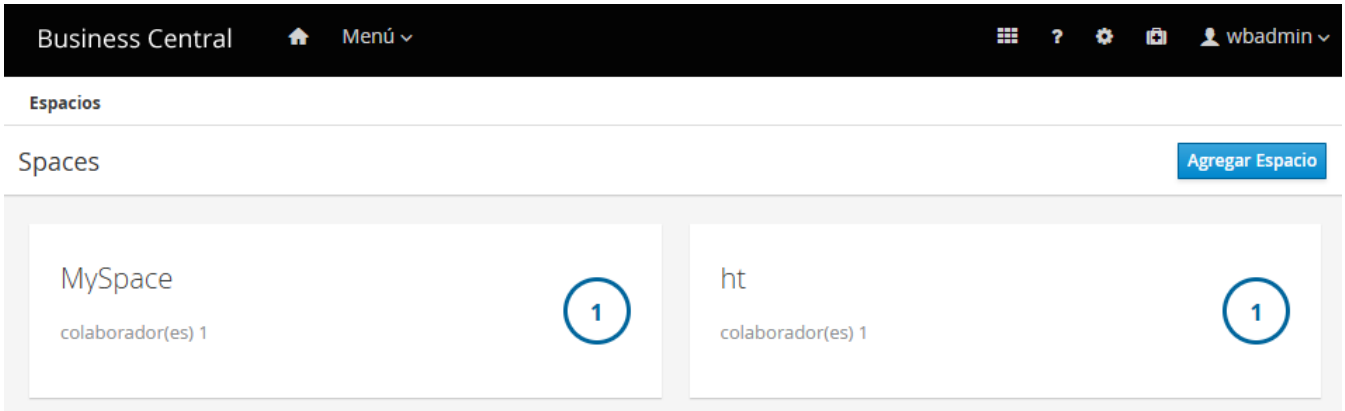

Ilustración 5. Menú selección espacios Business Central

2. Dentro del espacio seleccionado, a la derecha en el botón "Agregar proyecto" pulsamos el desplegable y clicamos en "Importar proyecto":

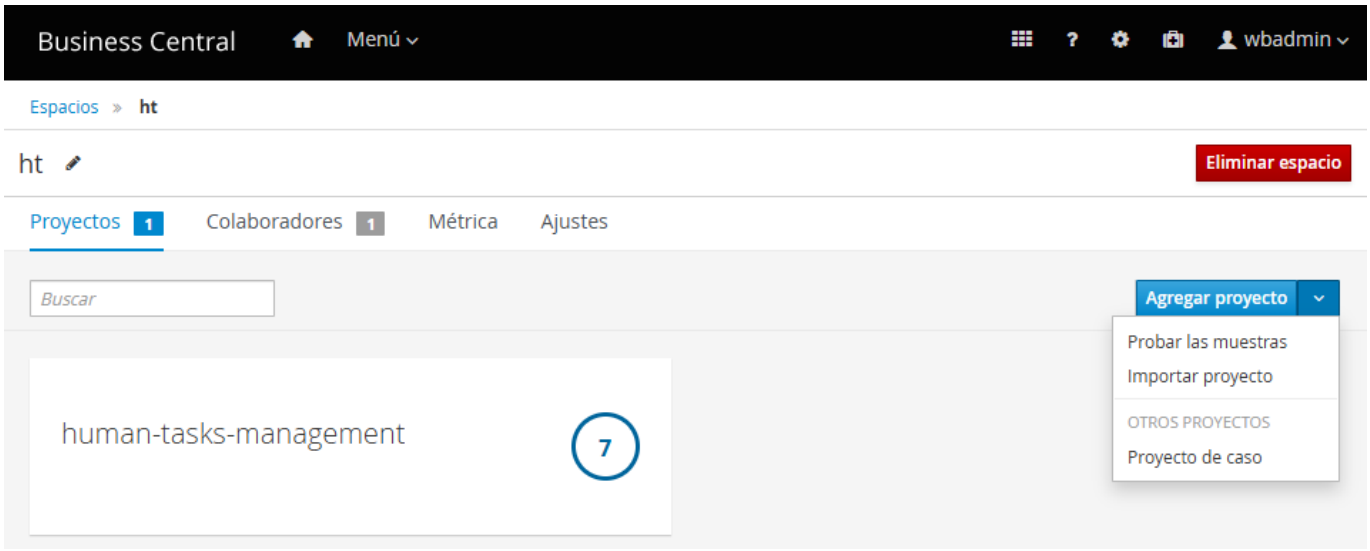

Ilustración 6. Menú proyectos Business Central

3. Añadimos la URL del proyecto Git que queremos importar a Business Central. Debemos saber que para que se pueda importar un proyecto desde un repositorio remoto, este repositorio **debe** tener creada la rama master, en caso contrario Business Central no podrá importar el repositorio:

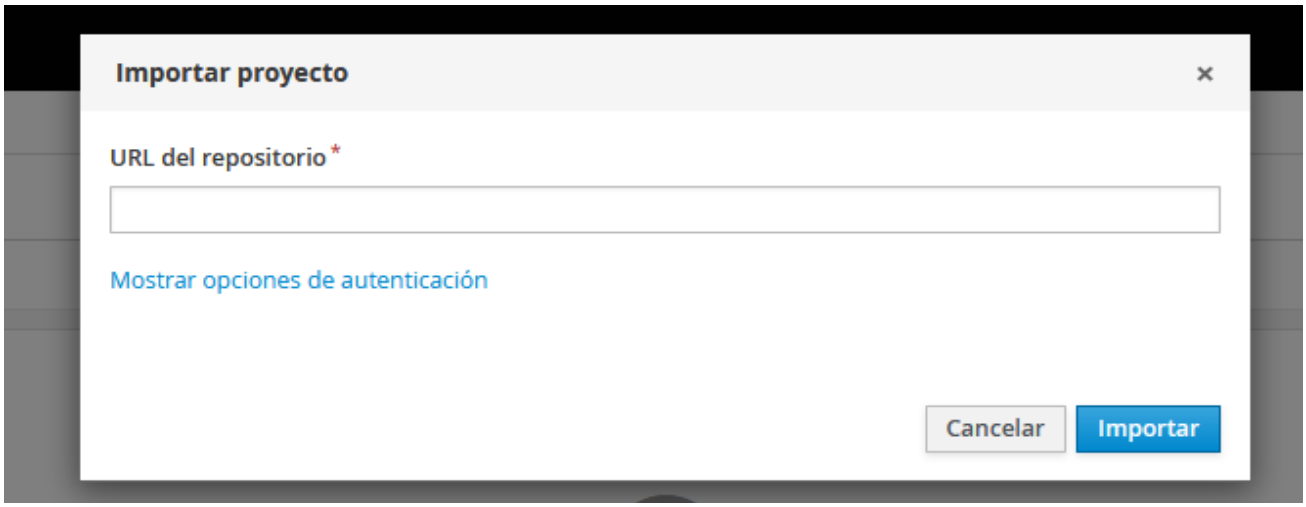

Ilustración 7. Menú introducción URL del repositorio remoto en Business Central

4. Seleccionar el kjar correspondiente y pulsar "Aceptar":

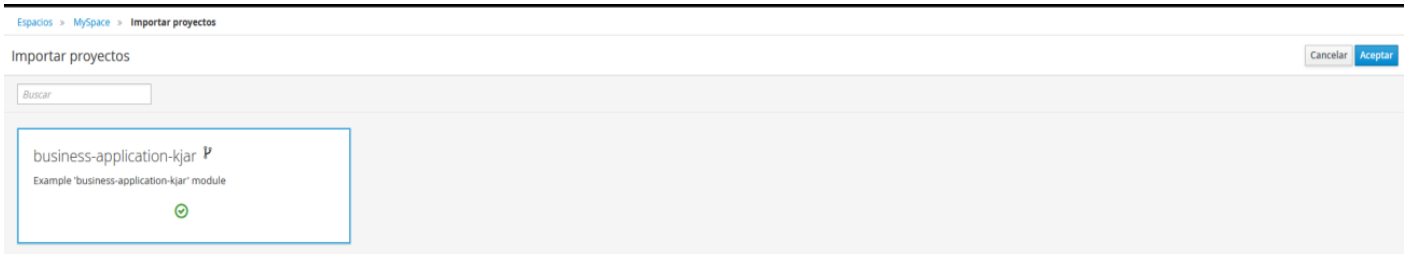

Ilustración 8. Menú importar proyectos Business Central

5. Ya estaríamos listos para trabajar con nuestros procesos:

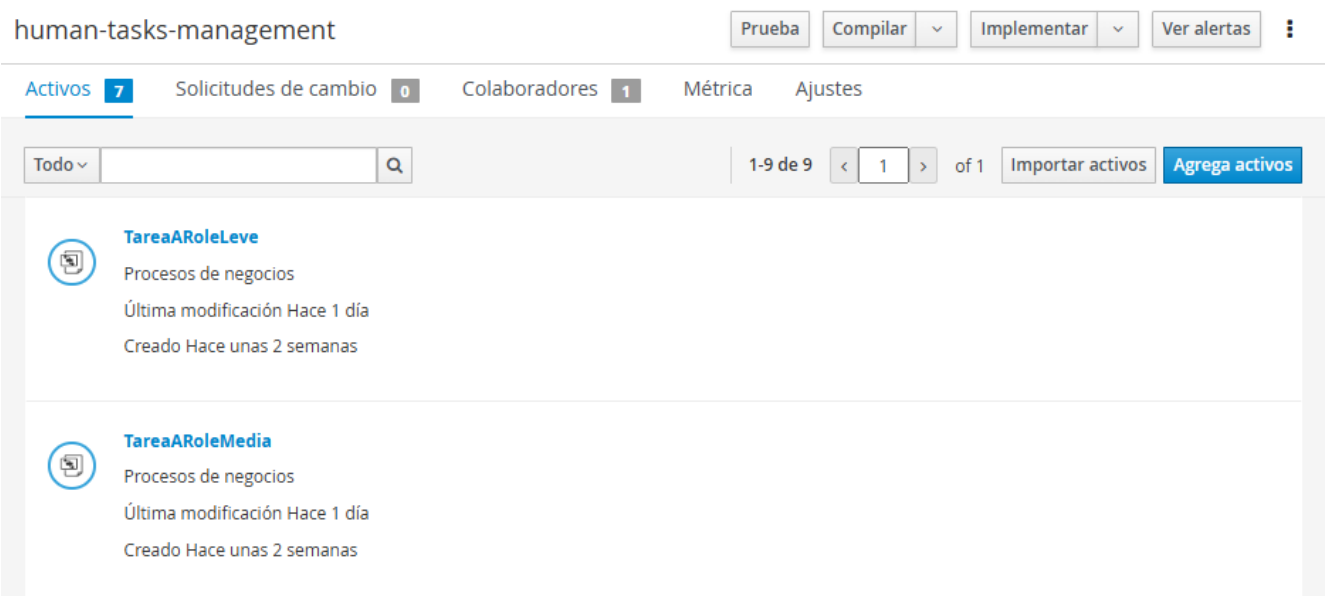

Ilustración 9. Menú principal de los activos Business Central

2.11 PostgreSQL

PostgreSQL [47] es un sistema gestor de base de datos relacional de código abierto que usa y extiende el lenguaje SQL combinado con otras características que ayuda a guardar de forma segura y escalar las cargas de trabajo de alto rendimiento.

Las distintas características que hace tan atractivo a PostgreSQL son:

- **Juego de datos:** Puede guardar muchos tipos de datos distintos
- **Integridad:** Proporciona integridad de los datos guardados.
- **Rendimiento.**
- **Fiabilidad.**
- **Seguridad.**
- **Extensibilidad.**
- **Búsquedas personalizadas:** Su búsqueda de texto permite caracteres internacionalizados y más.

2.12 JPDA

La Arquitectura de depuración de la Plataforma Java (JPDA) [1], es un sólido marco proporcionado por Java para la depuración de aplicaciones que se ejecutan en una Máquina Virtual Java (JVM). Este marco se compone de tres elementos principales que trabajan conjuntamente:

- **JVM TI (Java Virtual Machine Tool Interface)**: Interfaz que permite examinar el estado y controlar la ejecución de las aplicaciones dentro de la Máquina Virtual Java.
- **JDWP(Java Debug Wire Protocol)**: Protocolo de comunicación que facilita el intercambio de información de depuración entre el depurador y la JVM, posibilitando la depuración remota.
- **JDI(Java Debug Interface)**: Es una interfaz de alto nivel de Java que interactúa con el depurador y simplifica la creación de software de depuración con la JVM.

2.13 Docker

Docker [48] es un proyecto de software de código abierto, que nos facilita la creación, implementación y ejecución de aplicaciones mediante contenedores. Estos contenedores permiten empaquetar una aplicación de manera eficiente, asegurando que se ejecute de manera correcta y consistente en diferentes entornos a partir de una imagen. Este empaquetado incluye el código de la aplicación, las herramientas del sistema necesarias, las

bibliotecas específicas y dependencias requeridas para su funcionamiento, garantizando que no haya conflicto con el entorno subyacente.

Docker tiene una serie de ventajas bastante marcadas que justifican su popularidad:

- **Portabilidad**: Es la principal ventaja de Docker. El servicio de empaquetado que ofrecen sus contenedores puede ejecutarse en cualquier sistema que soporte Docker, y trasladarse fácilmente de una plataforma a otra, ya sea desde un entorno local de desarrollo hasta un servidor en la nube, pasando por diferentes sistemas operativos y arquitecturas. Asegurando que las aplicaciones funcionen de manera uniforme en distintos entornos.
- **Eficiencia**: Los contenedores comparten el núcleo del sistema operativo, lo que reduce el uso de recursos en comparación con las máquinas virtuales tradicionales.
- **Rapidez**: La creación y eliminación de contenedores es rápida, lo que agiliza los ciclos de desarrollo y despliegue.
- **Aislamiento**: Cada contenedor opera de manera independiente, evitando conflictos entre aplicaciones y mejorando la seguridad.

Además, gracias a su versatilidad, Docker es usado ampliamente en distintos ámbitos:

- **Desarrollo y pruebas**: Proporciona entornos aislados para desarrollar y probar aplicaciones sin interferencias.
- **Implementación**: Facilita la implementación de aplicaciones en diversos entornos, garantizando consistencia y reduciendo problemas relacionados con diferencias en configuraciones.
- **Escalabilidad**: Permite escalar aplicaciones de manera eficiente mediante la orquestación de múltiples contenedores.

3. TRABAJO REALIZADO

En este capítulo se va a presentar el trabajo desarrollado en el proyecto. Esto incluye:

- Una guía para poner en marcha el entorno de trabajo, compuesto por un modo desarrollador y un modo depuración.
- La implementación de nuevas funcionalidades y modificaciones en el código original para lograr los objetivos, siempre siguiendo la arquitectura del patrón Modelo Vista Controlador.

  En concreto, la gestión de tareas en múltiples motores de procesos, la consulta de un historial de tareas completadas, la generación dinámica de cuestionarios respuesta y la mejora en la visualización y presentación de tareas mediante nuevos campos, filtros y ordenación.
- Instrucciones de ejecución de la aplicación, tests de pruebas realizados y verificación y análisis de los resultados.

3.1 Configuración del entorno de trabajo

Con el objetivo de facilitar y agilizar el método de trabajo, además de mejorar la comprensión del proyecto, es recomendable montar un entorno de desarrollo adecuado que nos permita trabajar cómodamente.

Es por esto que se proporciona un modo desarrollador que nos permitirá modificar y cargar los procesos del motor KIE sin necesidad de reiniciar la aplicación y un modo debug para controlar el valor de las variables del programa y establecer puntos de parada en él.

3.1.1 Modo desarrollador

Su principal ventaja es la capacidad de trabajar en conjunto con Business Central en caliente, es decir, en tiempo de ejecución. Con Business Central actuando como controlador del motor KIE embebido en la aplicación, podremos editar y desplegar el kjar (acctivos de negocio) desde BC. La aplicación desarrollada interacciona con el motor para gestionar las tareas humanas y no será necesario detenerla para actualizar los modelos, ya que estos están siendo desplegados desde BC. También se activarán las trazas de la aplicación para tener indicadores del flujo del programa y facilitar su seguimiento. A continuación, se explicará en detalle cómo poner el modo desarrollador en funcionamiento:

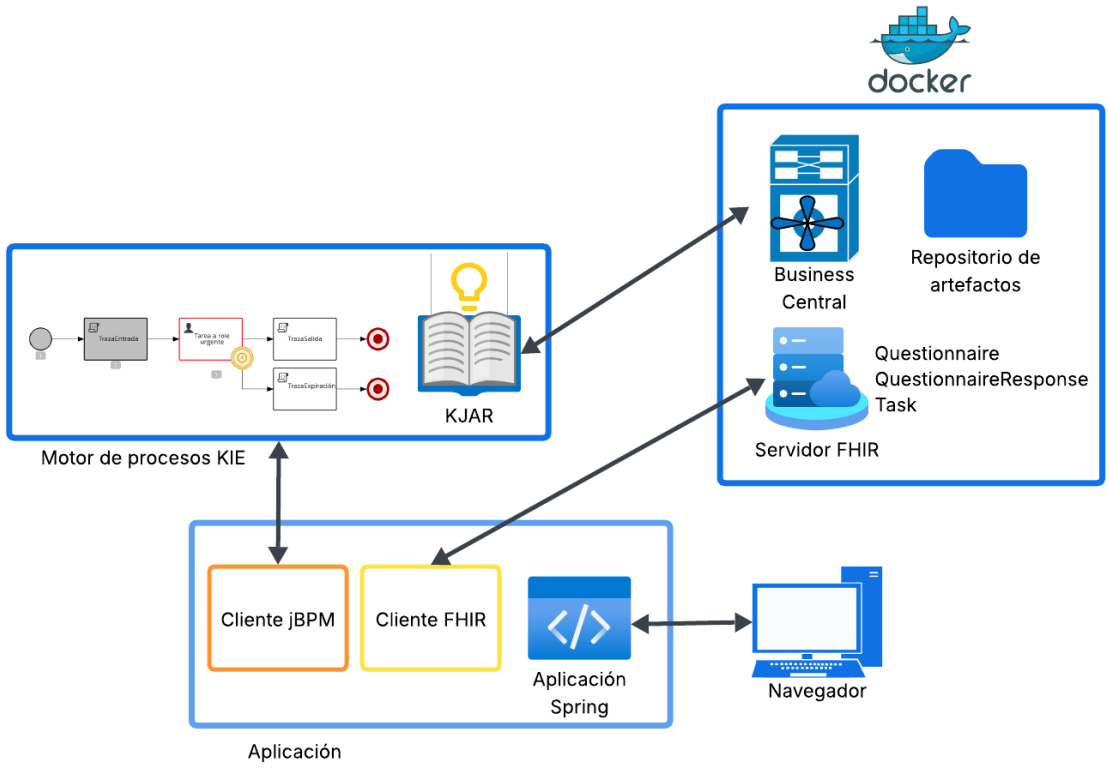

Ilustración 10. Estructura del modo desarrollador

3.1.1.1 Despliegue del Entorno de Desarrollo

En primer lugar, se clona el repositorio https://github.com/tfg-projects-dit-us/EntornoDesarrollojBPM para levantar localmente (en contenedores docker [48]) los servicios que dan soporte al desarrollo y verificación del proyecto. Simplemente deberá situarse en el directorio `jBPMDev-fhir+bc+repo` en caso de querer la versión más completa y ejecutar el comando `docker compose up -d`, lo que pondrá a su disposición, localmente, los servicios de:

- FHIR: http://localhost:8888/
- Business Central: http://localhost:8080/business-central/
- Repositorio de Artefactos (opcional): http://localhost:8081/

Es recomendable instalar alguna extensión en su IDE o alguna aplicación como Docker Desktop para manejar contenedores Docker.

