UNIVERSIDAD POLITÉCNICA DE CARTAGENA

ESCUELA TÉCNICA SUPERIOR DE
INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FINAL DE GRADO.

Ingeniería de Telemática

INVENTARIO FARMACÉUTICO DEL HOGAR.

JOSÉ ANTONIO SALMÉRON MARÍN

Cartagena, 11 de marzo 2024

DIRECTOR: JOSÉ FERNANDO CERDÁN CARTAGENA

Dpto. Tecnologías de la Información y las Comunicaciones

AGRADECIMIENTOS.

Agradezco al profesor y tutor del proyecto Fernando, toda su ayuda y predisposición mostrada en esta nueva etapa universitaria. Gracias a su dedicación, ha hecho que este paso por la universidad después de tantos años, quede guardado como un grato recuerdo en mi memoria.

También agradecer a mi amigo Juan sus consejos y ayuda cuando más lo he necesitado y a mi mujer Paula por su inagotable paciencia.

RESUMEN.

Este trabajo trata sobre una aplicación web que permite controlar el inventario farmacéutico casero, la cual facilita el control de los medicamentos para impedir el exceso de los mismos, evita gastos innecesarios, así como mal uso de los mismos por caducidad.

Para la realización del mismo se han usado diferentes herramientas, las cuales se documentan en el primer apartado de este trabajo, donde se explica su funcionalidad y origen de las mismas.

En el segundo, se profundiza en el código implementado para el funcionamiento del programa, desglosándolo en las utilidades que dispone para entender mejor su manejo.

Por último, se visualiza la interfaz del programa con imágenes para que el usuario observe la operatividad del mismo.

Tabla de contenido

Introducción.

Contexto y justificación del proyecto.

En la sociedad actual, el acceso y la gestión eficiente de los medicamentos en el ámbito doméstico se han convertido en aspectos importantes para la salud y el bienestar de las personas. Con la creciente conciencia sobre la importancia de mantener un inventario farmacéutico en el hogar, surge la necesidad de crear herramientas que simplifiquen el control de los medicamentos disponibles.

El aumento de la automedicación y la diversidad de tratamientos disponibles hacen que la gestión de un inventario farmacéutico casero sea una tarea cada vez más compleja. Este proyecto ha abordado esta problemática mediante el desarrollo de una aplicación basada en tecnologías modernas que permite a los usuarios gestionar de manera eficaz y segura su inventario de medicamentos en el entorno doméstico.

La relevancia de esta iniciativa radica en proporcionar a los usuarios una herramienta intuitiva y fácil de usar que simplifique la gestión de sus medicamentos.

Para la implementación de una solución tecnológica sencilla, se ha usado Spring Boot, Thymeleaf, Bootstrap y MySQL, cuya combinación proporciona una plataforma sólida y escalable para futuras mejoras.

Este proyecto busca resolver las siguientes problemáticas identificadas en el contexto actual:

- Complejidad en la gestión: La diversidad de medicamentos y tratamientos dificulta la tarea de llevar un control efectivo del inventario farmacéutico en el hogar, además de generar un desperdicio teniendo algunos medicamentos caducados o duplicados al desconocer la existencia de los mismos en casa.
- Necesidad de herramientas modernas: La integración de tecnologías actuales permite una gestión más eficiente y accesible del inventario, adaptándose a las necesidades de cada usuario dependiendo del lugar donde se encuentre.

En este contexto, la implementación de este proyecto ha respondido a la demanda actual de soluciones tecnológicas para la gestión de inventarios farmacéuticos caseros, y además marca el inicio de una tecnología inicialmente sencilla, que puede explotarse y mejorarse en un futuro en beneficio de una salud y bienestar equilibrado.

Objetivos.

En este proyecto se ha llevado a cabo el desarrollo de una aplicación basado en el modelo MVC con la ayuda de Spring Framework, una infraestructura de código abierto desarrollado en Java. En primer lugar, se ha llevado a cabo un estudio teórico de las herramientas que se han utilizado para la realización del proyecto, finalizando con la implementación de dicha aplicación como ejemplo práctico.

La primera parte consta de un análisis sobre las características de Spring, mostrando toda la información necesaria para poder desarrollar una aplicación basada en la estructura de MVC.

El objetivo es conocer esta herramienta para poder desarrollar un proyecto con las facilidades que proporciona.

La segunda parte, es el propio desarrollo del proyecto, una demostración práctica partiendo de todo el contenido de la primera. La aplicación desarrollada trata sobre un inventario casero farmacéutico, que consta por un lado de un registro de medicamentos, los cuales pueden ampliarse, modificarse o eliminarse y donde se reflejan las características genéricas de los mismos (nombre, forma de empleo, formato…) y por otro, de listados con los medicamentos que se disponen en un lugar concreto, por ejemplo, aquellas medicinas que se disponen en casa, teniendo en cuenta la cantidad de las mismas que se tiene o de su fecha de caducidad.

Asimismo, se tiene en cuenta aquellos medicamentos que estén próximos a agotarse o caducarse, mostrando una ventana modal al inicio de la aplicación en el caso que haya algún medicamento que cumpla esas condiciones, pudiendo redireccionarse al listado concreto, donde muestra en rojo si la cantidad está por debajo de un valor específico o la fecha de caducidad está próxima o ya caducada.

Se han añadido facilidades para encontrar medicamentos mediante un buscador, teniendo un acceso inmediato al mismo, sabiendo en que inventario se encuentra, su cantidad y caducidad.

Con el desarrollo de esta aplicación se ha intentado minimizar el despilfarro de medicamentos y el gasto innecesario, debido a que en muchas ocasiones se pueden llegar a acumular diferentes cajas del mismo medicamento por desconocimiento de lo que se tiene en el propio hogar.

Alcance del trabajo

El alcance de este proyecto se delimita a la planificación, desarrollo e implementación de una aplicación web de gestión de inventario farmacéutico casero, siendo los aspectos específicos los siguientes:

- Interfaz de Usuario (UI)
  Diseño y desarrollo de una interfaz de usuario intuitiva y fácil de usar, permitiendo a los usuarios gestionar su inventario de medicamentos de manera eficiente.

- Funcionalidades Principales
  Implementación de funciones clave, como agregar, eliminar y actualizar medicamentos en el inventario, así como alertas respecto a la cantidad o caducidad.

- Integración de diferentes tecnologías
  Utilización de tecnologías específicas como Spring Boot, Thymeleaf, Bootstrap y MySQL para el desarrollo tanto del frontend como del backend.

- Conexión con Base de Datos
  Configuración de la conexión con una base de datos MySQL para almacenar y recuperar la información del inventario.

Estudio teórico.

Inventario farmacéutico en el hogar y su importancia

El inventario farmacéutico se refiere al conjunto de medicamentos que se encuentran comúnmente en los hogares para uso personal, bien sean recetados por el médico al que se acude o bien que se venden de forma libre sin necesidad de receta. La presencia de una farmacia doméstica bien organizada es necesaria para ahorrar tiempo y recursos al evitar la necesidad acudir innecesariamente a la farmacia. Además, contribuye a una gestión más eficiente de los medicamentos y productos de salud, organizando eficientemente los recursos farmacéuticos que se dispone en el hogar, reduciendo la complejidad a la hora de controlar los diferentes medicamentos que se necesitan en una unidad familiar, evitando el consumo de productos caducados que pierden eficacia o puedan afectar a la salud y permitiendo prever la necesidad de compra de un medicamento antes de que se agote o evitar la duplicidad en la compra de medicamentos que ya se disponen.

Respecto a la importancia del inventario farmacéutico doméstico se puede comentar los siguientes aspectos:

- Acceso Rápido a Medicamentos Básicos:

Proporciona un acceso inmediato a medicamentos comunes para el tratamiento de dolencias menores, como analgésicos, antipiréticos y medicamentos para el resfriado, lo que permite una respuesta rápida ante enfermedades leves.