Una vez iniciados los contenedores deberemos acceder a Business Central, iniciar sesión con el usuario `wbadmin` y contraseña `wbadmin` y crear el usuario `controllerUser` con el rol `rest-all` y contraseña `controllerUser`. Este usuario se debe crear porque debe permitir acceder a la api del controlador KIE.

Dentro del menú principal de Business Central acceder a Ajustes > Usuarios > Nuevo Usuario e introducir los parámetros de usuario especificados.

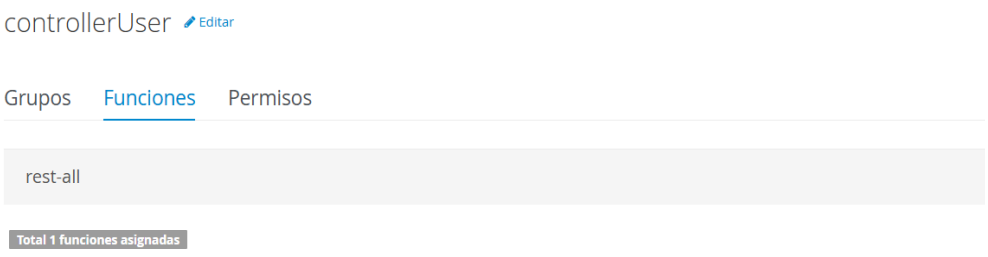

Ilustración 11. Creación de usuario controllerUser en BC

3.1.1.2 Ejecución en modo desarrollador

El siguiente paso será ejecutar en modo developer. Se muestra el caso para SO Windows. Deberá usar el comando `launch-dev.bat clean install`. En este caso el fichero de configuración utilizado es `application-dev.properties`. Es recomendable mirar que la variable de entorno JAVA_HOME configurada en `launch-dev.bat` sea conforme a la de su entorno local (o eliminar esta línea si existe la variable de entorno), y que los contenedores Docker estén funcionando.

human-tasks-service > launch-dev.bat

```
1  @echo off
2  set "JAVA_HOME=C:\Program Files\Java\jdk-11"
3  rem set "JAVA_HOME=D:\Programas\Java\jdk-11.0.8"
4  rem set "JAVA_HOME=C:\Program Files\Java\jdk-11.0.4"
5  set mavenInput="%*"
```

Ilustración 12. Configuración de variable de entorno JAVA_HOME

3.1.1.3 Conectar servidor KIE con Business Central

Llegados a este punto, la aplicación ya estaría corriendo en el puerto 8090 en modo developer. Ahora debemos conectar el servidor KIE embebido en nuestra aplicación con Business Central, que actuará como controlador, para poder manejar los procesos en tiempo de ejecución directamente desde su interfaz [42].

Una vez ejecutada la aplicación se debe de haber creado automáticamente una nueva configuración de Servidor en business-central llamada `healthcareTasks Dev`. Debe comprobar que esté conectada a un servidor remoto en el puerto 8090, iniciado por la ejecución de la aplicación. Se puede ver accediendo al menú principal de Business Central > Implementar.

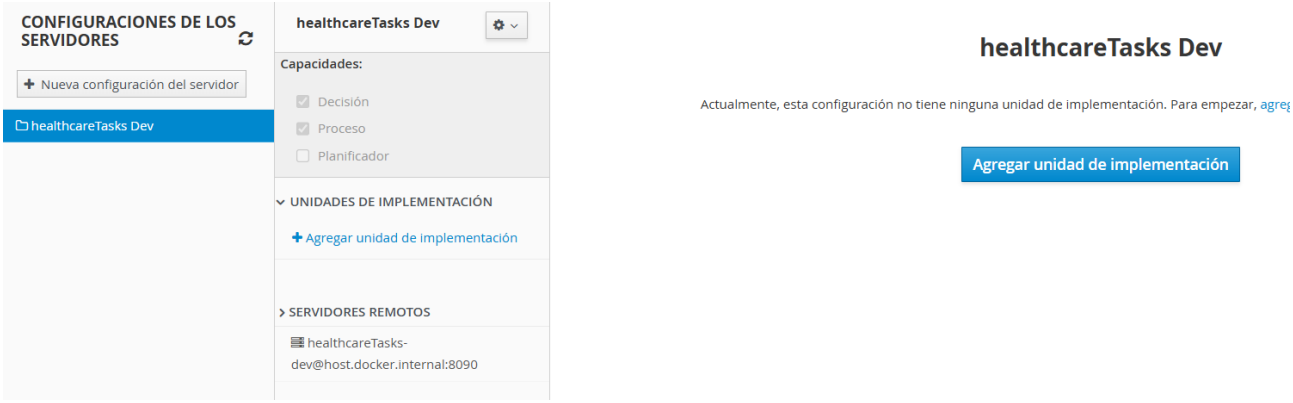

Ilustración 13. Creación del servidor remoto en Business Central

Puede comprobar que el kie-server esté activo accediendo a la url http://localhost:8090/rest/server :

Ilustración 14. Comprobación del servidor kie interno

Por último, solo falta importar el proyecto kjar que contiene los procesos BPM de nuestro programa desde nuestro repositorio remoto git siguiendo los pasos del apartado [2.10.2], y pulsando el botón de implementar. Deberá pulsar este botón cada vez que modifique su proyecto para que los cambios realizados en este se reflejen en el servidor remoto.

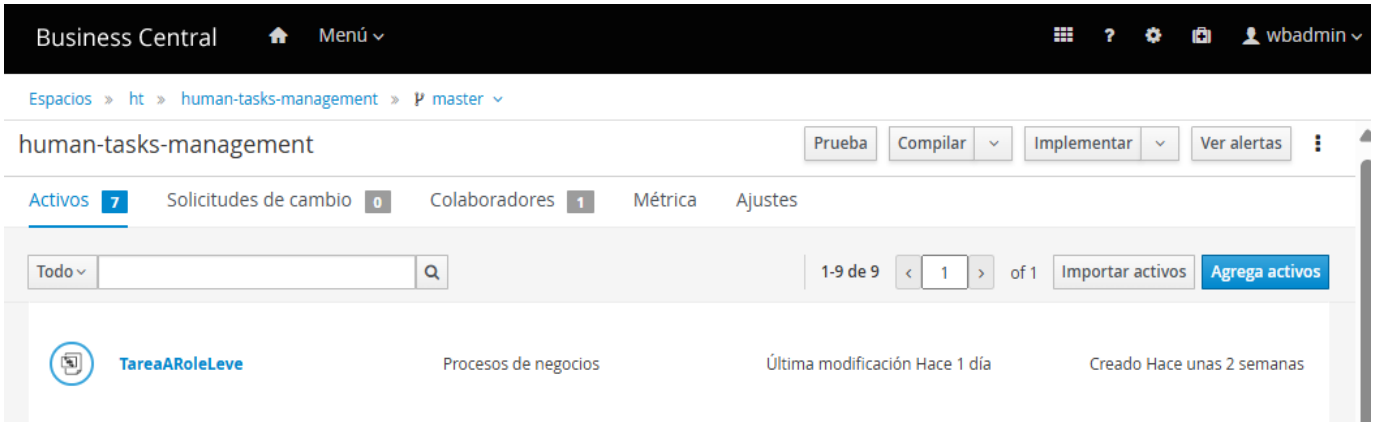

Ilustración 15. Implementación del proyecto remoto en el servidor kie

Si volvemos al menú de implementaciones veremos que se ha creado el contenedor resultado de la implementación de nuestro proyecto, con esto ya estaría en funcionamiento el modo desarrollador:

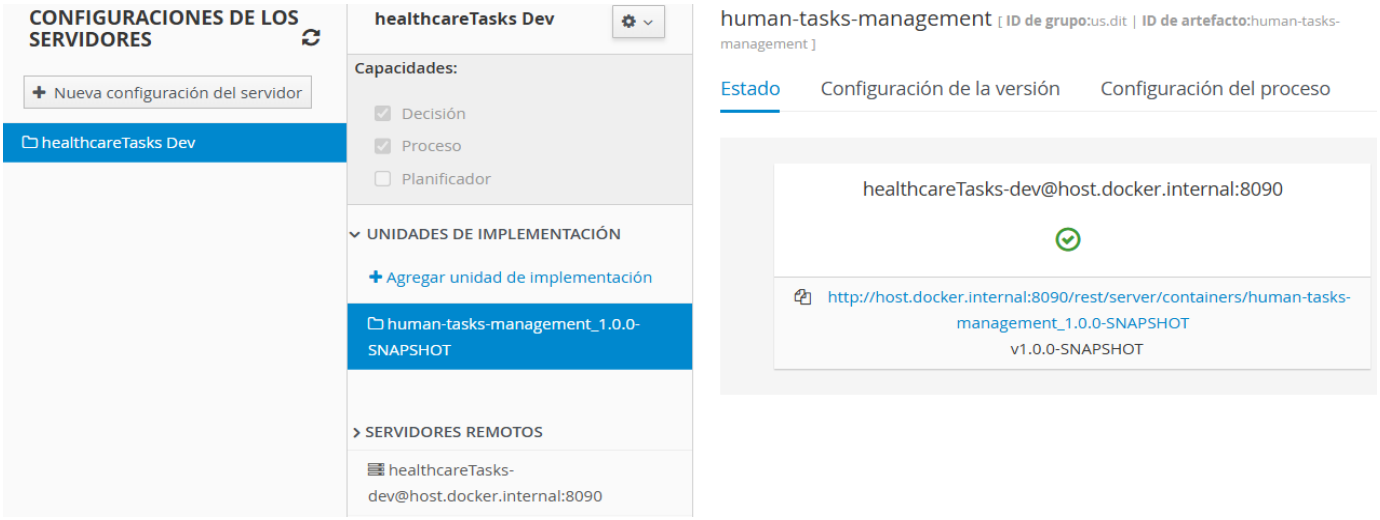

Ilustración 16. Creación del contenedor en kie-server

3.1.1.4 Verificación del modo desarrollador

Debe tener en cuenta que la conexión a base de datos no usa la base de datos PostgreSQL usada para el modo normal de ejecución, sino que se conecta a una base de datos h2 en memoria con información volátil con cada inicio de la aplicación. Esta proporciona una interfaz accesible mediante en http://localhost:8090/h2-console/ con la que podremos manejar todo el modelo de datos. Si ha ejecutado correctamente el modo developer podrá acceder a esta dirección.

Puede conectarse a la base de datos con el usuario y la contraseña por defecto `sa` y la url `jdbc:h2:./target/spring-boot-jbpm`.

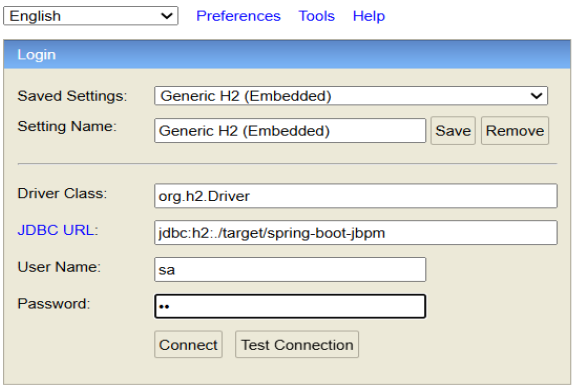

Ilustración 17. Conexión con consola h2

Es conveniente verificar que el contenedor kie está correctamente iniciado accediendo a http://localhost:8090/rest/server/containers .


```
▼<response type="SUCCESS" msg="List of created containers">
  ▼<kie-containers>
    ▼<kie-container container-alias="human-tasks-management" container-id="human-tasks-management_1.0.0-SNAPSHOT" status="STARTED">
      ▼<config-items>
          <itemName>KBase</itemName>
          <itemValue/>
          <itemType>BPM</itemType>
        </config-items>
      ▼<config-items>
          <itemName>KSession</itemName>
          <itemValue/>
          <itemType>BPM</itemType>
        </config-items>
      ▼<config-items>
          <itemName>MergeMode</itemName>
          <itemValue>MERGE_COLLECTIONS</itemValue>
          <itemType>BPM</itemType>
        </config-items>
      ▼<config-items>
          <itemName>RuntimeStrategy</itemName>
          <itemValue>SINGLETON</itemValue>
          <itemType>BPM</itemType>
        </config-items>
      ▼<messages>
          <content>Container human-tasks-management_1.0.0-SNAPSHOT successfully created with module us.dit:human-tasks-management:1.0.0-SNAPSHOT.
          <severity>INFO</severity>
          <timestamp>2025-04-21T19:04:10.103+02:00</timestamp>
        </messages>
      ▼<release-id>
          <artifact-id>human-tasks-management</artifact-id>
          <group-id>us.dit</group-id>
          <version>1.0.0-SNAPSHOT</version>
        </release-id>
      ▼<resolved-release-id>
          <artifact-id>human-tasks-management</artifact-id>
          <group-id>us.dit</group-id>
          <version>1.0.0-SNAPSHOT</version>
        </resolved-release-id>
        <scanner status="DISPOSED"/>
      </kie-container>
    </kie-containers>
</response>
```


Ilustración 18. Verificación del contenedor kie

Para realizar las pruebas de test y continuar con el desarrollo podría necesitar una serie de recursos que deben introducirse en el servidor FHIR, disponibles en la carpeta (../resources) del repositorio del proyecto https://github.com/tfg-projects-dit-us/Healthcare-Tasks/ .

3.1.2 Modo Depuración

El modo debug, o modo de depuración, es fundamental para detectar y corregir errores en el código de manera eficiente. A través de funciones como la ejecución paso a paso y la inspección de variables, el desarrollador puede analizar el comportamiento del programa en tiempo real y entender su flujo de ejecución. Esto facilita la identificación de fallos lógicos o de funcionamiento que podrían afectar el rendimiento o la estabilidad de la aplicación.

Para ejecutar nuestra aplicación Spring en modo debug usaremos un método conocido como Depuración Remota [1]. Es una técnica avanzada para depurar aplicaciones especialmente útil cuando tratamos con aplicaciones complejas implementadas en varios entornos y que usa JDWP (Java Debug Wire Protocol), un protocolo de comunicación que los depuradores utilizan para comunicarse con la JVM que ejecuta la aplicación. Permite la transmisión de información de depuración a través de una red, lo que hace posible la depuración remota.

3.1.2.1 Configuración

El único paso para configurar el modo depuración es indicar las opciones de JVM para abrir un puerto al que debe conectarse el depurador. Para ello se ha añadido al comando java de ejecución en el archivo `launch-dev.bat` la siguiente configuración:

- `agentlib:jdwp`: Carga el agente JDWP para depurar.
- `transport=dt_socket`: Usa transporte de sockets para la comunicación.
- `server=y`: Configura el JVM para actuar como servidor.
- `suspend=n`: Especifica al JVM que no espere una conexión al modo debug al iniciarse.
- `address=*:5005`: Abre el puerto 5005 para cualquier que se conecte cualquier host(indicado con *).

Será necesario ejecutar la aplicación en modo desarrollador y conectarse a este puerto para realizar la depuración. La forma de conectarse dependerá de su entorno de desarrollo.

3.1.2.2 Conectar con VisualStudioCode

En VisualStudioCode existe la posibilidad de crear un archivo de configuración para depurar la aplicación. Se proporciona el archivo `launch.json` para ejecutar en modo debug, configurado para conectarse a la JVM para depurar. Este fichero debe de estar presente en la carpeta `.vscode`, que a su vez debe ubicarse en el directorio raíz del proyecto. A continuación se explican los ajustes establecidos [2]:

- `"type": "java"`. Define que esta configuración es para proyectos Java.
- `"name": "Attach to Java Debug (port 5005)"`. Nombre visible para la configuración en el menú de depuración en VSCode.
- `"request": "attach"`. Indica que se trata de una conexión a una JVM ya corriendo.
- `"hostName": "localhost"`. Dirección del host donde corre la JVM.
- `"port": 5005`. Puerto de depuración remota al que se conectará VSCode.

Con el modo desarrollador corriendo, ejecutar el modo debug es tan sencillo como ir al menú Run and Debug y seleccionar en el desplegable que aparece junto al botón Start Debugging la opción correspondiente a nuestro archivo de configuración "`attach to Java Debug (port 5005)`". Una vez seleccionado presionamos este botón y el depurador se conectará con nuestra aplicación.

Ahora, debería poder establecer puntos de ruptura, inspeccionar variables y avanzar paso a paso por el código.

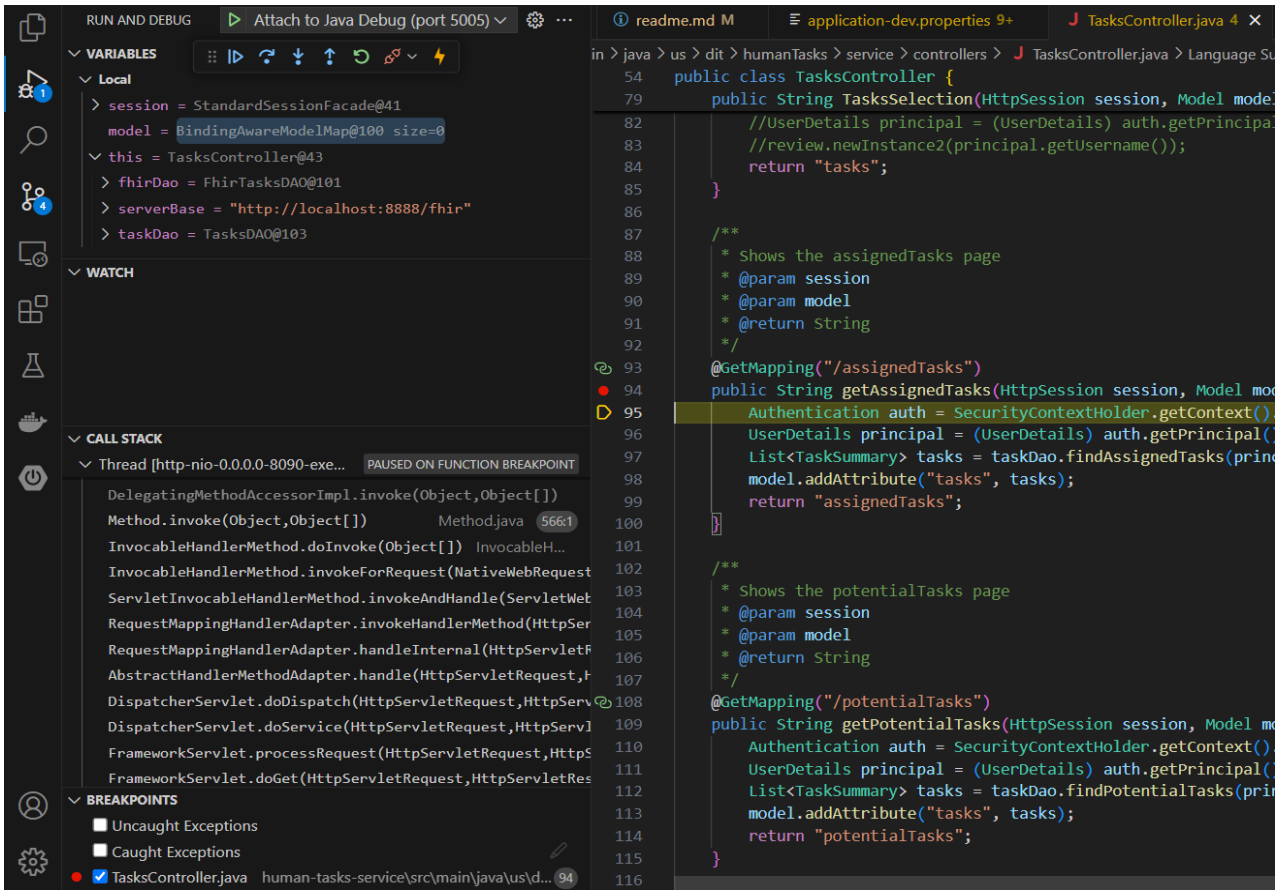

Ilustración 19. Ejecución en modo debug en IDE VisualStudioCode

3.1.2.3 Conectar con Eclipse

Eclipse también ofrece un sólido soporte para la depuración remota [49]:

- Inicia Eclipse y ve al menú Run, luego elige Debug Configurations.
- Haz clic derecho sobre Remote Java Application y selecciona New.
- Asigna un nombre a tu configuración de depuración para distinguirla de otras.
- En la configuración de conexión, ingresa el nombre del host o la dirección IP del servidor donde se está ejecutando la aplicación, y el tipo de conexión “Socket Attach”.
- Ingresa el número de puerto de depuración que configuraste en los argumentos JVM de tu aplicación Spring Boot.
- Guarda la configuración haciendo clic en Apply.
- Para iniciar la sesión de depuración, selecciona la configuración recién creada desde la lista de configuraciones de depuración y haz clic en el botón Debug.

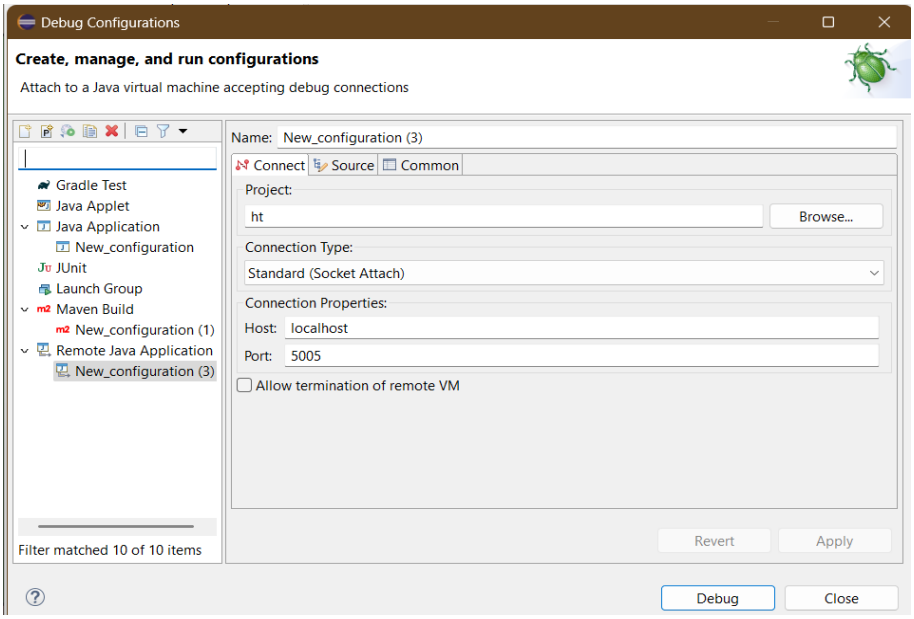

Ilustración 20. Configuración de modo debug en IDE Eclipse.

Eclipse intentará conectarse a la aplicación remota y, una vez conectado, te permitirá usar todo el conjunto de herramientas de depuración que ofrece.

3.2 Arquitectura de la solución

La arquitectura de la aplicación está basada en uno de los patrones más utilizados en el desarrollo de aplicaciones web, el patrón Modelo-Vista-Controlador (MVC) [24]. Que organiza la estructura de la aplicación en tres bloques. Modelo de información(Modelo), presentación (Vista) y lógica de negocio (Controlador). Con esto se busca mejorar la organización y legibilidad del código, facilitar el mantenimiento y favorecer la escalabilidad y robustez de la aplicación.

3.2.1 Modelo

Para el modelo vamos utilizaremos las clases propias de las librerías hapi.fhir en su versión 6.10.0 para los recursos FHIR. Los principales recursos FHIR manejados serán: `Task` [11], `Questionnaire` [12] y `QuestionnaireResponse` [13][14].

Para los recursos jBPM se usan las clases de la librería org.kie.server.api en su versión 7.74.1-Final. Los principales recursos jBPM manejados serán: `TaskSummary` [15], `TaskInstance` [16] y `WorkItemInstance` [17].

3.2.1.1 Relaciones entre recursos

Será de gran relevancia saber la conexión entre las clases de nuestro modelo para determinar la asignación de atributos análogos entre los diferentes recursos que manejamos. Por ello realizaremos una serie de análisis relacionales que detallarán dichas equivalencias:

A continuación, se muestran diagramas de estado para las tareas jBPM y FHIR. La aplicación se encarga de mantener la coherencia de estado entre las entidades FHIR y jBPM correspondientes a la misma tarea.

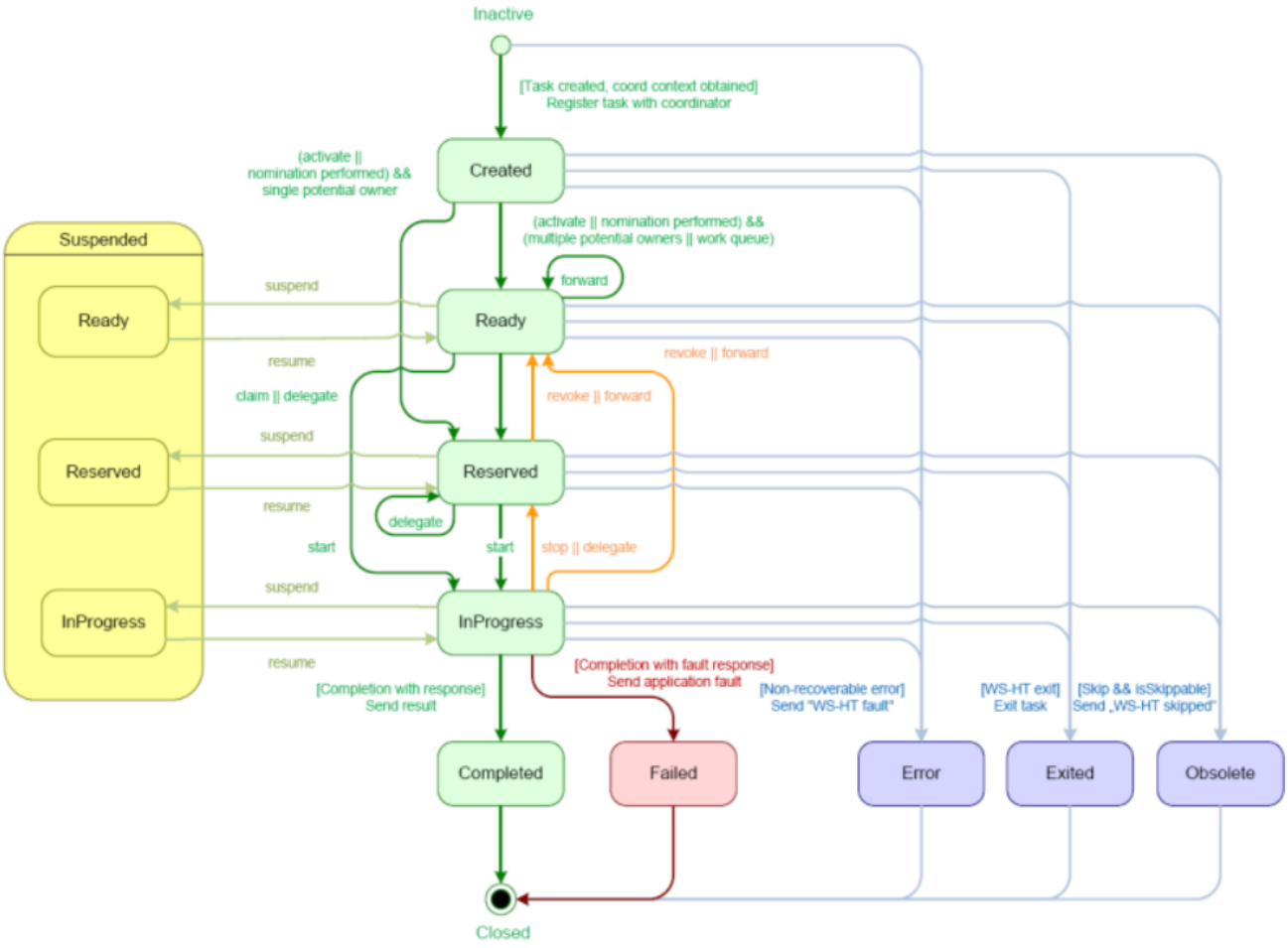

Ilustración 21. Diagrama de estado de tareas jBPM

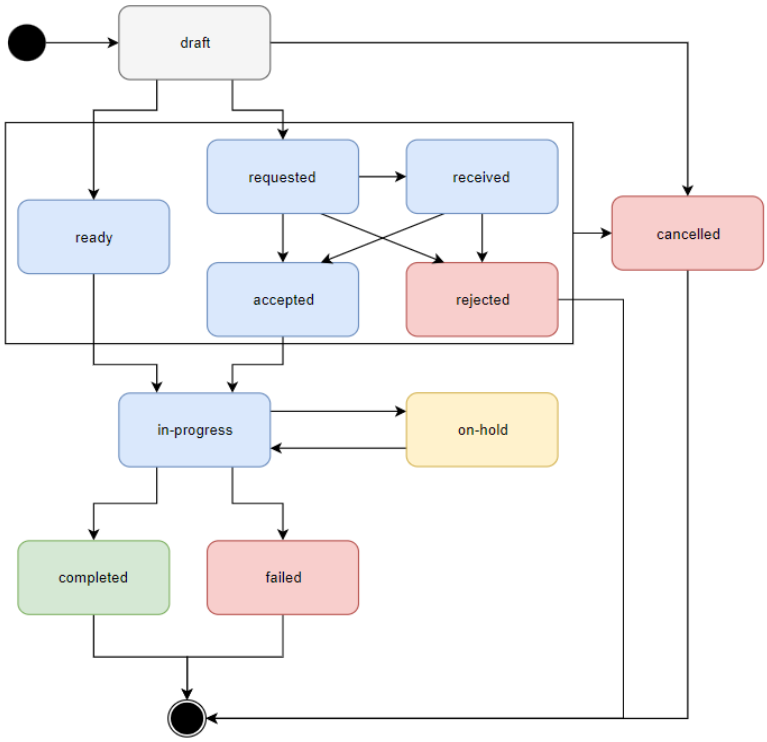

Ilustración 22. Diagrama de estado de tareas FHIR

- Equivalencia de estados entre tareas jBPM y FHIR: se ha buscado cual es la equivalencia entre los principales estados que vamos a usar de los recursos `Task` de jBPM y Task de FHIR, la siguiente tabla muestra el resultado de este análisis:

Estados jBPM	Estados FHIR
Ready	Draft (No ha sido asignada nunca)/Ready (Ha sido rechazada)
Reserved	Requested
In progress	In progress
Completed	Completed
Exited	Failed/Canceled/Rejected

Tabla 1. Equivalencia estados tareas jBPM y FHIR

- Esta tabla representa el análisis de equivalencias realizado entre campos de la definición de tareas humanas en business-central y los atributos de las clases de la API `org.kie.server.api` en su versión 7.74.1-Final de los objetos que representan las tareas jBPM [50]. Se ha usado para dar valor las variables de entrada y campos de la tarea humana en BC a partir de variables de proceso y obtener los atributos de cada instancia TaskSummary en la vista. En el futuro puede dar soporte a los desarrolladores que encuentren dificultades para establecer similitudes entre la definición del modelo de BC y los recursos que representan tareas jBPM.

Descripción de variable	Variables de la tarea jBPM	TaskSummary	TaskInstance	WorkItemInstance
Prioridad	Priority	getPriority()	getPriority()	-
Asunto	Comment	getSubject()	getSubject()	-
Usuario asignado	ActorId	getActualOwner()	getActualOwner()	-
Nombre tarea JBPM	NodeName	getName()	getName()	-
Fecha de expiración	DueDate	getExpirationTime()	getExpirationDate()	-
Fecha de creación	No se puede asignar	getCreatedOn()	getCreatedOn()	-

Estado	No se puede asignar	getStatus()	getStatus()	-
Id tarea JBPM	No se puede asignar	getId()	getId()	-
Id de contenedor KIE	No se puede asignar	getContainer()	getContainer()	-
Id de proceso JBPM	No se puede asignar	getProcessId()	getProcessId()	-
Id de instancia de proceso JBPM	No se puede asignar	getProcessInstanceId()	getProcessInstanceId()	-
Variables de entrada de la tarea JBPM	Todas	No se puede obtener	No se puede obtener	getParameters()

Tabla 2. Equivalencia entre recursos jBPM y la definición de tareas humanas en BC

3.2.2 Vista

La vista usa thymeleaf para generar html dinámico a partir de plantillas, parámetros de entrada y funciones javaScript. Sus acciones están controladas por una serie de controladores y archivos JavaScript que se explicarán más adelante.

-Se han actualizado las siguientes vistas:

- **Pantalla principal (URL: /tasks, plantilla: tasks):** Esta plantilla da acceso al menú principal de la aplicación una vez el usuario ha iniciado sesión con sus credenciales. Previamente existían únicamente dos botones que redirigen al usuario a la pantalla de tareas potenciales y tareas asignadas. Una vez completada una tarea asignada enviando su cuestionario, saltará una alerta en esta pantalla que nos indicará si se ha completado con éxito o ha habido algún error.

  Se ha ampliado la funcionalidad de la aplicación añadiendo una tercera opción a este menú para acceder a la pantalla de tareas completadas por el usuario, de modo que pueda consultar su historial de tareas finalizadas.

- **Pantalla de tareas potenciales (URL: /tasks/pontentialTasks, plantilla: potentialTasks):** En esta pantalla se muestran todas las tareas que pueden ser potencialmente asignables al usuario, marcadas con el estado "Listo". Se mostrarán en una tabla con la siguiente información:

  - Parámetros de entrada: La plantilla recibirá como parámetro un mapa de tareas, cuya clave hace referencia al índice que identifica al servidor KIE en la lista de servidores gestionados por la aplicación. Los valores de este mapa son listas de objetos `TaskSummary` pertenecientes a cada servidor. Mediante este objeto se obtendrán los datos de cada tarea y se mostrarán al usuario.

  - Parámetros de salida: Devueltos al ejecutar peticiones post asociadas a las acciones de una tarea específica. Devolverá los ids de la tarea, del contenedor y de la instancia del proceso jBPM. Además del índice del servidor kie donde se encuentra almacenada la tarea, necesario para poder saber dónde buscarla.