- Autocuidado y Automedicación Responsable:

Facilita el autocuidado y la automedicación responsable al ofrecer una variedad de productos de venta libre. Sin embargo, es esencial gestionar estos medicamentos de manera organizada y consciente para evitar errores en la administración.

- Gestión de Enfermedades Crónicas:

Para aquellos con condiciones crónicas, la farmacia doméstica puede contener medicamentos recetados y suministros necesarios para el manejo diario de la enfermedad. La organización eficiente de estos medicamentos es clave para garantizar el correcto seguimiento al tratamiento.

- Respuesta a Emergencias:

En situaciones de emergencia, como cortes, quemaduras o fiebre súbita, la disponibilidad inmediata de productos en la farmacia doméstica puede marcar la diferencia. La rápida respuesta es esencial para mitigar los efectos de emergencias de salud.

- Complejidad del Inventario:

La diversidad de productos y medicamentos puede hacer que la gestión del inventario sea compleja, especialmente cuando se trata de múltiples miembros en un hogar.

- Caducidad y Renovación:

Es fundamental realizar un seguimiento de las fechas de caducidad y renovar los productos vencidos. La falta de una gestión efectiva puede dar lugar al uso de medicamentos caducados, comprometiendo su eficacia y teniendo efectos perjudiciales para la salud.

Por lo tanto, el desarrollo de una aplicación de gestión de inventario farmacéutico doméstico se presenta como una solución que aborda los desafíos mencionados, mejora la organización, accesibilidad y seguridad de la farmacia del hogar para el beneficio de la salud y el bienestar familiar.

Tecnologías Utilizadas

Spring y Spring Boot

Spring es un framework para el desarrollo de aplicaciones y contenedor de inversión de control, de código abierto para la plataforma Java. Fue lanzado inicialmente bajo la licencia Apache 2.0 en junio de 2003 [2].

Entre sus características principales destacar[3]:

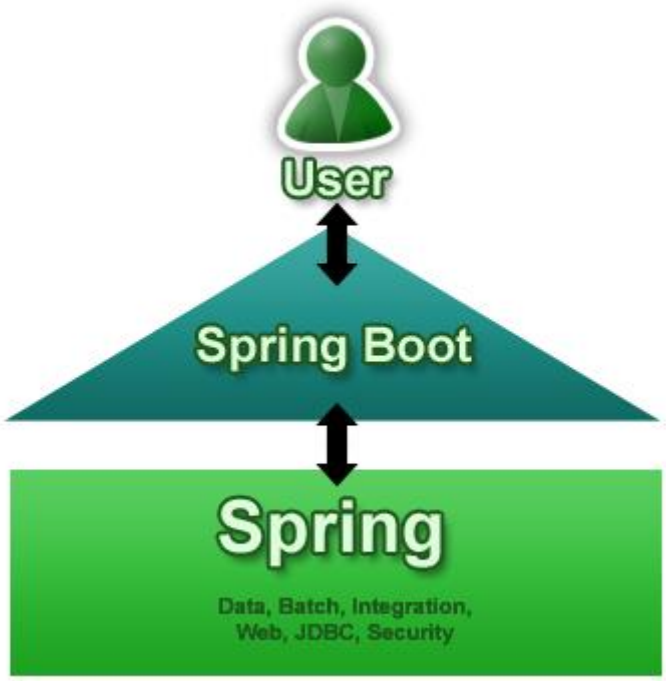
- Inyección de Dependencias (DI): Permite la inversión de control (IoC) y facilita la creación de aplicaciones flexibles y mantenibles.
- Aspect-Oriented Programming (AOP): Permite separar preocupaciones transversales como la seguridad y el registro del código de negocio.
- Módulos Especializados: Ofrece módulos para tareas específicas como Spring MVC para el desarrollo web y Spring Data para el acceso a datos.

Según la propia web de Spring las funciones a realizar pueden ser[1]:

- Microservicios: Las arquitecturas de microservicios son la "nueva normalidad". La creación de aplicaciones pequeñas, autónomas y listas para ejecutarse puede aportar una gran flexibilidad y mayor resiliencia a su código. Las características especialmente diseñadas de Spring Boot facilitan la creación y ejecución de sus microservicios en producción a escala. Además, con Spring Cloud, que facilita la administración y aumenta la tolerancia a fallos, la arquitectura de microservicios quedará mejor definida.
- Reactivo: Los sistemas reactivos tienen ciertas características que los hacen ideales para cargas de trabajo de baja latencia y alto rendimiento. Project Reactor y la cartera de Spring trabajan juntos para permitir a los desarrolladores crear sistemas reactivos de nivel empresarial que sean receptivos, resistentes, elásticos y basados en mensajes.

- Cloud: El desarrollo de sistemas distribuidos puede ser un desafío. La complejidad se traslada de la capa de aplicación a la capa de red y exige una mayor interacción entre los servicios. Hacer que el código sea "nativo de la nube" significa tratar con problemas de 12 factores, como la configuración externa, la statelessness, el registro y la conexión a servicios de respaldo. El conjunto de proyectos de Spring Cloud contiene muchos de los servicios necesarios para que las aplicaciones se ejecuten en la nube.
- Web apps: Spring hace que la creación de aplicaciones web sea rápida y sin complicaciones. Al eliminar gran parte del código repetitivo y la configuración asociados con el desarrollo web, obtiene un modelo de programación web moderno que agiliza el desarrollo de aplicaciones HTML del lado del servidor, API REST y sistemas bidireccionales basados en eventos.
- Serverless: Las aplicaciones sin servidor aprovechan las capacidades modernas de computación en la nube y las abstracciones para permitirle concentrarse en la lógica en lugar de en la infraestructura. En un entorno sin servidor, puede concentrarse en escribir el código de la aplicación mientras la plataforma subyacente se ocupa del escalado, los tiempos de ejecución, la asignación de recursos, la seguridad y otras especificaciones del "servidor".
- Event driven: Los sistemas basados en event-driven reflejan cómo funcionan realmente las empresas modernas: miles de pequeños cambios que ocurren todo el día, todos los días. La capacidad de Spring para manejar eventos y permitir a los desarrolladores crear aplicaciones a su alrededor significa que sus aplicaciones permanecerán sincronizadas con su negocio. Spring tiene una serie de opciones para elegir basadas en eventos, desde integración y transmisión hasta funciones en la nube y flujos de datos.
- Batch: La capacidad del procesamiento por lotes (batch) para procesar de manera eficiente grandes cantidades de datos lo hace ideal para muchos casos. La implementación de Spring Batch de patrones de procesamiento estándar le permite crear trabajos por lotes sólidos en la JVM. Agregar Spring Boot y otros componentes de la cartera de Spring permite crear aplicaciones por lotes de misión crítica.
- La finalidad, es crear proyectos de Spring, pero de una manera más ágil, mediante a una serie de convenciones que prevalecen sobre la configuración. Ahorrando el tener que perder tiempo en realizar configuraciones pesadas en ficheros, por ello aparece Spring Boot.

Spring Boot transforma la forma en la que se aborda las tareas de programación de Java, agilizando la experiencia. Combina herramientas tales como un contexto de aplicación y un servidor web incorporado autoconfigurado para hacer que el desarrollo de microservicios sea sencillo. Para ir aún más rápido, se puede combinar Spring Boot con el amplio conjunto de bibliotecas, servidores, patrones y plantillas compatibles de Spring Cloud para implementar de forma segura arquitecturas completas basadas en microservicios en la nube, en un tiempo récord.



(Imagen de: https://softwareevolutivo.com.ec/espanol-desarrollo-de-aplicaciones-web-con-spring-boot/)

Características Clave de Spring Boot:

- Configuración Automática: Reduce la necesidad de configuración manual al proporcionar configuraciones predeterminadas basadas en el entorno y las dependencias detectadas.
- Incrustado y Listo para Producir: Ofrece servidores integrados, como Tomcat o Jetty, lo que facilita la creación de aplicaciones autocontenidas y listas para producción.
- Gestión de Dependencias: Utiliza Spring Initializr para gestionar dependencias y configuraciones de proyectos de manera sencilla.
- Desarrollo Basado en Convenciones: Fomenta el desarrollo basado en convenciones, pero permite la personalización cuando sea necesario.
- Monitorización Activa: Proporciona características integradas para la monitorización y gestión de aplicaciones.