- Datos:
  - Proceso jBPM al que pertenece la tarea.
  - Nombre de la tarea.
  - Fecha de creación de la tarea.
  - Fecha de expiración [nuevo]: que orientará al usuario sobre el tiempo que dispone para reclamar y completar esa tarea.
  - Prioridad [nuevo]: que servirá como indicador de la urgencia que requiere la tarea (Integer que se formatea y se muestra al usuario como "Leve", "Media" o "Urgente").
  - Asunto [nuevo]: sirve como clasificador de las tareas en distintas categorías.
- Acciones:
  - Reclamar: el usuario podrá reclamar esta tarea para completarla pulsando el botón reclamar, lo que mandará una solicitud a `/tasks/claim` del `TasksController` y cambiará el estado de la tarea FHIR y jBPM de "Lista" a "Solicitada" y "Reservada" respectivamente y asignarle el usuario con el que se ha realizado la acción. Por tanto una vez asignada esta tarea se eliminará de este menú para todos los usuarios potenciales y se mostrará en la pantalla de tareas asignadas del usuario que la reclamó.
  - Filtrar [nuevo]: se trata de un selector que permite filtrar las tareas potenciales por los campos de nombre de tarea, un rango de fecha de creación o expiración, prioridad y asunto. Cuenta con un selector para indicar la columna de filtrado e inputs específicos según el campo seleccionado. Esta acción está controlada por el javaScript `filterAction` que se explicará más adelante.
  - Ordenar [nuevo]: botón que ordenará de forma ascendente (▲) o descendente (▼) las entradas de la tabla cada vez que se pulse en la cabecera de una columna de datos. Todos los campos de datos son ordenables. Esta acción está controlada por el javaScript `orderAction` que se explicará más adelante.

- **Pantalla de tareas asignadas (URL: /tasks/assignedTasks, plantilla: assignedTasks):** Pantalla donde se muestran todas las tareas jBPM asignadas al usuario. Es decir, las que se encuentran en los estados "En progreso" o "Reservado". Se mostrarán en una tabla con la siguiente información:
  - Parámetros de entrada: La plantilla recibirá como parámetro un mapa de tareas, cuya clave hace referencia al índice que identifica al servidor KIE en la lista de servidores de la aplicación. Los valores de este mapa son listas de objetos `TaskSummary` pertenecientes a cada servidor. Mediante este objeto se obtendrán los datos de cada tarea y se mostrarán al usuario.
  - Parámetros de salida: Devueltos al ejecutar peticiones post asociadas a las acciones de una tarea específica. Devolverá los ids de la tarea, del contenedor y de la instancia del proceso jBPM. Además del usuario al que está asignada la tarea y el índice del servidor kie donde se encuentra almacenada, necesario para perder saber dónde buscarla.
  - Datos:
    - Proceso jBPM al que pertenece la tarea.
    - Nombre de la tarea.
    - Fecha de creación de la tarea.
    - Estado de la tarea.
    - Fecha de expiración [nuevo]: que orientará al usuario sobre el tiempo que dispone para reclamar y completar esa tarea.
    - Prioridad [nuevo]: que servirá como indicador de la urgencia que requiere la tarea (Integer que se formatea y se muestra al usuario como "Leve", "Media" o "Urgente").
    - Asunto [nuevo]: sirve como clasificador de las tareas en distintas categorías.

- Acciones:
  - Comenzar: el usuario podrá empezar una tarea y rellenar su formulario pulsando el botón comenzar, lo que mandará una solicitud manejada por `TasksController` a `/tasks/start` y cambiará el estado de la tarea FHIR y jBPM de "Solicitada" y "Reservada" respectivamente a "En progreso". Por tanto una vez comenzada una tarea solo se podrá rechazar o continuar. Para obtener el formulario se obtiene el id de la tarea FHIR a partir de una variable del proceso jBPM y luego se obtiene el id del cuestionario FHIR asociado al id de la tarea.
  - Continuar: se podrá continuar la tarea y rellenar su formulario con el progreso guardado presionando el botón continuar, solo en caso de que se haya comenzado previamente. Esa acción envía una solicitud a `/tasks/continue` y hace lo mismo que `/tasks/start` solo que no cambia ningún estado FHIR ni jBPM.
  - Rechazar: es posible rechazar una tarea asignada pulsando el botón rechazar. Esto envía una petición a `/tasks/reject` qué actualizará el estado de la tarea FHIR de vuelta a "Listo" y de la tarea jBPM a "Creada". Además se le desasocia el usuario asignado, así otro usuario o el mismo se la podrá asignar en otro momento.. En consecuencia la tarea desaparecerá de la pantalla y volverá a mostrarse en la pantalla de tareas potenciales.
  - Filtrar [nuevo]: este selector permitirá filtrar las tareas asignadas al usuario por los campos de nombre de tarea, un rango de fecha de creación o expiración, prioridad, asunto y estado. Cuenta con un selector para indicar la columna de filtrado e inputs específicos según el campo seleccionado. Esta acción está controlada por el javaScript `filterAction` que se explicará más adelante.
  - Ordenar [nuevo]: botón que ordenará de forma ascendente (▲) o descendente (▼) las entradas de la tabla cada vez que se pulse en la cabecera de una columna de datos. Todos los campos de datos son ordenables. Esta acción está controlada por el javaScript `orderAction` que se explicará más adelante.

- **Pantalla del cuestionario de cierre de una tarea (URL: /questionnaire, plantilla: questionnaireForm):** Esta pantalla genérica se encarga de representar las preguntas de cualquier tipo de recurso `Questionnaire` [12] de FHIR. Una vez se complete el cuestionario se podrá pulsar el botón de "Enviar cuestionario". Se realiza un Post a la URL "`/submit`" controlada por el controlador `FormController`.

  A esta plantilla únicamente se ha añadido un parámetro extra de salida al hacer la petición Post, representativo del índice del servidor KIE donde se encuentra alojada la tarea asociada con el cuestionario.

-Se han desarrollado las siguientes vistas:

- **Pantalla de tareas completadas (URL: /tasks/completedTasks, plantilla: completedTasks):** Esta nueva funcionalidad se basa en una plantilla donde se muestran todas las tareas jBPM completadas por el usuario. Es decir, las que ha completado previamente enviando su respuesta del formulario y poseen el estado "Completado". Se mostrarán en una tabla con la siguiente información:
  - Parámetros de entrada: La plantilla recibirá como parámetro un mapa de tareas, cuya clave hace referencia al índice que identifica al servidor KIE en la lista de servidores de la aplicación. Los valores de este mapa son listas de objetos `TaskSummary` pertenecientes a cada servidor. Mediante este objeto se obtendrán los datos de cada tarea y se mostrarán al usuario.
  - Parámetros de salida: Devueltos al ejecutar peticiones post asociadas a las acciones de una tarea específica. Devolverá los ids de la tarea, del contenedor y de la instancia del proceso jBPM. Además del usuario al que está asignada la tarea y el índice del servidor kie donde se encuentra almacenada, necesario para perder saber dónde buscarla.
  - Datos:

        - Proceso jBPM al que pertenece la tarea.
        - Nombre de la tarea.
        - Fecha de creación de la tarea.
        - Fecha de entrega de la tarea.
        - Prioridad: que servirá como indicador de la urgencia que requiere la tarea (Integer que se formatea y se muestra al usuario como “Leve”, “Media” o “Urgente”).
        - Asunto: usado como clasificador de las tareas en distintas categorías.

    - Acciones:

        - Ver: esta acción manda una petición al controlador `TasksController` a la URL `/tasks/view`. Permite ver las respuestas del formulario asociado a una tarea completada a partir de un recurso FHIR quiestionnaireResponse. Este recurso es generado al enviar un recurso questionnaire al completar una tarea. Dado que cada tarea completada posee una respuesta única se obtendrá el id del cuestionario respuesta FHIR a partir del campo output de la tarea FHIR asociada. Por último se redirigirá a la URL `/questionnaireResponse` manejada por `QuestionnaireResponseController` para mostrar el formulario respuesta. Dado que una tarea completada siempre permanecerá en estado “Completado”, los estados FHIR y jBPM se mantienen.

        - Filtrar: un selector que permitirá filtrar las tareas completadas por el usuario por los campos de nombre de tarea, un rango de fecha de creación o entrega, prioridad y asunto. Cuenta con un selector para indicar la columna de filtrado e inputs específicos según el campo seleccionado. Esta acción está controlada por el javaScript `filterAction` que se explicará más adelante.

        - Ordenar: botón que ordenará de forma ascendente (▲) o descendente (▼) las entradas de la tabla cada vez que se pulse en la cabecera de una columna de datos. Todos los campos de datos son ordenables. Esta acción está controlada por el javaScript `orderAction` que se explicará más adelante.

- **Pantalla del cuestionario de respuesta de una tarea (URL: /questionnaireResponse, plantilla: questionnaireResponseForm):** Es una plantilla genérica capaz de mostrar cualquier objeto `QuestionnaireResponse` FHIR que se le pase. Sin embargo la plantilla no maneja directamente este objeto, si no que se recibe un mapa cuyas claves son las preguntas y sus valores corresponden con una lista `String` de respuestas formateadas por el controlador independientemente del tipo de cada `Answer` del `QuestionnaireResponse`. Esta plantilla es simple ya que se limita a recorrer el mapa e imprimir las preguntas con sus respuestas.

  También se recibe como parámetro de la plantilla el id del cuestionario respuesta obtenido a partir del campo output de la tarea FHIR.

- Para dar dinamismo a las acciones de filtrado y ordenación de tareas, se ha optado por ejecutar código JavaScript en la parte del cliente frente a hacer peticiones al controlador de Spring, principalmente por dos razones. En primer lugar, proporciona una interacción inmediata, ya que no hay que esperar que el servidor procese la solicitud y recargue la página. Por último, reduce la carga en el servidor, debido a que las tareas permanecen cargadas en el navegador. A continuación, se explica la lógica de funcionamiento de estos archivos:

- **OrderAction:** Archivo JavaScript que controla las acciones de ordenación para la vista de tareas potenciales, asignadas y completadas [18]. Tiene un único método:

    - `OrderByColumn`: Ordena las filas de la tabla según la columna seleccionada, que es recibida como parámetro, al hacer clic en un encabezado alternando entre orden ascendente y descendente. En caso de que el usuario vuelva a hacer clic en la misma columna, se cambia la dirección del orden. De lo contrario si se hace clic en una columna distinta, se actualiza la columna actual y se establece el orden inicial como ascendente.

Esta función recopila todas las filas de la tabla en un array para poder ordenarlas y obtiene el texto de la celda correspondiente a la columna seleccionada para cada fila y la dirección del orden. Cuando el contenido de las celdas puede interpretarse como una fecha válida, se ordenarán como fechas. Si no, se ordenan como texto. Posteriormente se borra el contenido original de la tabla y se vuelve a insertar cada fila en el nuevo orden.

Por último controla la eliminación de los indicadores visuales de orden de otras columnas no seleccionadas y actualiza el indicador (▲ o ▼) de la columna actual cada vez que se hace click en ella.

- **FilterAction:** Archivo JavaScript que controla las acciones de filtrado para la vista de tareas potenciales, asignadas y completadas [19]. Cuenta con dos funciones que explicaremos a continuación:
    - `ToggleFilterInputs`**:** Muestra u oculta los inputs de filtrado específicos de cada campo según la columna de filtrado seleccionada. Si se quiere filtrar por fecha de creación o expiración muestra inputs de fecha y hora para seleccionar un intervalo de tiempo. Si es por prioridad o estado muestra sus selectores con las opciones correspondientes. En el caso de las columnas de nombre y asunto muestra un input de texto. Además, llama a `filterTable` automáticamente para aplicar el filtro cada vez que cambia el criterio.
    - `FilterTable`**:** Filtra las filas de la tabla mediante un bucle en función del criterio seleccionado por el usuario. (`filterColumn`) determina el tipo de filtro a aplicar. Cada fila quedará marcada con un booleano que indicará si debe mostrarse o no.
        - Filtrado por texto (`textFilterInput`): busca coincidencias de texto en las columnas de Nombre o Asunto.
        - Filtrado por fecha y hora (`startDate`, `endDate`): filtra por rango de fechas y horas en las columnas de Fecha de expiración o creación.
        - Filtrado por selector de prioridad o estado (`prioritySelect`, `statusSelect`): filtra solo si el texto en la celda coincide con la opción seleccionada.

      Finalmente se actualiza el estilo de cada fila (`row.style.display`) para mostrar u ocultar según si cumple las condiciones.

3.2.3 Controlador

El controlador se encarga de gestionar las peticiones HTTP, procesar los datos y devolver la vista adecuada.

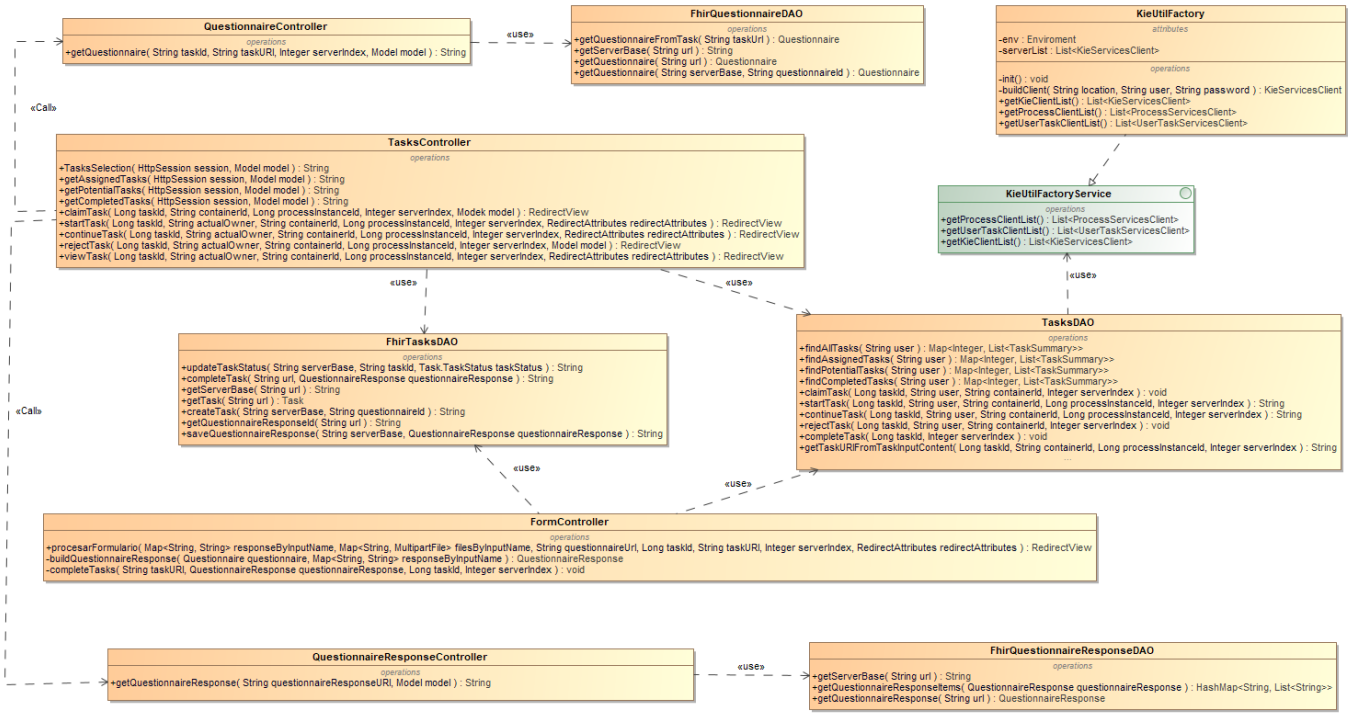

Ilustración 23. Diagrama de clases del controlador

-Se han actualizado los siguientes controladores:

- **TasksController:** Este controlador se encarga de todo lo relacionado con tareas, tanto FHIR como jBPM, haciendo uso de los servicios `FhirTasksDAO` y `TasksDAO`. Se han añadido dos nuevas URLs a las que atiende:

    - `/tasks/completedTasks`: hace uso del servicio `TasksDAO` para obtener un mapa que relaciona cada servidor KIE con una lista de tareas jBPM que contiene, con estado "Completado" y asociadas a un usuario. Devuelve la vista `completedTasks` pasando el mapa de tareas como parámetro.

    - `/tasks/view`: utiliza el servicio `FhirTasksDAO` para obtener la uri del cuestionario respuesta a partir del campo output de una tarea FHIR, buscando previamente el uri de la tarea jBPM a partir de un id y el servidor KIE correspondiente. Posteriormente, nos redirecciona al controlador `/questionnaireResponse`, pasando la uri del cuestionario respuesta como parámetro.

  También se han modificado las URLs existentes `/tasks/potentialTasks` y `/tasks/assignedTasks` para que funcionen con múltiples servidores KIE a partir de un mapa, de forma análoga a `/tasks/completedTasks`. De la misma manera, se han actualizado las URLs `/claim`, `/start`, `/continue`, `/reject` para que reciban como parámetro adicional el índice del servidor KIE donde se debe buscar la tarea para ejecutar la acción asociada.

- **FormController:** Este controlador se ocupa exclusivamente de la URL `/submit`. Es responsable de procesar el formulario enviado desde la vista `questionnaireForm` y completar las tareas FHIR y jBPM ligadas a ese formulario. Se han realizado unos cambios básicos en varios métodos:

    - `procesarFormulario`: Se ocupa exclusivamente las peticiones de la url `/submit`. Obtiene el recurso `Questionnaire` y construye el `QuestionnaireResponse` a partir de este y las respuestas recibidas al enviar el formulario respuesta. Para lograrlo hace uso del servicio `FhirQuestionnaireDAO`. Recibirá un nuevo parámetro, el índice del servidor KIE que contiene la tarea a completar y se lo pasará al método que describiremos a continuación.

    - `completeTasks`: Usa los servicios `FhirQuestionnaireDAO` y `TasksDAO` para finalizar y cambiar el estado de las tareas FHIR y jBPM a "Completado". Recibe un parámetro de entrada

extra que es el índice del servidor KIE donde buscar la tarea jBPM que se debe completar.