Thymeleaf

Thymeleaf es un motor de plantillas para Java que se utiliza en el desarrollo web para crear vistas de manera dinámica. Su principal objetivo es integrar las plantillas de manera más natural en el desarrollo de aplicaciones web basadas en Java[5].

Thymeleaf utiliza una sintaxis fácil de entender, lo que facilita la creación y mantenimiento de plantillas. Además, puede trabajar con documentos HTML5 y XML, haciéndolo versátil para su uso en diferentes contextos de desarrollo web[5].

Respecto a este proyecto, se integra de manera natural y sencilla con Spring Framework, siendo la opción preferida para el desarrollo web con Spring. Como ejemplo de integración, los *"@Controller"* de Spring pueden redireccionarse a las plantillas Thymeleaf, estando la propia web de spring.io construida con Thymeleaf.

Hay que tener en cuenta que Thymeleaf realiza el procesamiento del lado del servidor, generando HTML dinámico que se envía al navegador del usuario.

Por último, permite el uso de expresiones en las plantillas, facilitando la inclusión de datos dinámicos y la evaluación de condiciones en la vista, siendo dichas expresiones las siguientes[4]:

- Expresiones variables: Suelen ser las más usadas, como por ejemplo ${…}
  ```
  <td th:text="${expiredOrQuantity[0]}"></td>
  ```
- Expresiones de selección: Son expresiones para reducir la longitud de la expresión si se prefija un objeto mediante una expresión variable, como por ejemplo *{…}
- Expresiones de mensaje: permite, a partir de ficheros properties o ficheros de texto, cargar los mensajes e incluso realizar la internalización de las aplicaciones, como por ejemplo #{…}
  ```
  <td th:text="${#dates.format(inventarios.date_edit, 'dd/MM/yyyy')}"></td>
  ```
- Expresiones de enlace: crean o direccionan a una URL, como por ejemplo @{…}
  ```
  <a th:href="@{/home}"
  ```
- Expresiones de fragmentos: Usados para dividir las plantillas en plantillas más pequeñas e ir cargándolas según se vayan necesitando, como por ejemplo ~{…}

Thymeleaf se vale de una serie de atributos básicos para que su manejo sea más sencillo, entre los más usados tenemos los siguientes[4]:

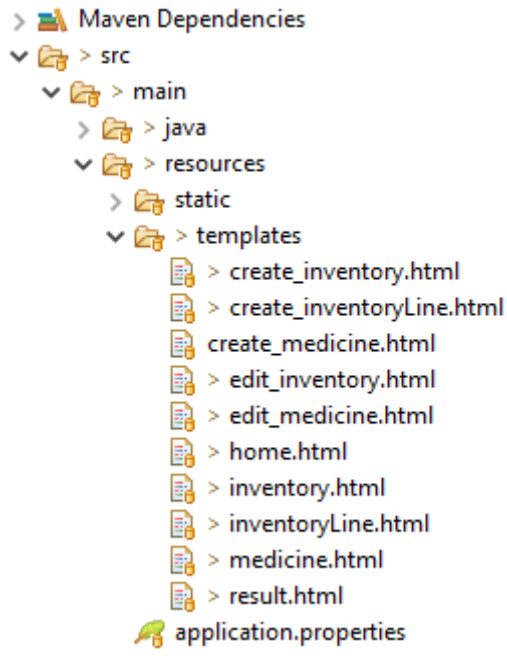
- th:text: Permite reemplazar el texto de la etiqueta por el valor de la expresión que se le de.
  ```
  <td th:text="${lineaInv.medicine.name}"></td>
  ```
- th:each: Va a permitir repetir tantas veces como se indique o iterar sobre los elementos de una colección.
  ```
  <tr th:each="lineaInv, rowStat : ${listinv}"
  ```

Uso básico de Thymeleaf:

1. Configuración en Proyecto Spring Boot agregando la dependencia de Thymeleaf en el proyecto Spring Boot. Para ello la siguiente dependencia debe encontrarse en el archivo pom.xml. Generalmente se suele hacer a través de Maven o Gradle una vez que se genera el proyecto.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

2. Estructura de Proyecto: Thymeleaf sigue una estructura de carpetas convencional. Las plantillas se colocan en el directorio ***src/main/resources/templates/***.



3. Integración con Páginas HTML: utilizando atributos específicos en las etiquetas HTML para indicar dónde se deben insertar los datos dinámicos mediante el uso de expresiones, variables (vistas anteriormente), iteraciones y condiciones, teniendo en cuenta que se distinguen con el uso del prefijo "*th:*". como por ejemplo este código, donde cada iteración (each) de la lista muestra el valor en una celda de la tabla.

```
<tr th:each="expiredOrQuantity : ${expiredMedicinesorLowQuantity}">
    <td th:text="${expiredOrQuantity[0]}"></td>
    <td th:text="${expiredOrQuantity[2]}"></td>
```

Por todo lo anteriormente expuesto, Thymeleaf se ha convertido en una elección popular para el desarrollo web en Spring debido a su integración fluida y su sintaxis clara. Su capacidad para generar contenido dinámico y su uso sencillo lo hacen ideal para el desarrollo de vistas en aplicaciones web basadas en Java.

Boostrap

Bootstrap es un framework front-end de código abierto basado en HTML, CSS y JavaScript, utilizado para desarrollar aplicaciones web y sitios mobile first. Fue desarrollado por Twitter en 2011 y está diseñado para simplificar el desarrollo web haciendo que la creación de páginas web sea más rápida y fácil, permitiendo a los desarrolladores centrarse en la creatividad y la innovación. Inicialmente se llamó Twitter Blueprint adoptando su nombre actual en 2011, donde se dejó en código abierto[7].

Utilizando HTML, CSS y JavaScript, Bootstrap ofrece una serie de estilos predefinidos, componentes y utilidades que permiten a los desarrolladores construir interfaces atractivas y receptivas de manera eficiente.

Uno de los aspectos más destacados de Bootstrap es su enfoque en la responsividad[7], es decir, los sitios web creados con Bootstrap se adaptan automáticamente a diferentes tamaños de pantalla, desde dispositivos móviles hasta pantallas de escritorio. Esto no solo mejora la

experiencia del usuario, sino que también es esencial para el posicionamiento en motores de búsqueda, ya que Google favorece los sitios web responsivos.

Bootstrap se basa en un sistema de cuadrícula (grid system) que facilita la creación de diseños flexibles y adaptables. Los desarrolladores pueden dividir la pantalla en columnas y definir cómo se comportarán en diferentes tamaños de pantalla proporcionando un control preciso sobre la disposición de los elementos en la página. A continuación, se muestra un ejemplo del grid system mostrando campos de un formulario a rellenar:

```
<div class="mb-3">
    <label for="composition" class="form-label">Composición</label>
    <input type="text" class="form-control" id="composition" name="composition"
    placeholder="Indique los compuestos que lo forman">
</div>
<div class="mb-3">
    <label for="treatment" class="form-label">Tratamiento*</label>
    <input type="text" class="form-control" id="treatment" name="treatment" required
    placeholder="Modo de administrar según el médico">
</div>
```

Dispone de una amplia gama de componentes predefinidos, como barras de navegación, botones, formularios, modales, etc., que permiten ahorrar tiempo y esfuerzo, ya que los desarrolladores no tienen que empezar desde cero. Aunque dispone de dichos componentes prediseñados, también es personalizable. Los desarrolladores pueden modificar fácilmente los estilos y comportamientos según las necesidades específicas del proyecto. Como ejemplo de estos componentes se muestra el código de uno de los más sencillos y útiles, un botón:

```
<input class="btn btn-primary" type="submit" value="Buscar">
```

Todo lo anteriormente explicado está plasmado en una documentación extensa donde hay explicaciones detalladas, ejemplos de código y directrices que facilitan el aprendizaje y la implementación.