- **TasksDAO:** Encargado de comunicarse con los servidores KIE a través del servicio `KieUtilFactoryService` y de gestionar las tareas jBPM que contienen. Recupera tareas humanas y ejecuta acciones sobre estas, cambiando sus estados.

  -Se han modificado los métodos:
  - `FindAllTasks`: Este método recibe el nombre de usuario y devuelve contenidas en un mapa todas las tareas jBPM potencialmente asignables y asignadas con fecha de expiración válida, además de todas las completadas. Es decir con estado "Listo", "Reservado o En progreso" y "Completado" respectivamente para un usuario determinado. Se buscarán las tareas en todos los servidores KIE.

  - `FindPotentialTasks`: Este método recibe el nombre de usuario devuelve dentro de un mapa las tareas jBPM potencialmente asignables para este, buscando en todos los servidores KIE. Hace uso del método `findAllTasks`, para luego filtrar por las tareas que no tienen usuario asignado en el campo `ActualOwner`. Es decir, devolviendo únicamente las tareas que no están asignadas a ningún usuario, pero pueden asignarse al usuario pasado por parámetro.

  - `FindAssignedTasks`: Recibe el nombre de usuario para el que deseemos obtener en forma de mapa todas las tareas jBPM asignadas a dicho usuario en todos los servidores KIE. Hace uso del método `findAllTasks`, utilizando la clase `Stream` [23] para filtrar las tareas que tienen en el campo `ActualOwner` el usuario indicado por parámetro y campo `Status` distinto de "Completado". Es decir, han sido asignadas a un usuario pero no completadas.

  - `CompleteTask`: este método se llama desde la URL `/submit` para completar una tarea jBPM, marcándola con estado"Completado". Recibe dos parámetros, el primero es el id de la tarea jBPM, a partir del cuál se obtiene el id del contenedor y el usuario asociado necesarios para completar la tarea. El segundo es el índice del servidor KIE, que usando el servicio `KieUtilFactoryService` obtendrá el cliente correspondiente de la lista de clientes KIE definidos. Por último, se actualiza la variable `ExpirationDate` de la tarea, que una vez completada será representativa de la fecha de entrega de la tarea. Por último se completa la tarea.

  -Se han añadido los métodos:
  - `FindCompletedTasks`: Devuelve un mapa que contiene listas de tareas jBPM completadas, pertenecientes a cada servidor KIE, para el usuario pasado como parámetro. Usa el método `findAllTasks` y la clase `Stream` [23] para filtrar todas las tareas donde el campo `ActualOwner` coincida con el usuario pasado por parámetro y `Status` igual a "Completado". Es decir, han sido asignadas y completadas por el usuario.

  - `ViewTask`: Este método se llama desde la URL `/view` para obtener la URI del cuestionario respuesta asociado a una tarea jBPM. Para ello la función obtiene los parámetros de salida de la tarea jBPM a partir de su id y busca la variable de salida `questionnaireResponseURI` para posteriormente devolverla.

  Para el resto de métodos de la clase, en concreto `claimTask`, `rejectTask`, `continueTask`, `startTask` y `getTaskURIFromTaskInputContent`, lo único que se ha modificado es la forma en la que obtiene el cliente KIE. Para ello se usará el servicio desarrollado `KieUtilFactoryService` para obtener la lista de clientes KIE a los que nuestra aplicación se conecta, y posteriormente escoger únicamente el que se encuentra en la posición indicada por el índice de servidor que reciben estas funciones como parámetro de entrada.

- **FhirTasksDAO:** Encargado de comunicarse con el servidor FHIR. Se ha modificado únicamente el siguiente método:

  - `CompleteTask`: este método se llama desde la URL `/submit` para completar la tarea FHIR. Recibe dos parámetros, el primero la url de la tarea FHIR para obtener el recurso `Task` asociado, necesario para marcar su estado como "Completado" y añadir al campo output de la tarea la uri del cuestionario respuesta, de forma que sea recuperable desde aquí. El segundo es un objeto

`QuestionnaireResponse` que persistirá en el servidor FHIR, esto generará un id correspondiente a este cuestionario que será devuelto por la función.

-Se han creado dos métodos nuevos:

- `createTask`: esta función crea desde cero un recurso `Task` FHIR, añadiendo al campo input la etiqueta "closingQuestionnaire", junto con el valor del id del recurso `Questionnaire` de cierre asociado a la tarea, e inicializando su estado en "Listo". Posteriormente persiste el recurso en el servidor FHIR especificado, lo que genera un id único para esta tarea que devolveremos como resultado. Se utiliza exclusivamente en los controladores de test que detallaremos más adelante para crear las tareas de prueba.

- `getQuestionnaireResponseId`: Es de vital importancia, ya que es la encargada de obtener y devolver el valor del id del recurso `QuestionnaireResponse` presente en el campo output de una la tarea FHIR. Para acceder a este campo, se debe obtener el recurso FHIR `Task` asociado al id de tarea que recibe este método a la entrada.

-Se han desarrollado los siguientes controladores:

- **KieUtilFactory:** Implementa la interfaz `KieUtilFactoryService`, que ofrece todos los servicios de conectividad de la aplicación con los servidores KIE. Esta clase es auto instanciada al ejecutar el método `init`, al terminar de iniciarse la aplicación. En ese momento se creará una lista de clientes KIE adheridos en orden (siempre que se consiga establecer una conexión con ellos), de modo que se puedan identificar por su índice en la lista. A continuación explicaremos los métodos que ofrece la interfaz y que son implementados aquí:

  - **getKieClientList:** Devuelve una lista de `KieServicesClient` [22], es decir, de servidores presentes en nuestra configuración con los que se ha conseguido conectar. Este objeto representa la interfaz principal para interactuar con un servidor KIE. Se utiliza para obtener otros clientes especializados, como `UserTaskServicesClient` y `ProcessServicesClient`. Además, mostrará en las trazas de la aplicación, el índice asignado a cada servidor, que será la posición que ocupa en esta lista. No tiene por qué coincidir con el orden definido en la configuración del fichero `application.properties`, ya que puede fallar la conexión con alguno.

  - **getUserTaskClientList:** Devuelve una lista de `UserTaskServicesClient` [21] de servidores con los que se ha conseguido conectar. Este objeto es un cliente especializado para manejar tareas humanas en jBPM. Usado por el controlador `TasksDAO`.

  - **getProcessClientList:** Devuelve una lista de `ProcessServicesClient` [20] de servidores con los que se ha conseguido conectar. Este objeto es un cliente especializado en manejar procesos en jBPM. Usado por las clases de test para crear procesos jBPM.

  - **buildClient:** Construye un cliente en caso de que se logre establecer una conexión a partir de los parámetros de conexión configurados en el fichero `application.properties`, que son el usuario, la contraseña, y la localización del servidor KIE (su URI). Es un método privado, por lo que no se define en la interfaz.

- **QuestionnaireResponseController:** Este controlador se ocupa de todas las peticiones relacionadas con cuestionarios de respuesta invocadas desde la url /`questionnaireResponse` . Aquí se obtiene un objeto FHIR `QuestionnaireResponse` a partir de su URI haciendo uso de los servicios ofrecidos por `QuestionnaireResponseDAO`. Esta URI es obtenida como hemos dicho previamente desde el output de la tarea FHIR.

  Por último se devuelve la plantilla `questionnaireResponseForm` pasando como parámetros de entrada la URI del cuestionario respuesta y un mapa que contiene las preguntas, cada una asociada a una lista de respuestas. Este mapa se obtiene a partir del objeto `questionnaireResponse`.

- **FhirQuestionnaireResponseDAO:** Servicio que devuelve los datos que necesita el controlador `QuestionnaireResponseController`. Contiene tres métodos principales**:**

- `getQuestionnaireResponse`**:** se encarga de obtener el objeto `QuestionnaireResponse` a partir del cliente FHIR y la URI del cuestionario respuesta**.**
- `getServerBase`**:** devuelve la url del servidor FHIR a partir de la uri del cuestionario respuesta.
- `getQuestionnaireResponseItems`**:** devuelve un mapa ordenado con las preguntas y respuestas del `QuestionnaireResponse` formateadas. Para ello es crucial saber cómo se construye este tipo de objeto a partir de un objeto FHIR `Questionnaire`. Esto se realiza en el método `buildQuestionnaireResponse` del controlador `FormContoller`.

  Es necesario obtener el campo `Item` del `QuestionnaireResponse` que contiene todas las respuestas y preguntas como una lista del objeto `QuestionnaireResponseItemComponent` y recorrerlo con un bucle. A su vez, dado que una pregunta puede contener más de una respuesta, debemos obtener el campo `Answer` como una lista del objeto `QuestionnaireResponseItemAnswerComponent` a partir de un objeto `QuestionnaireResponseItemComponent` y recorrerlo para obtener el tipo de cada respuesta y darle el formato que necesitemos. Los tipos de datos de las respuestas que maneja cada tipo de `Answer` del `QuestionnaireResponse` se especifican en el apartado [2.4].

3.3 Verificación del desarrollo

En este apartado definiremos las bases para poner en funcionamiento y probar la aplicación.

3.3.1 Procesos de test

Este es el modelo de procesos o kjar que cargaremos en los servidores KIE y que contiene los procesos jBPM que maneja nuestra aplicación para hacer pruebas. A continuación explicaremos en profundidad las mejoras introducidas, en que se diferencian los distintos procesos usados, su configuración de variables, cómo es su flujo de ejecución y su configuración en la aplicación:

- **Tipos de procesos:**
  - `TareaARoleLeve/TareaARoleMedia/TareaARoleUrgente`: Son procesos análogos entre sí pero con distintos niveles de prioridad asignados a las tareas humanas que contienen, ya que no se permite asignación de la prioridad de una tarea desde una variable de proceso. Estos procesos inician su tarea sin un usuario propietario definido y es por eso que aparecerán como tareas potenciales asignables para un grupo de usuarios ligados al rol webadmin, de modo que cualquier usuario con este rol podrá reclamarla y ejecutarla.
  - `TareaAUsuario`: También análogo a los procesos previos, con la diferencia de que inician su tarea con un usuario propietario definido y es por eso que aparecerán como tareas asignadas para el usuario que cree la instancia.
- **Variables:** Es importante saber diferenciar entre variables del proceso y de la tarea jBPM, ya que pertenecen a contextos distintos, por lo que una modificación de una variable de tarea no afectará a una de proceso y viceversa, a no ser que copiemos su valor con asignaciones de datos al otro contexto. La tarea jBPM tiene campos predefinidos que son automáticamente generados, aunque también se definirán nuestras propias variables de tarea. Por el contrario, en el proceso todas sus variables se deben definir siempre.

  El modelo cuenta únicamente con variables de entrada recibidas al iniciar el proceso desde el controlador de test. Posteriormente, se pasan como asignación de datos de entrada a la tarea jBPM para definir su `DataInputSet`. Es decir estos datos se copiarán del contexto del proceso al contexto de la tarea. Son las siguientes [6]:

Variable de proceso	Variable de tarea	Características	Descripción	Tipo
taskURI	taskURI	• Definida por usuario de BC. • Permite mapeo de entrada desde variable de proceso.	URI de la tarea FHIR asociada a la tarea jBPM	String
user	ActorId	• Exclusiva del proceso de prueba `TareaAUsuario`. • Predefinida. • Permite mapeo de entrada desde variable de proceso.	Usuario propietario de la tarea.	String
p_DueDate	DueDate	• Predefinida [8]. • Añadido en esta versión • Permite mapeo de entrada desde variable de proceso.	Fecha de expiración de la tarea en formato ISO8601.	String
p_Subject	Comment	• Predefinida. • Añadido en esta versión • Permite asignación por interfaz desde variable de proceso.	Cadena de texto representativa del asunto de una tarea jBPM	String
No permite asignación	Priority	• Predefinida. • Añadido en esta versión • No permite mapeo de entrada ni asignación por interfaz desde variable de proceso.	Un número que indica la prioridad de la tarea jBPM	Integer

Tabla 3. Equivalencia entre variables de entrada de proceso y de tarea en BC

- **Flujo de ejecución:** Tras el evento de inicio se da valor a las variables de entrada del proceso y posteriormente se imprime una traza de entrada que notifica del inicio del proceso.

  Inmediatamente después, inicia la tarea jBPM dando valor a sus variables de entrada. El flujo del proceso se detendrá hasta que la tarea se complete o el temporizador de la tarea se ejecute. En caso de que sea completada, se notifica que se ha finalizado la tarea mediante la traza de salida y se llega al evento de fin del proceso.

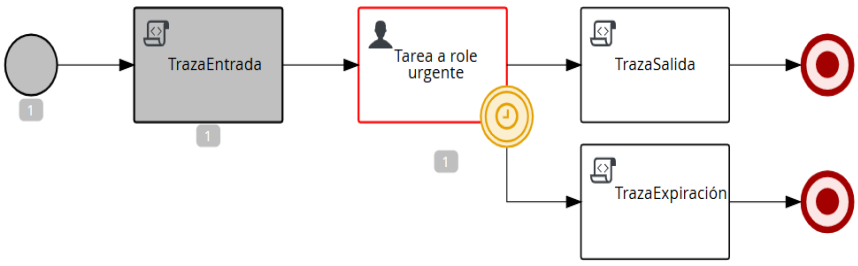

Ilustración 24. Proceso de test iniciado

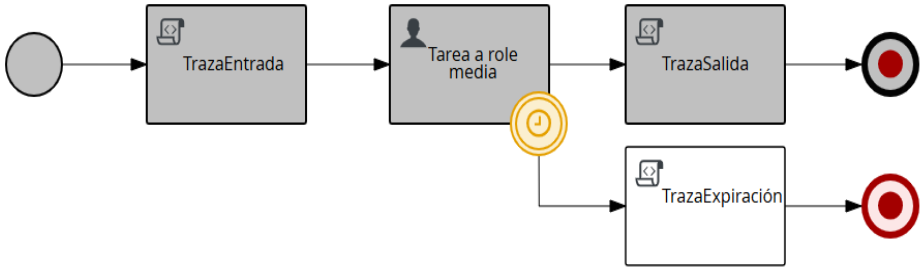

Ilustración 25. Proceso de test completado

Como hemos mencionado, la tarea tiene configurado un temporizador [5] para que se ejecute una única vez en una fecha específica. Esta fecha se obtiene como parámetro de entrada desde la variable de proceso `p_DueDate` y debe estar en formato ISO8601. El timer detiene el flujo de su rama de salida hasta que se alcance la fecha de expiración. Una vez alcanzada la fecha, continuará el flujo por esta rama, se notificará de que el proceso ha expirado mediante una traza y se llegará al evento de fin del proceso.

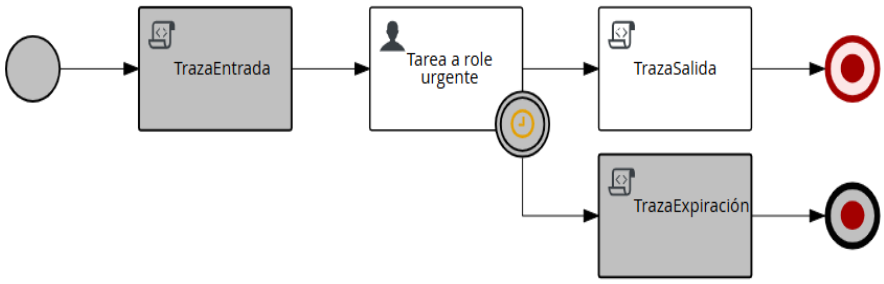

Ilustración 26. Proceso de test expirado

El evento de fin es terminante [4], lo que significa que cuando se alcanza este nodo en cualquier rama, se termina el proceso y todas las demás ramas que puedan seguir activas en paralelo. Es decir, si se completa una tarea (estado "Completado") antes de que se ejecute el temporizador, éste se cancela. De lo contrario si se ejecuta el temporizador antes de que se complete una tarea, ésta finaliza marcándose con estado "Cerrado", lo cuál implica que la tarea ha expirado. Éste es un estado final, por lo que si se alcanza, no puede cambiarse a ningún otro estado, y la tarea no será recuperable.

- **Configuración:** Configuración de fichero `application.properties`. Donde definiremos los procesos de test que usará nuestra aplicación, indicando su id de proceso.

```
#nombre del proceso que asigna una tarea al usuario que lo inicia
test.userprocess=HumanTasksManagement.TareaAUsuario
#nombre del proceso que contiene una tarea asignada al role wbadmin
test.roleprocessHigh=HumanTasksManagement.TareaARolUrgente
test.roleprocessMedium=HumanTasksManagement.TareaARolMedia
test.roleprocessLow=HumanTasksManagement.TareaARolLeve
```

Ilustración 27. Configuración de procesos de test

3.3.2 Controladores de test

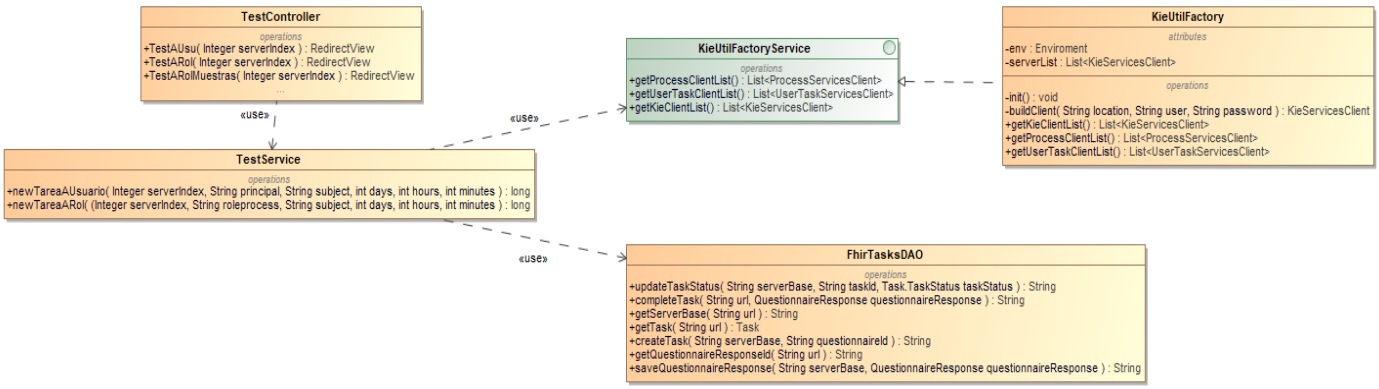

Ilustración 28. Diagrama de clases de test

- **TestController:** Controlador que crea los procesos jBPM de muestra que usaremos para probar el funcionamiento de nuestra aplicación, usando el servicio `TestService`. Se han realizado cambios en las URLs que atiende este fichero conforme a los cambios en el modelo de los procesos jBPM:
    - `/test/initTareaARol`: inicia una tarea a rol con parámetros configurables de prioridad (según el nombre de proceso), asunto y fecha de expiración. Esta tarea será potencialmente asignables para un grupo de usuarios. Por último nos redirecciona a la pantalla principal de `/tasks`.
    - `/test/initTareaAUsuario`: crea una tarea a usuario con parámetros configurables de usuario, prioridad (según el nombre de proceso), asunto y fecha de expiración. Esta tarea será asignada al usuario especificado. Finalmente nos redirige a la vista de `/tasks`.
    - `/test/initTareasARolMuestra`: tiene el mismo comportamiento que `/initTareaARol` con la diferencia de que inicia varios procesos jBPM de muestra, con diferentes valores de prioridad, asunto y fecha de expiración.