La inclusión de Bootsrap en los proyectos es fácil, simplemente se necesita incluir los archivos CSS y JS de Bootstrap en las plantillas del proyecto antes de la etiqueta </body>. Esto se puede hacer descargando los archivos desde el sitio oficial de Bootstrap o utilizando una CDN (Content Delivery Network) para acceder a las versiones más recientes[6].

```
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
```

MySQL

Para el almacenamiento de los datos se utiliza una de las bases de datos más populares y fáciles de usar, MySQL. Desarrollado originalmente por la empresa sueca MySQL AB, fue adquirida por Sun MicroSystems en 2008 y esta su vez comprada por Oracle Corporation en 2010. Es un sistema de gestión de bases de datos con una doble licencia, es decir, una parte es de código abierto (principal motivo por el que muchos desarrolladores trabajan con ella), y otra, cuenta con una versión comercial gestionada por Oracle[8].

MySQL es un sistema de gestión de bases de datos relacional (RDBMS, por sus siglas en inglés) que se basa en el modelo de datos relacional, es decir, utiliza tablas múltiples que se interconectan entre sí para almacenar la información y organizarla correctamente[8]. Utiliza el lenguaje SQL (Structured Query Language) para acceder y manipular los datos[10].

Se debe diferenciar entre SQL y MySQL, siendo el primero el lenguaje universal de consulta de bases de datos, mientras que el segundo es un sistema de gestión de bases de datos específico que utiliza SQL para interactuar con la base de datos[10].

Las principales características que lo definen son las siguientes:

- Compatibilidad: existe un amplio abanico de API nativas, librerías, paquetes, etc. que permiten integrar una base de datos MySQL en un sistema desarrollado en cualquier lenguaje de programación y herramientas de desarrollo[9].
- Multiplataforma: Compatible con diferentes sistemas operativos, pudiendo ejecutarse en Windows, Linux y macOS[10].
- Escalabilidad: se puede escalar para adaptarse a las necesidades de diferentes aplicaciones. MySQL puede manejar desde pequeñas aplicaciones hasta grandes sistemas empresariales[10].
- Rendimiento: Ofrece una recuperación de datos rápida y eficiente, ideal para aplicaciones de alto rendimiento[10].
- Almacenamiento de diferentes tipos de datos: Soporta una amplia gama de tipos de datos, desde datos numéricos hasta cadenas de texto y archivos multimedia, lo que permite tener una gran versatilidad[9].
- Seguridad: utiliza un sistema de privilegios de acceso y contraseñas encriptadas que permite la verificación basada en el host[10]. Para acceder a los datos, se deberá utilizar un nombre de cuenta válido y conocer la contraseña asociada a esa cuenta. Además, esa persona debe estar conectada desde un computador con el permiso para conectar a esa base de datos a través de esa cuenta específica. Una vez dentro, dependiendo del tipo de privilegios que disponga podrá realizar unas operaciones u otras.

En cuanto al funcionamiento de MySQL hay que tener en cuenta los siguientes conceptos:

- Base de datos relacionales
- Modelo de cliente-servidor

Respecto al primer punto, se explica anteriormente que se hace uso de una llave para relacionar los datos de una tabla con los de otra. En cuanto al modelo cliente-servidor, es en este último donde se almacenan los datos. Para acceder a los mismos, el cliente debe solicitarlo lanzando una petición al servidor requiriendo aquellos datos que desea obtener[9]. Para llevar a cabo esa petición, se debe realizar mediante una conexión, para la cual se puede usar el cliente MySQL o conectar el programa mediante controladores JDBC o bibliotecas ORM como Hibernate.

Mockup

Un mockup es una representación visual de un diseño o proyecto que se utiliza para evaluar su apariencia y funcionalidad[13]. A diferencia de un prototipo completamente funcional, un mockup se centra principalmente en la apariencia visual y en cómo se presentará el diseño final, sin tener ningún tipo de backend desarrollado. Además, al ser mostrado a clientes, estos pueden proponer sus propias ideas de mejora y así aproximarse al concepto final que se desea con el mínimo esfuerzo.

Para el desarrollo correcto de un Mockup hay que tener clara la estructura visual de lo que se quiere representar (sito web, aplicación móvil…), los elementos estáticos que la van a formar (botones, colores usados, imágenes, logotipos…) y la interactividad entre las diferentes ventanas, quedando todo el diseño lo más próximo al proyecto final, sin buscar la réplica completa.

Entre las principales ventajas de usar este tipo de herramientas para los proyectos tenemos:

- Diseño de un producto con poco esfuerzo y recursos (ahorro en gastos)[11][12].
- Rapidez para mostrar los resultados a los clientes de una manera realista[11].
- Estudio de la experiencia del cliente para identificar posibles errores o mejoras[11].
- Gran variedad de herramientas para la creación de mockups[13].

Respecto a este último punto, se muestra un listado de herramientas que permiten la creación, habiendo optado en la presentación de este proyecto por usar Figma[11]:

- Figma
- Ninjamock
- InstaMockup
- FrameAPP
- Balsamiq
- Auxure
- Pencil
- Canva
- Freepik
- Pexels

Desarrollo del proyecto

Arquitectura del sistema.

La arquitectura del sistema viene implícita en el propio software de desarrollo, usando el popular patrón Modelo-Vista-Controlador (Spring MVC), siendo necesario para el desarrollo de este tipo de aplicación un proyecto de Spring Boot, el cual contiene estas dos dependencias como Starters:

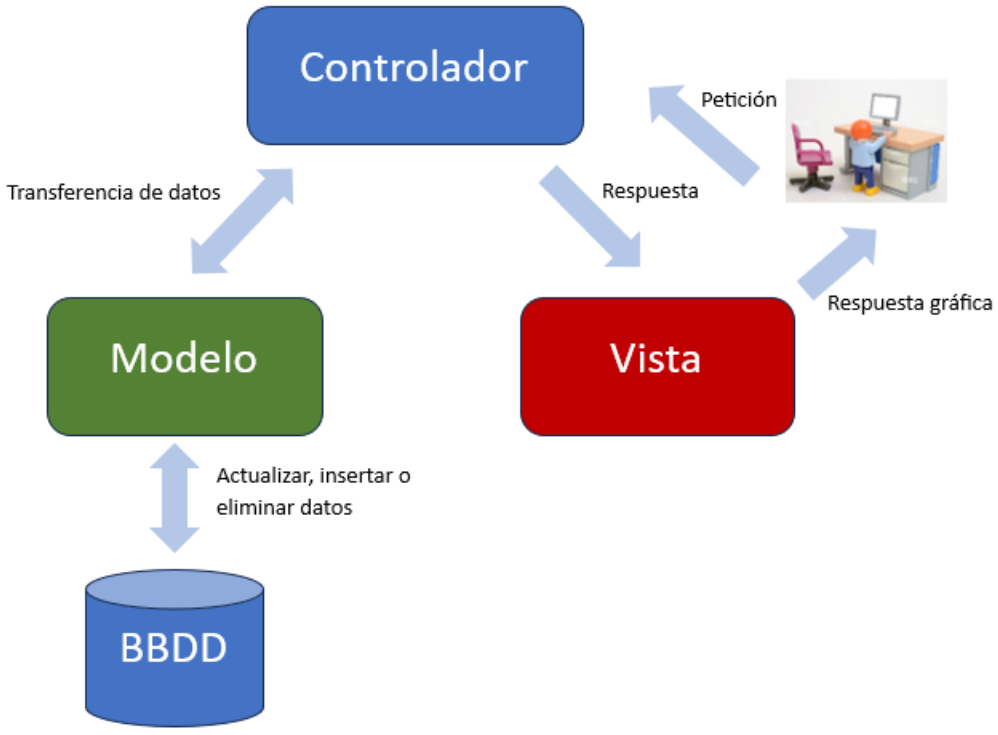
```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Thymeleaf

Spring Web

Al tratarse de un framework MVC, trabaja sobre todo con el concepto de Controlador (Controller) que es el encargado de soportar todas las peticiones Web y redirigirlas a los componentes que sean necesarios.

Explicada el tipo de arquitectura que se usa, se expone mediante imagen, desglosando paso a paso el flujo de datos:

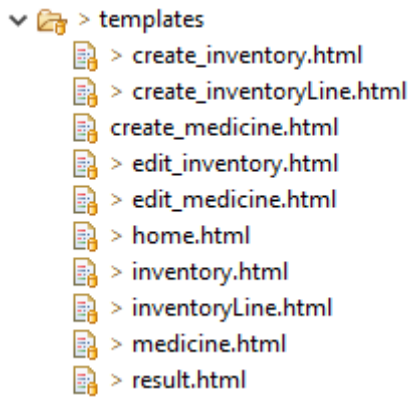


1. El usuario interactúa con las páginas Thymeleaf en el navegador.
2. El controlador de Spring Boot recibe las solicitudes HTTP y llama a los servicios correspondientes.
3. Los servicios interactúan con los repositorios para realizar operaciones en la base de datos.

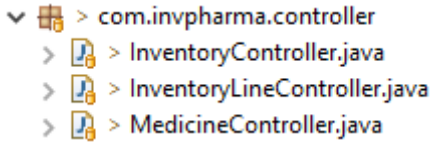
4. La información se devuelve a los controladores, que la pasan a las páginas Thymeleaf para su presentación.

En la imagen se pueden diferenciar por colores las capas de la arquitectura, siendo estas:

✓ **CAPA VISTA:** para el desarrollo de esta capa se usa Thymeleaf y Bootstrap, siendo codificadas hasta diez páginas HTML o templates. Su ubicación en el proyecto está en "*src/main/resources/templates*". Su función es interactuar entre el controlador y el usuario para mostrar los datos.

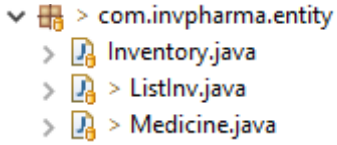


✓ **CAPA DEL CONTROLADOR:** Gestiona las solicitudes y las respuestas. Está compuesta por los Controllers, componentes que actúan como intermediarios entre el modelo y la vista. Se encargan de procesar la entrada del usuario, actualiza el modelo en consecuencia y también actualizan la vista para reflejar los cambios en el modelo. En resumen, los controladores gestionan el flujo de datos y controlan cómo se actualiza tanto el modelo como la vista. Para este proyecto se han dispuesto tres controladores situados en el paquete "*com.invpharma.controller*".



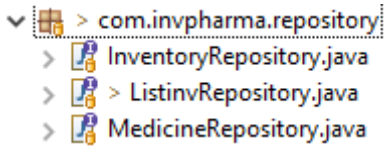
✓ **CAPA MODELO (ACCESO A DATOS):** Formada por dos componentes:

- Entidades: son las clases que representan las tablas de la base de datos, en nuestro caso, las tres tablas que almacenan todos los datos. Se caracterizan por estar anotadas con el valor "***@Entity"***, lo que indica a JPA (Java Persistence API) que esta clase representa una tabla en la base de datos. Los campos de la clase están mapeados en columnas dentro de la tabla, y cada instancia de la clase se almacenará como una fila en esa tabla.
  El proyecto cuenta también con tres entidades.

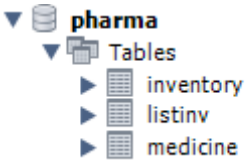


- Repositorios: los repositorios son interfaces que proporcionan un conjunto de métodos para realizar operaciones de acceso a datos en una base de datos. Estos

repositorios son parte de Spring Data, que es un conjunto de proyectos que simplifican el acceso a datos en aplicaciones basadas en Spring.
En particular, Spring Data JPA es un subproyecto de Spring Data que facilita la implementación de repositorios basados en JPA (Java Persistence API), permitiendo el acceso a bases de datos relacionales.
Con el comando JpaRepository, Spring Data JPA proporciona métodos estándar para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en las entidades. En resumen, ofrecen una forma sencilla y consistente de interactuar con bases de datos reduciendo la cantidad de código necesario para realizar operaciones de acceso a datos.
Como en el caso anterior, contamos con un repositorio por cada entidad.



✓ **BASE DE DATOS:** Compuesta por las tres tablas que almacenan los datos.

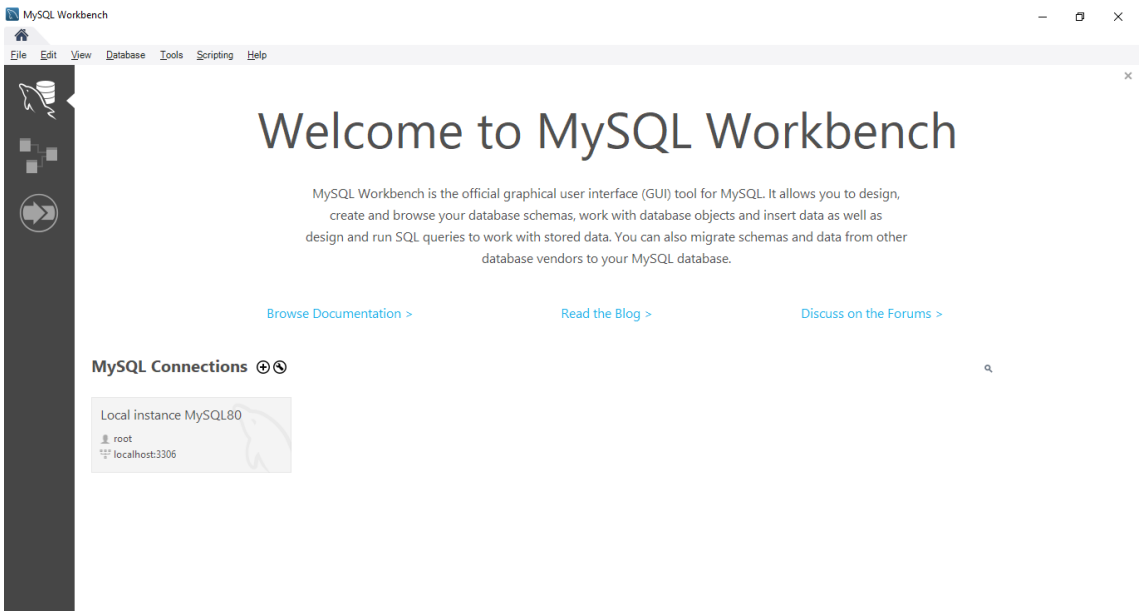


Diseño de la base de datos

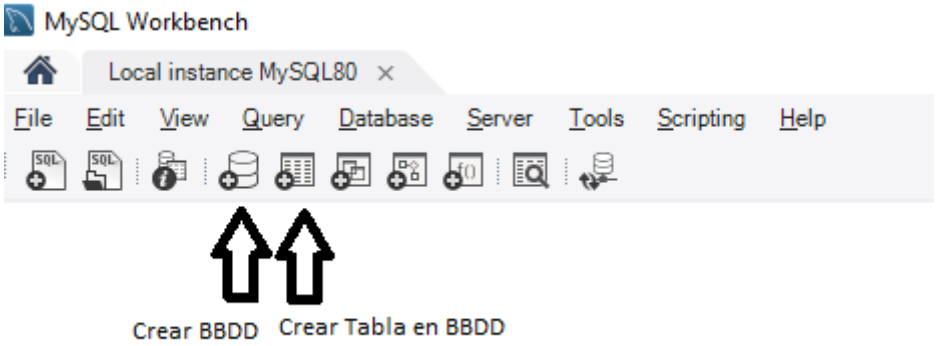
En este apartado continuamos el párrafo anterior, explicando la funcionalidad de cada una de las tablas, así como de las columnas por las que está compuesta cada una de ellas.

Para la creación de las mismas se ha usado el programa MySQL Workbench, el cual se descarga desde esta web https://www.mysql.com/products/workbench/ y una vez descargado se instala.

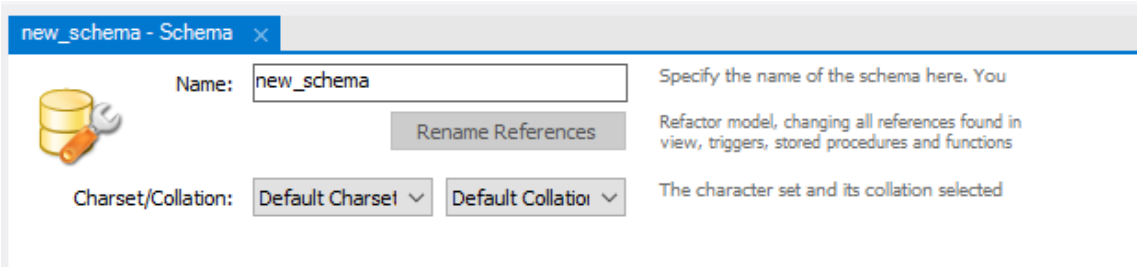
Terminada la instalación, la pantalla inicial de Workbench es la siguiente:



Una vez que se entra en la conexión configurada (cuya contraseña de root se introduce en la instalación del programa), se puede crear una nueva BDDD y añadir tablas a la misma con estos dos botones del menú superior:



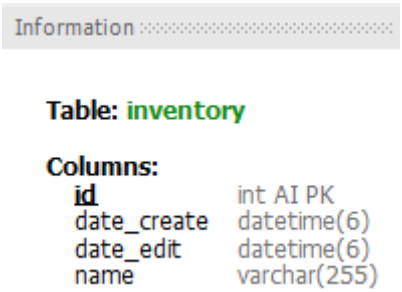
Pulsado el botón de “Crear BBDD” aparece esta ventana, donde se configura el nombre de la base de datos.



Una vez creada la base de datos, se han agregado las tres tablas a través del botón “Crear Tabla en BBDD”:

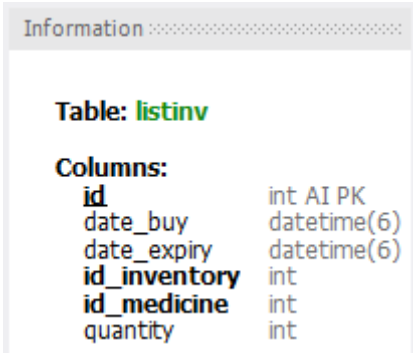
- ✓ INVENTORY: la funcionalidad de esta tabla es guardar los inventarios que se pueden llegar a disponer, es decir, una misma persona puede disponer de diferentes domicilios o lugares, teniendo en cada uno de ellos un inventario diferente de medicamentos. Pues esta tabla es la que define ese domicilio o lugar donde se pueda disponer de un inventario. Se compone de las siguientes columnas:
  - id: Clave primaria única que identifica cada inventario.
  - date_create: Fecha de creación del inventario.
  - date_edit: Fecha de la última modificación del inventario.
  - name: Nombre del inventario.

  A continuación, se muestra una imagen con el tipo de dato de las anteriores columnas.



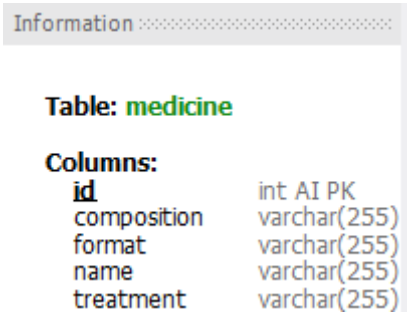
- ✓ LISTINV: La tabla central del proyecto, pues en ella se guarda el listado de medicamentos que tendrá cada uno de los inventarios creados previamente. Esta tabla tiene relación con columnas de las otras tablas para poder rescatar los datos correctos. Las columnas que la forman son:
  - id: Clave primaria única que identifica cada entrada en la lista de inventario.
  - date_buy: Fecha de compra del medicamento.
  - date_expiry: Fecha de caducidad del medicamento.
  - id_inventory: Id que se relaciona con la columna “id” de la tabla “inventory”.
  - id_medicine: Id que se relaciona con la columna “id” de la tabla “medicine”.
  - quantity: Cantidad que se dispone del fármaco adquirido.

  Imagen con el tipo de datos:

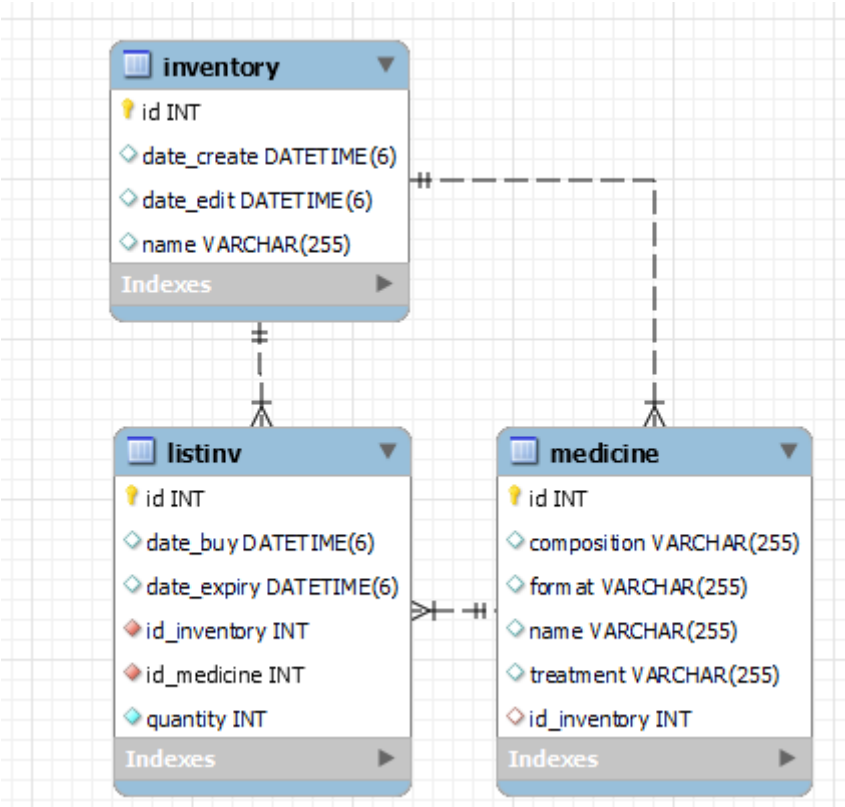


- ✓ MEDICINE: Tabla en donde se introducen todos los medicamentos que puedan existir en un inventario o puedan ser posiblemente adquiridos. Solo se añaden a los inventarios los medicamentos que se encuentren en esta tabla, teniendo además datos relativos de dicho fármaco. Sus columnas son las siguientes:
  - id: Clave primaria única que identifica cada medicamento.