  Cuando invoquemos estos procesos se les debe pasar como parámetro de entrada el índice del servidor kie donde queremos que se instancien, de la forma `/test/peticion?serverIndex=x`.

- **TestService:** Usa los servicios de `FhirTasksDAO` para crear tareas FHIR y `KieUtilFactoryService` para crear procesos jBPM en un servidor concreto. Se han realizado cambios en sus métodos para adaptar el código al nuevo modelo de procesos jBPM explicado en el apartado [3.3.1]:
    - `newTareaARol`: inicia un proceso jBPM en un servidor KIE concreto a partir del id del contenedor KIE, un nombre de proceso perteneciente a nuestro modelo kjar y un mapa con las variables del proceso. Entre estas variables se encuentran el asunto y la fecha de expiración, obtenidos a través de los parámetros de entrada.

      También se pasa como variable la uri de la tarea FHIR que se asociará a la tarea jBPM. En versiones anteriores esta URI era obtenida a través de la configuración de nuestra aplicación. Sin embargo esto generaba que todos los procesos y tareas jBPM creados tuvieran asociados la misma tarea FHIR, lo que provocaba que un recurso `Task` FHIR tuviera varias referencias a recursos `QuestionnaireResponse` en el campo output al completar una tarea jBPM. En consecuencia, era imposible obtener el cuestionario respuesta asociado a una tarea jBPM. Es por esto que en esta versión se ha desarrollado el método `createTask` del servicio `FhirTasksDAO` para crear un nuevo recurso de tarea FHIR único asociada a cada tarea jBPM y generar una una URI exclusiva que pasar al proceso jBPM.
    - `newTareaAUsuario`: mismo comportamiento que newTareaARol solo que al mapa de variables del proceso se le añade el usuario recibido también como parámetro.

3.3.3 Servidores KIE

Para probar que nuestra aplicación puede conectarse a varios motores de procesos KIE, levantaremos dos servidores KIE. Detallaremos los pasos a seguir para ponerlos en funcionamiento, y la configuración necesaria en nuestra aplicación para conectarnos a dichos motores externos.

La solución está implementada para que levante un servidor KIE embebido al iniciar la aplicación. Es decir, ya disponemos de un servidor KIE simplemente ejecutando la aplicación. El cuál será el primero al que nos conectaremos. Será accesible desde la URL http://localhost:8090/rest/server.

El segundo servidor que usaremos es el servidor remoto que ofrece por defecto Business Central [7]. Es un servidor de test provisional para hacer las pruebas, por lo que deberá desaparecer en la versión de producción. Para levantarlo, primero deberemos desplegar el Entorno de Desarrollo jBPM, acorde con los pasos especificados en el apartado [3.1.1.1]. En este caso no será necesario crear el usuario "ControllerUser" si no lo desea.

Una vez esté desplegado accederemos a http://localhost:8080/business-central/kie-wb.jsp e importamos nuestro proyecto kjar accediendo al menú de Diseño en BC y siguiendo los pasos del apartado [2.10.2].

Business Central debe traer por defecto una configuración de Servidor en business-central llamada `sample-server`. Se puede ver accediendo al menú de Implementar en BC.

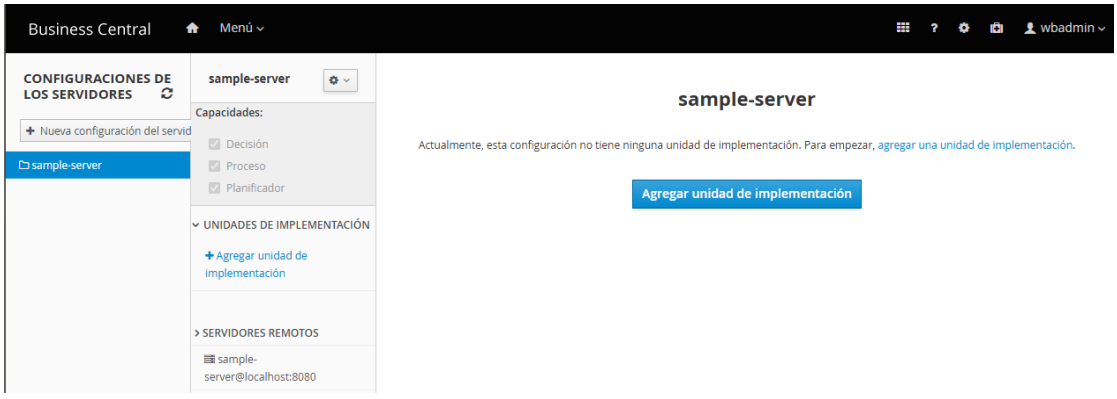

Ilustración 29. Verificación del sample-server

Puede comprobar que el servidor kie esté activo accediendo a la url http://localhost:8080/kie-server/services/rest/server :

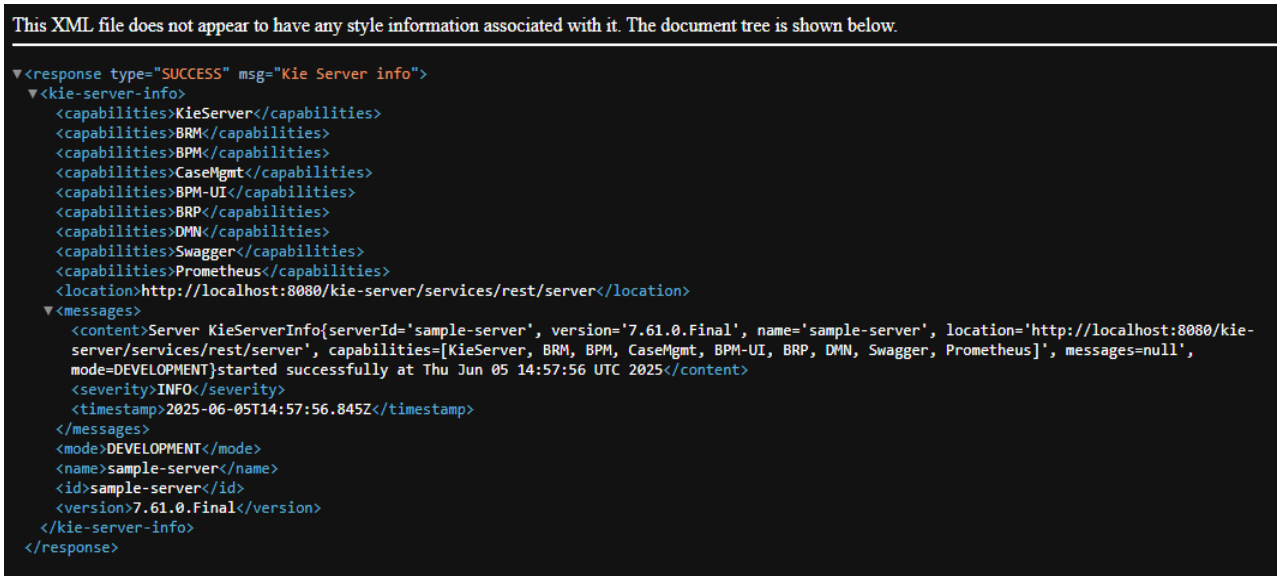

Ilustración 30. Comprobación del servidor Kie sample-server

Por último, solo falta importar el proyecto kjar que contiene los procesos jBPM de nuestra aplicación desde nuestro repositorio remoto git siguiendo los pasos del apartado [2.10.2], y pulsando el botón de implementar.

Deberá pulsar este botón cada vez que modifique el proyecto para que los cambios realizados en este se reflejen en el servidor remoto.

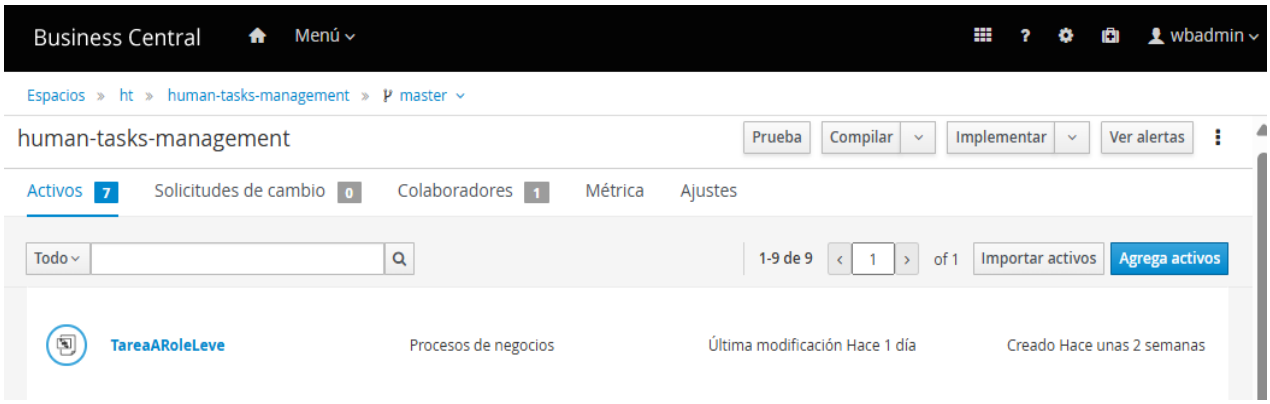

Ilustración 31. Implementación del proyecto remoto en el servidor Kie sample-server

Con esto ya estaría en funcionamiento el segundo servidor KIE. Si volvemos al menú de implementaciones podemos comprobar que se ha creado el contenedor resultado de la implementación de nuestro proyecto:

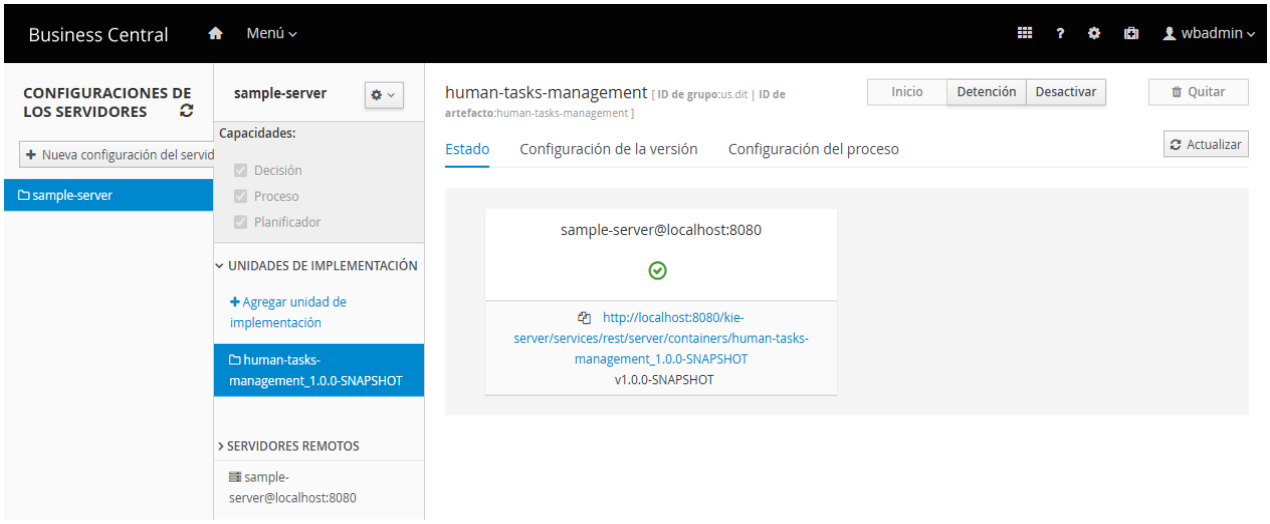

Ilustración 32. Creación del contenedor Kie en sample-server

Ahora solo falta configurar nuestra aplicación para que se conecte con los servidores que hemos levantado. Simplemente abra el fichero `application.properties` y añada a la lista de servidores su localización, el usuario de acceso y su contraseña. Es importante que el usuario del servidor KIE remoto alojado en Business Central con el que accedemos debe tener el rol `kie-server`.

```
#############################################################################
###                    IMPORTACIÓN DE KIE SERVERS                         ###
#############################################################################
#para que se pueda acceder desde BC alojado e docker al servidor kie de mi app
kieserver[0].location=http://localhost:8090/rest/server
kieserver[0].user=admin
kieserver[0].pwd=admin

kieserver[1].location=http://localhost:8080/kie-server/services/rest/server
kieserver[1].user=wbadmin
kieserver[1].pwd=wbadmin
#############################################################################
```

Ilustración 33. Configuración de los servidores Kie

El primer servidor siempre será de índice 0, y el orden en el que se configuran los servidores afecta a la creación de la lista de servidores KIE, debido a que se instancian en orden según su índice. Si el establecimiento de conexión con uno de ellos falla, ese servidor no se añadirá a la lista.

3.3.4 Servidor y recursos FHIR

El servidor FHIR de respaldo al que se conecta la aplicación se configura en el archivo `application.properties` del servicio. Por defecto se utiliza el servidor público https://hapi.fhir.org/baseR5/swagger-ui/ de Hapi Fhir. Aunque si se ha desplegado el entorno de desarrollo también puede conectarse al servidor local FHIR http://localhost:8888/fhir .

Este servidor aloja los recursos `Task` asociados a una tarea humana en el proceso. La versión actual de los procesos de test genera estos recursos automáticamente. No ocurre lo mismo con los recursos `Questionnaire`, por lo que deben estar previamente alojados en el servidor y configurados en la aplicación con su id asociado para que los tests funcionen correctamente. Los recursos `QuestionnaireResponse` se generarán también automáticamente al completar una tarea.

```
#fhir
fhir.server.base=https://hapi.fhir.org/baseR5
#fhir.server.base=http://localhost:8888/fhir

#variables para los tests

#identificador del cuestionario fhir pasado como entrada a las tareas humanas de test
test.questionnaireid=765063
```

Ilustración 34. Configuración de FHIR

En el caso de que se conecte al servidor público FHIR, no es necesario generar un recurso `Questionnaire`, se puede utilizar uno ya existente con el id "765063". Puede verificarlo accediendo a la url https://hapi.fhir.org/baseR5/Questionnaire/765063 .

Si está usando el servidor FHIR local desplegado por el entorno de desarrollo, necesitará crear el recurso manualmente. Para ello:

1. Acceda a http://localhost:8888/ .

2. Busque entre todos los recursos el de `Questionnaire`.

3. Seleccione la opción "CRUD Operations", y pegue el recurso `Questionnaire.json` de prueba que se facilita en nuestra aplicación, disponible en la carpeta `/resources`. Por último presiona el botón "Create":

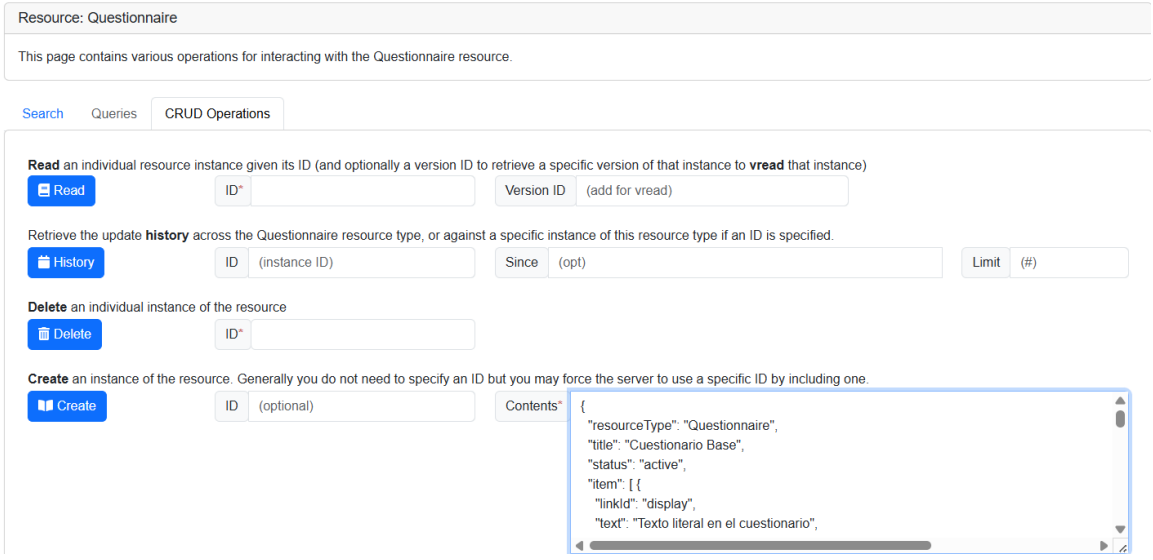

Ilustración 35. Creación de recurso Questionnaire

4. Verifique la respuesta con el cuestionario generado y copie su atributo "id" en la configuración de la aplicación. También puede ver los cuestionarios creados seleccionando la opción "Search", y luego presionando el botón "Search".

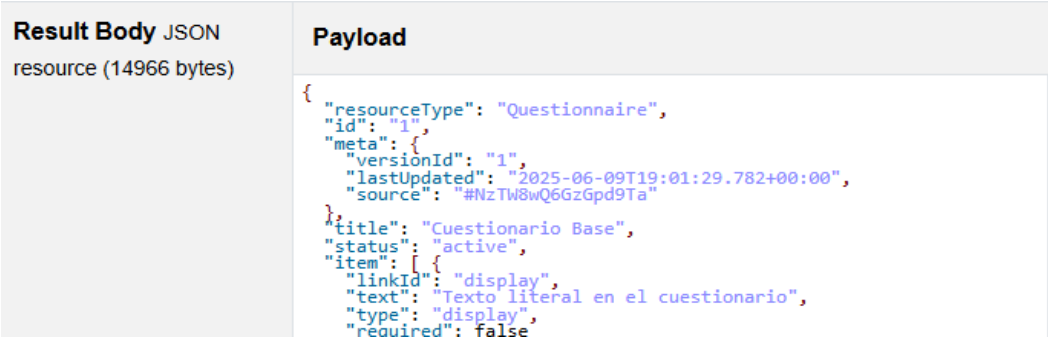

Ilustración 36. Verificación de recurso Questionnaire generado

3.3.5 Base de datos

La información de control e histórico de los procesos del servidor KIE embebido se almacena en una base de datos. La configuración de la misma se realiza también en el fichero `application.properties`. En la configuración proporcionada se utiliza una base de datos local PostgreSQL , de nombre `ht`.

Se presenta también la configuración necesaria para JPA, api utilizada para la persistencia, cuando se utiliza postgresql.

```
#data source configuration
spring.datasource.username=jbpm
spring.datasource.password=jbpm.2.DDBB*
spring.datasource.url=jdbc:postgresql://localhost:5432/ht
spring.datasource.driver-class-name=org.postgresql.xa.PGXADataSource

#hibernate configuration
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.properties.hibernate.show_sql=false
spring.jpa.properties.hibernate.hbm2ddl.auto=update
spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
```

Ilustración 37. Configuración de la base de datos

Es necesario que la base de datos esté creada antes de ejecutar la aplicación, y configurar adecuadamente este fichero para incluir el usuario y contraseña de la misma. Para ello ejecute en una consola PostgreSQL los comandos:

```
CREATE USER jbpm WITH PASSWORD 'jbpm.2.DDBB*';

CREATE DATABASE ht OWNER jbpm;
```

3.3.6 Ejecución

Una vez seguidos los pasos previos de configuración, ya estaremos en disposición de poner en marcha la aplicación. Para ejecutarla sitúese en un terminal en la carpeta `/human-tasks-service` del proyecto y ejecute el comando: `launch.bat clean install -Ppostgres` . Cuando cargue por completo, accederemos a la url http://localhost:8090/ , e introducimos los credenciales de registro, en este caso usuario “wbadmin” y contraseña “wbadmin”. Todos los usuarios configurados los podemos encontrar en el archivo `DefaultWebSecurityConfig.java`. Tras el inicio de sesión, se redireccionará al usuario al menú principal de la aplicación.

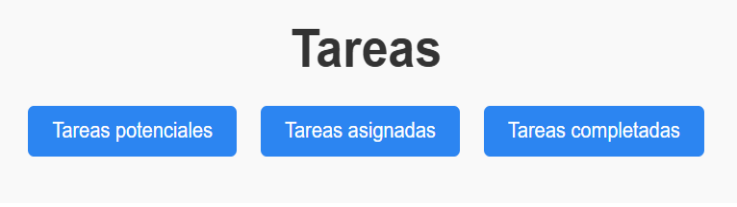

Ilustración 38. Pantalla menú principal

A continuación, se ha diseñado un escenario de prueba para mostrar con la mayor claridad posible las nuevas funcionalidades de la aplicación. En primer lugar haremos dos peticiones al controlador de test. La primera invocación será `/test/initTareaARol?serverIndex=0` , que creará una tarea jBPM en el primer servidor configurado (en nuestro caso el embebido por la aplicación). La segunda será `/test/initTareasARolMuestra?serverIndex=1`, que instancia tres tareas jBPM en el segundo servidor (el remoto ubicado en Business Central). A su vez también se debe haber creado una tarea FHIR por cada tarea jBPM iniciada. Puede comprobar si se ha establecido correctamente la conexión con los servidores y el índice asignado a cada uno en las trazas de la aplicación.

```
 main] u.d.h.s.services.kie.KieUtilFactory      : Inicializando KieUtilFactory...
xec-1] o.a.c.util.SessionIdGeneratorBase        : Creation of SecureRandom instance for session ID generation using [SHA1PRNG] took [1,023] milliseconds.
 main] u.d.h.s.services.kie.KieUtilFactory      : Creado cliente para servidor KIE http://localhost:8090/rest/server con indice 0
 main] u.d.h.s.services.kie.KieUtilFactory      : Creado cliente para servidor KIE http://localhost:8080/kie-server/services/rest/server con indice 1
 main] u.d.h.s.services.kie.KieUtilFactory      : Total de servidores KIE cargados: 2
```

Ilustración 39. Asignación de índices y comprobación de conexión con servidores KIE

Como resultado, si accedemos a la pantalla de tareas potenciales, veremos todas las tareas FHIR creadas. Se puede distinguir cuál ha sido creada en cada llamada a partir de la hora de creación.

Ilustración 40. Tareas potenciales-Resultado de invocación de controladores de test

Podemos observar que la tarea con prioridad urgente expira pasados cinco minutos desde su fecha de creación. Pasado ese tiempo, el timer de la tarea jBPM asociada debería ejecutarse y la tarea debería desaparecer automáticamente.

Ilustración 41. Tareas potenciales-Tarea expirada

Si observamos en la parte superior de la tabla, aparece un selector con el que podremos filtrar por los campos de "Nombre", "Fecha de creación", "Fecha de expiración", "Prioridad", y "Asunto". Mostrando solo las tareas que cumplan con las restricciones introducidas. En las siguientes ilustraciones se proporcionan varios ejemplos de filtrado, según distintos campos.

Ilustración 42. Tareas potenciales-Filtrado por fecha de expiración

Ilustración 43. Tareas potenciales-Filtrado por prioridad

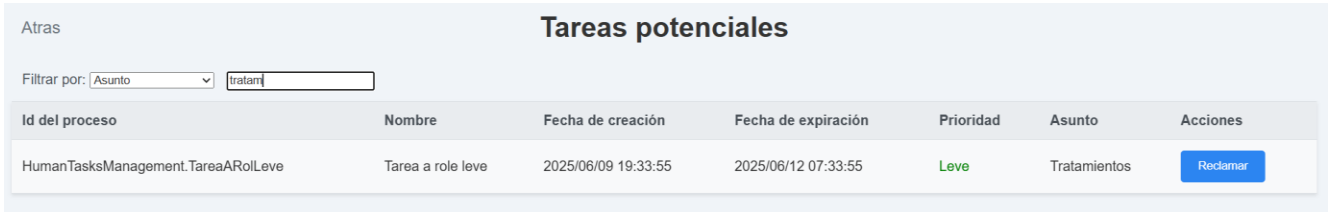

Ilustración 44. Tareas potenciales-Filtrado por asunto

Ahora, probaremos la funcionalidad de ordenación por campos. Para ello, pulsamos sobre la cabecera del campo por el que deseemos ordenar la tabla. A la derecha de dicho campo, aparecerá una flecha que nos indica el criterio seguido para ordenar la tabla, alternando entre ascendente o descendente cada vez que pulsemos sobre el mismo campo.

Ilustración 45. Tareas potenciales-Ordenación por fecha de creación ascendente

Ilustración 46. Tareas potenciales-Ordenación por fecha de creación descendente

Si pulsamos el botón “Reclamar”, se le asignará esa tarea al usuario que la reclamó. En consecuencia desaparecerá de la pantalla de tareas potenciales para todos los usuarios, y se mostrará en la pantalla en la pantalla de “Tareas asignadas” de ese usuario.

Ilustración 47. Pantalla tareas asignadas

Situados en la pantalla de tareas asignadas, presionaremos sobre "Comenzar" o "Continuar" si ya habíamos comenzado previamente la tarea para rellenar el formulario del cuestionario.

Cuando rellenemos todas las preguntas marcadas como (*), que son obligatorias de contestar (marcadas por el recurso `Questionnaire` como "Required"), debemos pulsar el botón "Enviar formulario" para completar la tarea. Entonces se generará un recurso `QuestionnaireRespone` cuyo id se añade al campo `output` de la tarea FHIR correspondiente. Por último nos redirecciona a la pantalla principal de la aplicación. Indicando con una alerta si se ha completado con éxito o ha habido algún error.

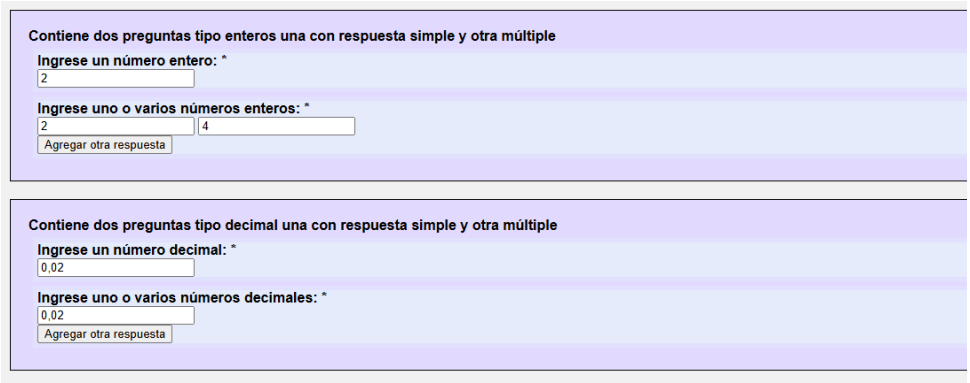

Ilustración 48. Pantalla del cuestionario

Si todo ha ido bien, el cuestionario que acabamos de completar habrá desaparecido de la pantalla tareas asignadas, y se mostrará ahora en la pantalla de tareas completadas.

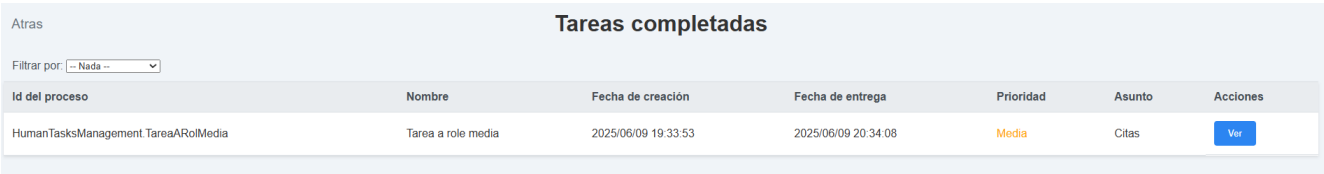

Ilustración 49. Pantalla tareas completadas

Por último, pulsamos en "Ver" para revisar el formulario del cuestionario con las respuestas que introducimos para esa tarea. Obtenidas a través del recurso `QuestionnaireResponse` generado en el paso anterior.

Ilustración 50. Pantalla del cuestionario de respuesta

3.3.7 Análisis del resultado

En este apartado se realizarán una serie de comprobaciones que se obtendrán como resultado de la ejecución de nuestro escenario de prueba. Llevando a cabo tras ello un razonamiento sobre las respuestas obtenidas, y si son las esperadas.

- Para comprobar que nuestra aplicación está conectada a varios servidores KIE, veremos las tareas que se han creado en cada uno de ellos, en función de la petición al controlador de test realizada. Recordemos que en el servidor KIE remoto invocamos el controlador de test `/test/initTareasARolMuestra?serverIndex=1`, cuya llamada instancia tres procesos. En consecuencia deberían haberse creado tres nuevas tareas en nuestro sample-server en Business-Central. Puede ver las tareas creadas navegando al menú principal > Gestionar tareas.

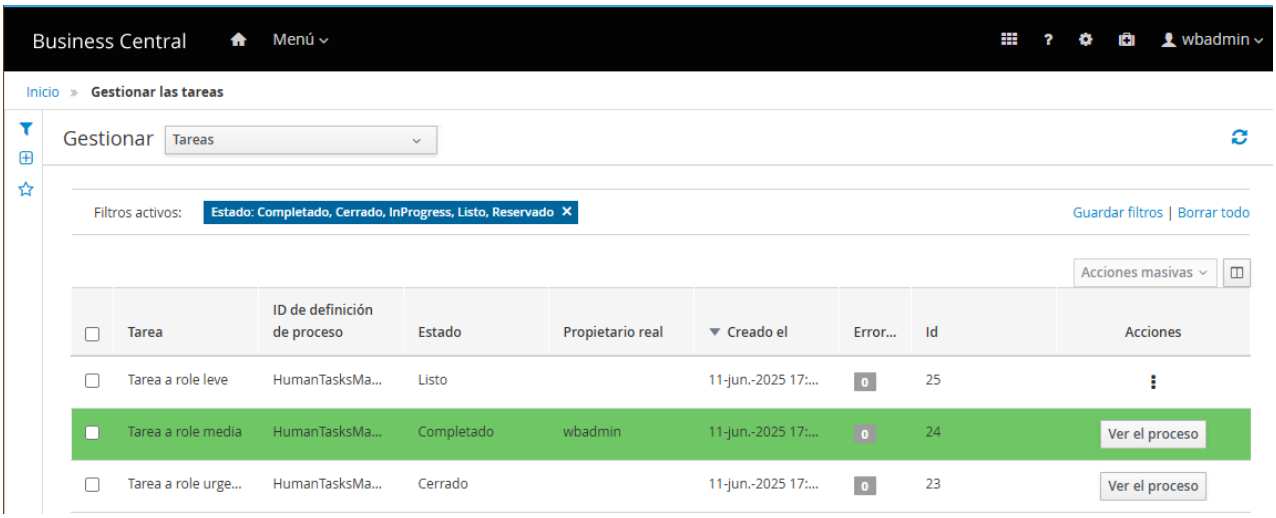

Ilustración 51. Tareas creadas en el servidor remoto de Business Central

En cambio en el servidor KIE embebido instanciamos una única tarea jBPM llamando a `/test/initTareaARol?serverIndex=0`, por lo que si nos conectamos a la base de datos `ht`, que gestiona la información de nuestro servidor Kie local embebido, y ejecutamos el comando mostrado en la pantalla inferior, deberíamos obtener una única tarea.

```
ht=# select * from task;
 id | archived | allowedtodelegate | description |  formname  |        name        | priority | subtaskstrategy | subject |     activationtime      |
 createdon       |           deploymentid           | documentaccesstype | documentcontentid |  documenttype   |  expirationtime  | faultaccessty
pe | faultcontentid | faultname | faulttype | outputaccesstype | outputcontentid | outputtype | parentid | previousstatus |            processid
    | processinstanceid | processsessionid | skipable | status | workitemid | tasktype | optlock | taskinitiator_id | actualowner_id | createdby_id
----+----------+-------------------+-------------+------------+--------------------+----------+-----------------+---------+-------------------------+-------
-----------------+----------------------------------+--------------------+-------------------+-----------------+------------------+-------
---+----------------+-----------+----------+------------------+-----------------+------------+----------+----------------+-------------------------
----+--------------------+------------------+---------+--------+------------+----------+---------+------------------+----------------+-------------
  1 |        0 |                   | Citas      | HumanTask1 | Tarea a role media |        5 | NoAction        | Citas   | 2025-06-09 19:29:12.457 | 2025-0
6-09 19:29:12.457 | human-tasks-management_1.0.0-SNAPSHOT |                  0 |                   1 | java.util.HashMap | 2025-06-12 07:29:11 |
    |              -1 |           |          |                  |                 |         -1 |          |       -1 |                   0 | HumanTasksManagement.TareaARolMe
dia |                1 |                1 | f        | Ready  |          1 |          |       1 |                    |                     |
(1 fila)
```

Ilustración 52. Tareas creadas en el servidor embebido en la aplicación

Para comprobar que la relación entre tareas FHIR y procesos jBPM es única, podemos consultar la asociación realizada entre ellas en las trazas de nuestra aplicación, y ver que efectivamente las asignaciones son únicas.

```
u.d.h.service.services.kie.TestService   : Entro en newTareaARol
u.d.h.service.services.kie.TestService   : Instanciado proceso 23 con la tarea FHIR asociada https://hapi.fhir.org/baseR5/Task/800112
ca.uhn.fhir.context.FhirContext          : Creating new FHIR context for FHIR version [R5]
u.d.h.service.services.kie.TestService   : Entro en newTareaARol
u.d.h.service.services.kie.TestService   : Instanciado proceso 24 con la tarea FHIR asociada https://hapi.fhir.org/baseR5/Task/800113
ca.uhn.fhir.context.FhirContext          : Creating new FHIR context for FHIR version [R5]
u.d.h.service.services.kie.TestService   : Entro en newTareaARol
u.d.h.service.services.kie.TestService   : Instanciado proceso 25 con la tarea FHIR asociada https://hapi.fhir.org/baseR5/Task/800114
```

Ilustración 53. Asociación entre proceso jBPM y tarea FHIR

Podemos acceder en un navegador a las url mostradas para ver si se han creado las tareas correctamente. Dado que se acaban de iniciar, estas tareas no deben tener campo output.

Response Body


```
{
  "resourceType": "Task",
  "id": "800113",
  "meta": {
    "versionId": "1",
    "lastUpdated": "2025-06-11T15:29:12.330+00:00",
    "source": "#gcRfUHyOpOTVBiO2"
  },
  "status": "ready",
  "intent": "order",
  "input": [ {
    "type": {
      "coding": [ {
        "system": "http://terminology.hl7.org/CodeSystem/task-input-type",
        "code": "closingQuestionnaire"
      } ],
      "text": "Id del cuestionario de cierre de la tarea"
    },
    "valueString": "765063"
  } ]
}
```


Ilustración 54. Tarea FHIR automáticamente generada

Y por último verificar los procesos jBPM, accediendo al menú principal de Business Central > Gestionar procesos, y comprobando que el valor de la variable "taskURI" sea el indicado por las trazas para cada id de proceso.