  - composition: Composición química del medicamento.
  - format: Formato del medicamento (por ejemplo: sobres, pastillas, líquido).
  - name: Nombre del medicamento.
  - treatment: Información indicada por el prospecto o el propio médico de cómo debe ser la administración del fármaco.

  Los tipos de datos anteriores se muestra en esta imagen.

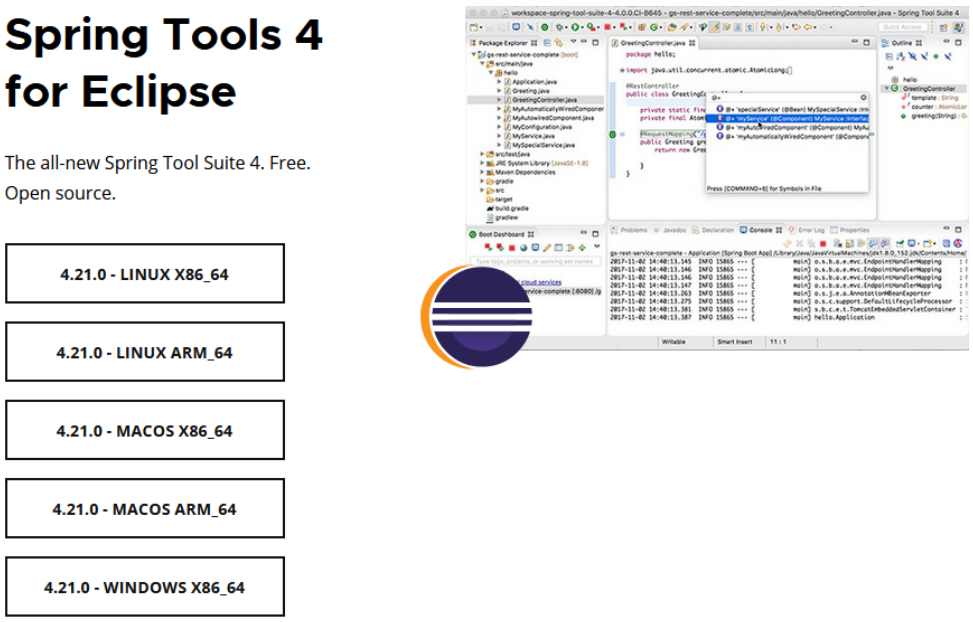


Por último, se muestra el esquema relacional de las tablas anteriores.

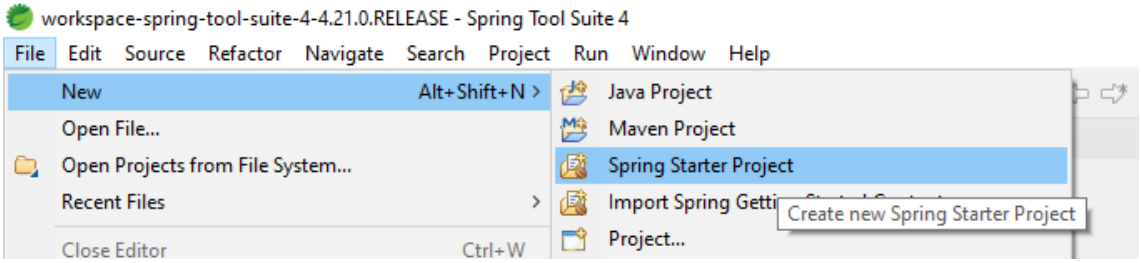


Creación del proyecto con Spring Starter Project

En primer lugar, para el desarrollo Java se ha optado por usar el software Eclipse, debido a que se trata de un software de código abierto completamente gratuito. A través de la página oficial de spring se obtiene el conjunto de herramientas "Spring Tools 4 for Eclipse" (https://spring.io/tools), diseñado especialmente para facilitar el desarrollo de aplicaciones de este tipo.

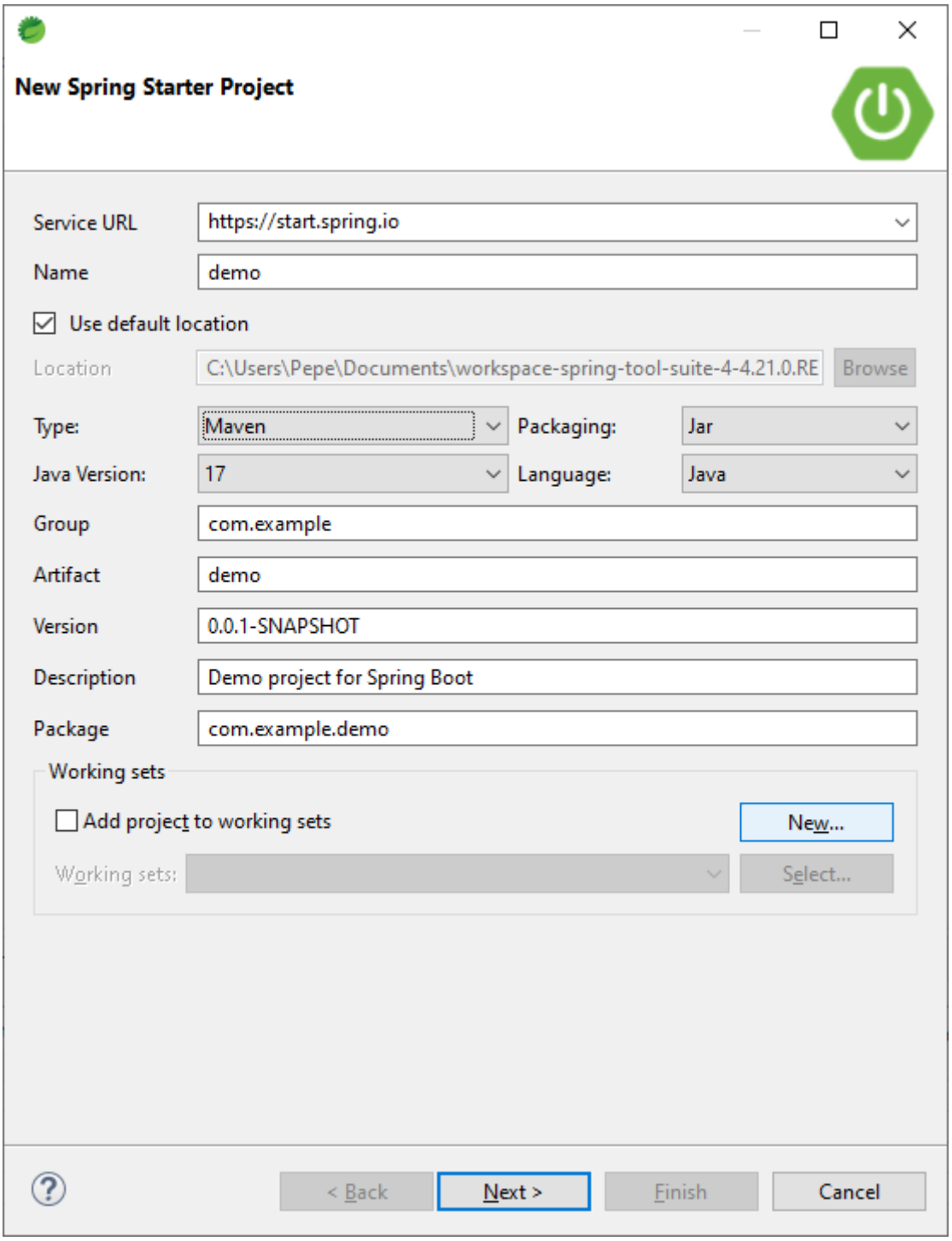


El primer paso del proyecto es crear el proyecto a través del anterior software "Spring Tool Suite". Una vez que se ejecuta y se abre, se debe seleccionar de menú File → New →Spring Starter Project



Se abre una nueva ventana donde se rellenan los siguientes apartados:

- ✓ Name: Nombre del proyecto
- ✓ Type: Se escoge la opción de Maven.
- ✓ Java Versión: La última es la 17.
- ✓ Group: El grupo es la identificación del paquete de la aplicación. Se utiliza para organizar las clases en un espacio de nombres único.
- ✓ Artifact: El artefacto es esencialmente el nombre del proyecto. Este es el identificador único para la aplicación dentro del contexto de Maven.
- ✓ Versión: Versión actual del proyecto.
- ✓ Descripción: Breve descripción de la funcionalidad que realiza el proyecto.
- ✓ Package: Nombre del paquete principal para las clases. El paquete se crea bajo el directorio de código fuente y ayuda a organizar y estructurar el código.



Rellenados los anteriores campos, pulsando el botón “Next” se cambia a la ventana donde se debe seleccionar las dependencias necesarias para poder desarrollar el proyecto. En nuestro caso se han añadido las siguientes:

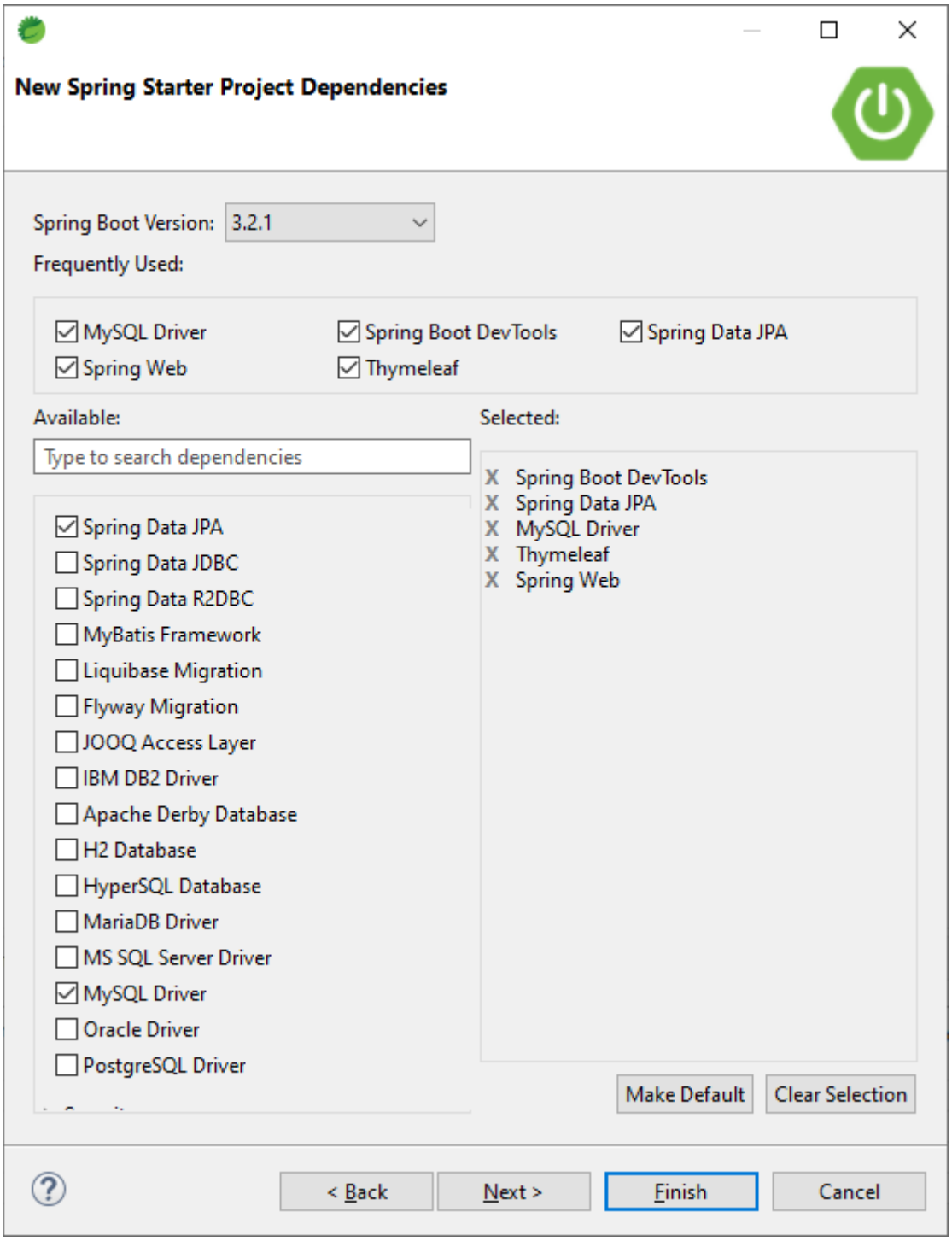
- ✓ MySQL Driver: Es el controlador necesario para poder trabajar con la base de datos de MySQL. Al agregar la dependencia "MySQL Driver", se incluyen las bibliotecas necesarias para que Spring pueda establecer una conexión y realizar operaciones en una base de datos MySQL.

- ✓ Spring Web: proporciona funcionalidades relacionadas con el desarrollo de aplicaciones web, tales como estas:
  - **Spring MVC (Model-View-Controller):** La dependencia incluye el framework Spring MVC, que facilita la creación de controladores, vistas y modelos para construir aplicaciones web de manera estructurada y modular.
  - **Embebido en el Servidor Web**: Spring Boot incluye un servidor web embebido (por defecto, Tomcat) que facilita la implementación y ejecución de aplicaciones web sin necesidad de configuración externa de un servidor web.
  - **Manejo de Solicitudes y Respuestas:** Proporciona abstracciones para manejar solicitudes HTTP entrantes y generar respuestas, lo que facilita el desarrollo de controladores HTTP.