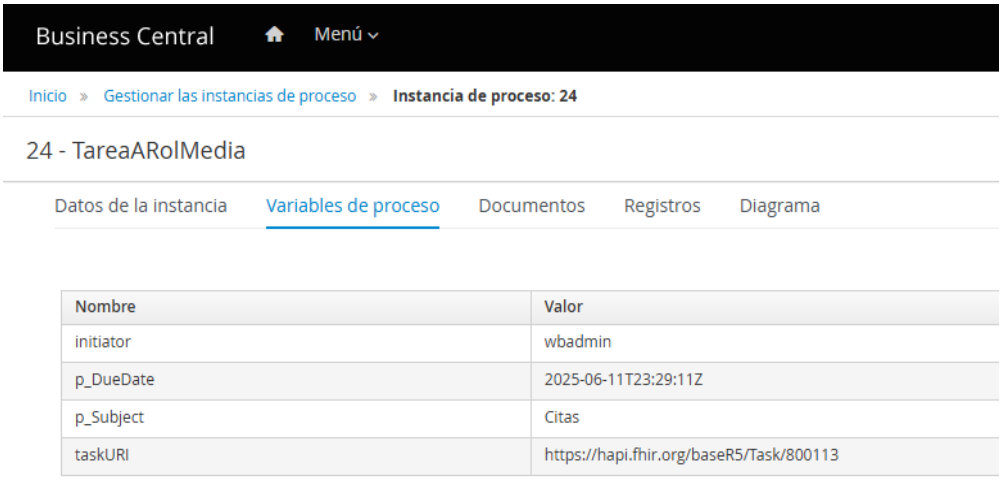

Ilustración 55. Proceso jBPM automáticamente generado

- Para comprobar si el proceso se ha finalizado correctamente, podemos acceder en el navegador a la URI de la tarea, y comprobar si se ha creado el campo output de la tarea añadiendo como valor la URI del recurso `QuestionnaireResponse` generado a partir de las respuestas del cuestionario. También podemos revisar que el recurso `QuestionnaireResponse` está persistido en el servidor FHIR y las respuestas se corresponden con las introducidas en el cuestionario, navegando a la URI proporcionada en el recurso Task.

Response Body


```
{
  "resourceType": "Task",
  "id": "800113",
  "meta": {
    "versionId": "4",
    "lastUpdated": "2025-06-11T16:12:03.387+00:00",
    "source": "#2mWof7RdLpFizvk1"
  },
  "status": "completed",
  "intent": "order",
  "input": [ {
    "type": {
      "coding": [ {
        "system": "http://terminology.hl7.org/CodeSystem/task-input-type",
        "code": "closingQuestionnaire"
      } ],
      "text": "Id del cuestionario de cierre de la tarea"
    },
    "valueString": "765063"
  } ],
  "output": [ {
    "type": {
      "coding": [ {
        "code": "closingQuestionnaireResponse"
      } ],
      "text": "Id de la respuesta al cuestionario de cierre de la tarea"
    },
    "valueString": "https://hapi.fhir.org/baseR5/QuestionnaireResponse/800117/_history/1"
  } ]
}
```


Ilustración 56: Tarea FHIR completada con campo output generado

4. CONCLUSIONES Y LÍNEAS FUTURAS

A modo de recapitulación, el objetivo principal de este proyecto ha sido continuar con el desarrollo y ampliar las funcionalidades de un servicio web de gestión de tareas sanitarias que se adhiere al paradigma de Business Process Management (BPM).

La solución se ha diseñado e implementado utilizando tecnologías clave como jBPM y Spring, aprovechando sus librerías. Para la gestión de plantillas en la aplicación, se ha empleado Thymeleaf, y la librería HapiFHIR ha sido fundamental para el manejo de tareas humanas dentro del ámbito sanitario.

La arquitectura de la solución, basada en el patrón Modelo-Vista-Controlador (MVC), ha permitido una mayor organización, legibilidad, mantenimiento, escalabilidad y robustez de la aplicación. Asimismo, Git ha sido una herramienta esencial para el control de versiones y la gestión del proyecto, especialmente con Business Central para la gestión de los proyectos kjar.

El esfuerzo dedicado en desplegar un entorno de desarrollo ha agilizado y facilitado significativamente el método de trabajo, permitiendo la modificación, la carga, y el monitoreo de procesos del motor KIE en tiempo de ejecución, e incorporando un modo depuración para un control preciso del flujo de ejecución del programa.

La gestión de tareas en múltiples motores de procesos ha supuesto una evolución considerable, que permite a la aplicación conectarse con varios servidores KIE declarados, recibiendo y gestionando tareas de diversas fuentes de origen desde una única ubicación.

Se ha optimizado la organización y visualización de tareas para los usuarios, añadiendo funciones de filtrado y ordenación. Además, se ha incorporado nueva información relevante referente a la tarea.

El proyecto también gestiona el ciclo de vida de vida de las tareas no completadas. Además añade la consulta de un historial de tareas completadas por el usuario, y la visualización de las respuestas a los cuestionarios asociados a estas. Para ello se ha establecido una vinculación única entre las tareas humanas del proceso jBPM y los recursos FHIR Task y QuestionnaireResponse.

Aunque los objetivos planteados han sido plenamente satisfechos, existen aún líneas futuras de desarrollo que podrían enriquecer aún más esta solución:

- Gestión de los ítems de tipo “attachment” en la plantilla del Questionnaire de FHIR, para poder guardar esta información al construir el QuestionnaireResponse.

- Añadir a la tarea FHIR el usuario que la realiza. Al igual que en la tarea jBPM está el campo actualOwner, en la tarea FHIR habría que hacer lo propio para saber qué usuario la tiene asignada, ya que ahora mismo únicamente se está cambiando el estado.

- El estado “Exited” de una tarea jBPM es final, por lo que una tarea expirada no podrá reasignarse nunca. Se debería modificar a “Suspend” para permitir reactivarlas. Y automatizar la actualización a dicho estado en la tarea FHIR.

- Debería de existir otro rol en la aplicación encargado de reactivar tareas expiradas, o incluso crearlas desde cero en un motor de procesos. Así como de crear cuestionarios y persistirlos en el servidor FHIR.

- Establecer un sistema de notificación a los usuarios para tareas urgentes o con expiración próxima.

- Gestionar la seguridad de la aplicación, ya que ahora mismo los usuarios y contraseñas están en un archivo de configuración dentro del proyecto. Para que a la hora de lanzar esta aplicación sea una aplicación robusta y segura.

- Añadir las acciones de cambio de estados de las tareas jBPM y FHIR en una misma transacción, para que en caso de fallo se haga un rollback de ambas y no se quede la aplicación en un estado inconsistente.

- Mejorar la automatización de la creación de la tarea FHIR, para que se pueda incluir directamente en los procesos, como un componente reutilizable.

En resumen, este proyecto conlleva un desarrollo significativo en una aplicación web en Spring para la gestión de tareas humanas en el ámbito sanitario, integrando tecnologías como jBPM, FHIR R5 y Thymeleaf. La solución permite a los usuarios gestionar tareas y cuestionarios FHIR de manera eficiente, proporcionando una conectividad, presentación y funcionalidad mejorada. Las bases están sentadas para posibles ampliaciones a futuro.

REFERENCIAS

[1] Alexander Obregon, Remote Debugging [En línea]. [Último acceso: 21 05 2025]. Available:https://medium.com/@AlexanderObregon/interactive-debugging-with-remote-spring-boot-applications-6d3f391f513b .

[2] Visual Studio Code VSCode debugging configuration [En línea]. [Último acceso: 22 05 2025]. Available:https://code.visualstudio.com/docs/debugtest/debugging .

[3] jBPM, human tasks [En línea]. [Último acceso: 22 05 2025]. Available:https://docs.jbpm.org/7.17.0.Final/jbpm-docs/html_single/#_jbpmtaskservice .

[4] jBPM, end events [En línea]. [Último acceso: 22 05 2025]. Available:https://docs.jbpm.org/7.17.0.Final/jbpm-docs/html_single/#_end_events .

[5] jBPM, timers [En línea]. [Último acceso: 22 05 2025]. Available: https://docs.jbpm.org/7.17.0.Final/jbpm-docs/html_single/#_timers .

[6] jBPM, data mappings [En línea]. [Último acceso: 22 05 2025]. Available: https://docs.jbpm.org/7.17.0.Final/jbpm-docs/html_single/#_datamappings .

[7] jBPM, KIE execution server [En línea]. [Último acceso: 10 06 2025]. Available: https://docs.jbpm.org/7.17.0.Final/jbpm-docs/html_single/#_ch.kie.server .

[8] JBoss Community, Due Date [En línea]. [Último acceso: 22 05 2025]. Available: https://developer.jboss.org/thread/253506 .

[9] The HL7 FHIR Foundation, FHIR DataTypes [En línea]. [Último acceso: 23 05 2025]. Available: https://build.fhir.org/datatypes.html#2.1.28.0 .

[10] The HL7 FHIR Foundation, Reference DataType FHIR [En línea]. [Último acceso: 23 05 2025]. Available: https://build.fhir.org/references.html#Reference

[11] The HL7 FHIR Foundation, «Task - FHIR v5.0.0,» [En línea]. [Último acceso: 26 05 2025]. Available: https://www.hl7.org/fhir/task.html.

[12] The HL7 FHIR Foundation, «Questionnaire - FHIR v5.0.0,» [En línea]. [Último acceso: 26 05 2025]. Available: https://www.hl7.org/fhir/questionnaire.html.

[13] The HL7 FHIR Foundation, «QuestionnaireResponse - FHIR v5.0.0,» [En línea]. [Último acceso: 26 05 2025]. Available: https://www.hl7.org/fhir/questionnaireResponse.html.

[14] HAPI FHIR, «QuestionnaireResponse - FHIR v5.0.0» [En línea]. [Último acceso: 26 05 2025]. Available:https://hapifhir.io/hapi-fhir/apidocs/hapi-fhir-structures-r5/org/hl7/fhir/r5/model/QuestionnaireResponse.html

[15] Javadoc, «TaskSummary» [En línea]. [Último acceso: 26 05 2025]. Available:https://javadoc.io/doc/org.kie.server/kie-server-api/latest/org/kie/server/api/model/instance/TaskSummary.html.

[16] Javadoc, «TaskInstance)» [En línea]. [Último acceso: 26 05 2025]. Available:https://javadoc.io/doc/org.kie.server/kie-server-api/latest/org/kie/server/api/model/instance/TaskInstance.html.

[17] Javadoc, «WorkItemInstance» [En línea]. [Último acceso: 26 05 2025]. Available:https://javadoc.io/doc/org.kie.server/kie-server-api/latest/org/kie/server/api/model/instance/WorkItemInstance.html.

[18] W3, JavaScript Sorting [En línea]. [Último acceso: 26 05 2025]. Available: https://www.w3.org/WAI/ARIA/apg/patterns/table/examples/sortable-table.

[19] Daext, JavaScript Filtering [En línea]. [Último acceso: 26 05 2025]. Available:https://daext.com/blog/how-to-create-an-html-table-with-sorting-and-filtering/.

[20] Javadoc, ProcessServicesClient. [En línea] [Último acceso: 04 07 2025]. Available: https://javadoc.io/doc/org.kie.server/kie-server-client/latest/org/kie/server/client/ProcessServicesClient.html

[21] Javadoc, UserTaskServicesClient.[En línea][Último acceso: 04 07 2025]. Available: https://javadoc.io/doc/org.kie.server/kie-server-client/latest/org/kie/server/client/UserTaskServicesClient.html

[22] Javadoc, KieServicesClient.[En línea][Último acceso: 04 07 2025]. Available: https://javadoc.io/doc/org.kie.server/kie-server-client/7.52.0.Final/org/kie/server/client/KieServicesClient.html

[23] Stackify, Streams.[En línea][Último acceso: 04 07 2025]. Available: https://stackify.com/streams-guide-java-8/

[24] Arquitectura Java, Modelo Vista Controlador.[En línea][Último acceso: 04 07 2025]. Available: https://www.arquitecturajava.com/el-modelo-vista-controlador-y-sus-responsabilidades/#:~:text=%C2%BFQu%C3%A9%20es%20el%20Patr%C3%B3n%20MVC,selecciona%20la%20vista%20a%20devolver.

[25] HAPI FHIR, Fhir Server: getting-started.[En línea][Último acceso: 04 07 2025]. Available: https://hapifhir.io/hapi-fhir/docs/server_plain/get_started.html

[26] jBPM, «Procesos jBPM. [En línea][Último acceso: 04 07 2025]. Available: https://docs.jbpm.org/7.17.0.Final/jbpm-docs/html_single/#jBPMBPMN2

[27] KIE Community, «KIE Community-About.[En línea][Último acceso: 04 07 2025]. Available: https://www.kie.org/about/

[28] GeeksforGeeks, Lenguaje de programación Java: Getting-Started>>. [En línea][Último acceso: 04 07 2025]. Available: https://www.geeksforgeeks.org/java/java/

[29] Oracle, «Java 8. [En línea][Último acceso: 04 07 2025]. Available: https://docs.oracle.com/javase/8/docs/

[30] The Thymeleaf Team, «Thymeleaf. [En línea][Último acceso: 04 07 2025]. Available: https://www.thymeleaf.org/documentation.html

[31] Wikipedia, «Paradigma BOM.[En línea] [Último acceso: 04 07 2025]. Available: https://en.wikipedia.org/wiki/Browser_Object_Model

[32] HAPI FHIR, Fhir Introduction.[En línea][Último acceso: 04 07 2025]. Available: https://hapifhir.io/hapi-fhir/docs/getting_started/introduction.html

[33] GBTEC, Qué is BPM? .[En línea][Último acceso: 04 07 2025]. Available: https://www.gbtec.com/es/wiki/gestion-de-procesos/bpm/

[34] Iberdrola, Desarrollo Low-Code. [En línea] [Último acceso: 04 07 2025]. Available: https://www.iberdrola.com/innovacion/low-code

[35] Karinavarela, Qué is KIE?.[En línea][Último acceso: 04 07 2025]. Available: https://karinavarela.me/2022/04/23/what-is-kie/

[36] Red Hat, Procesos de negocio.[En línea][Último acceso: 04 07 2025]. Available: https://www.redhat.com/en/topics/automation/what-is-business-process-management

[37] jBPM, Aplicaciones y activos de negocio.[En línea][Último acceso: 04 07 2025]. Available: https://www.jbpm.org/businessapps/gettingStarted.html

[38] IBM, Aplicaciones de negocio.[En línea][Último acceso: 04 07 2025]. Available: https://www.ibm.com/docs/en/taddm/7.3.0?topic=using-business-applications

[39] Ambit-Iberia, Qué es jBPM? [En línea][Último acceso: 04 07 2025]. Available: https://www.ambit-iberia.com/blog/qu%C3%A9-es-jbpm-tutorial

[40] IBM, BPMN. [En línea][Último acceso: 04 07 2025]. Available: https://www.ibm.com/docs/es/iis/11.5.0?topic=types-business-process-modeling-notation-bpmn-model

[41] Karinavarela, Kjar packaging.[En línea][Último acceso: 04 07 2025]. Available: https://karinavarela.me/2020/05/05/key-concepts-of-jbpm/

[42] RedHat, Conectar Business Central con KIE Server.[En línea][Último acceso: 04 07 2025]. Available: https://docs.redhat.com/es/documentation/red_hat_decision_manager/7.8/html/managing_and_monitoring_kie_server/kie-server-configure-central-proc_execution-server

[43] IBM, Que es SpringBoot? .[En línea][Último acceso: 04 07 2025]. Available: https://www.ibm.com/mx-es/topics/java-spring-boot

[44] Spring, SpringBoot Initializr.[En línea][Último acceso: 04 07 2025]. Available: https://docs.spring.io/initializr/docs/current/reference/html/

[45] Arquitectura Java, Qué es Maven?. [En línea][Último acceso: 04 07 2025]. Available: https://www.arquitecturajava.com/que-es-maven/

[46] Microsoft Learn, Control de versiones Git. [En línea][Último acceso: 04 07 2025]. Available: https://learn.microsoft.com/es-es/devops/develop/git/what-is-git

[47] IBM, Qué es PostgreSQL?. [En línea][Último acceso: 04 07 2025]. Available: https://www.ibm.com/mx-es/think/topics/postgresql

[48] IBM, Qué es Docker?. [En línea][Último acceso: 04 07 2025]. Available: https://www.ibm.com/es-es/think/topics/docker

[49] Rieckpil, Eclipse remote debugging configuration. [En línea] Available: https://rieckpil.de/howto-remote-debug-spring-boot-applications-intellij-idea-eclipse/

[50] Github, Asignación de variables entre recursos jBPM.[En línea][Último acceso: 04 07 2025]. Available: https://github.com/kiegroup/jbpm/blob/main/jbpm-human-task/jbpm-human-task-workitems/src/main/java/org/jbpm/services/task/wih/AbstractHTWorkItemHandler.java#L156

[51] jBPM, Business Central. [En línea][Último acceso: 04 07 2025]. Available: https://docs.jbpm.org/7.17.0.Final/jbpm-docs/html_single/#_business_central

[52] jBPM, Definiciones de proceso. [En línea][Último acceso: 04 07 2025]. Available:https://docs.jbpm.org/7.17.0.Final/jbpm-docs/html_single/#_process_definitions_management

[53] jBPM, Instancias de proceso. [En línea][Último acceso: 04 07 2025]. Available:https://docs.jbpm.org/7.17.0.Final/jbpm-docs/html_single/#_process_instances_management

[54] jBPM, Variables de proceso. [En línea][Último acceso: 04 07 2025]. Available: https://docs.jbpm.org/7.17.0.Final/jbpm-docs/html_single/#_variables