- ✓ Spring Boot Dev Tools: conjunto de herramientas diseñadas para proporcionar características que facilitan la implementación y actualización rápida de la aplicación sin tener que reiniciar manualmente el servidor de aplicaciones, destacando entre ellas:
  - **Reinicio Automático (LiveReload):** Una de las características más destacadas es la capacidad de reiniciar automáticamente la aplicación cuando se detectan cambios en el código fuente. Esto agiliza el proceso de desarrollo, ya que se pueden ver los cambios reflejados en la aplicación sin tener que detener y reiniciar manualmente el servidor.
  - **Conexión Remota**: DevTools incluye un servidor remoto que permite conectarse a la aplicación desde un entorno de desarrollo integrado (IDE) como IntelliJ IDEA o Eclipse. Esto facilita la depuración remota y otras operaciones sin tener que acceder directamente al servidor.

- ✓ Spring Data JPA: proporciona soporte para trabajar con el mapeo objeto-relacional (ORM) utilizando Java Persistence API (JPA). JPA es una especificación de Java que describe el manejo de datos en una base de datos relacional a través de objetos Java. Como características principales:
  - **Mapeo de Entidades:** Spring Data JPA permite mapear clases Java a tablas de bases de datos, y las instancias de esas clases a filas en esas tablas. Esto se realiza mediante anotaciones en las clases Java que representan entidades.
  - **Repositorios JPA**: Spring Data JPA proporciona repositorios basados en interfaces que permiten realizar operaciones de consulta y manipulación de datos de manera sencilla. Estos repositorios eliminan la necesidad de escribir consultas SQL manualmente, ya que se pueden definir métodos en las interfaces que se traducen automáticamente a consultas JPA.
  - **Consultas Derivadas:** Spring Data JPA permite la generación automática de consultas JPA basadas en el nombre de los métodos en los repositorios. Esto significa que se pueden definir consultas simplemente utilizando la convención de nombres de métodos, lo que simplifica la escritura de consultas.
  - **Soporte Transaccional**: Spring Data JPA integra con el sistema de gestión de transacciones de Spring, permitiendo la gestión transaccional de las operaciones en la base de datos. Las transacciones garantizan la consistencia de los datos y la integridad de la base de datos.

- ✓ Thymeleaf: se utiliza para integrar el motor de plantillas Thymeleaf en una aplicación web. Algunas de las características y usos clave de Thymeleaf incluyen:
  - **Sintaxis Clara y Legible**: Thymeleaf utiliza una sintaxis sencilla y fácil de leer que se integra directamente en el HTML. Esto facilita la creación de plantillas y la mezcla de contenido estático y dinámico.
  - **Soporte para Expresiones**: Thymeleaf permite el uso de expresiones para acceder a datos del modelo y mostrarlos en las vistas. Estas expresiones pueden evaluar variables, realizar iteraciones sobre listas y mapas, y ejecutar lógica condicional directamente en las plantillas.
  - **Integración con Spring:** Thymeleaf se integra de manera natural con Spring Framework y Spring Boot. Se puede usar Thymeleaf en combinación con controladores de Spring MVC para construir vistas dinámicas que interactúan con el modelo de la aplicación.
  - **Procesamiento del Lado del Servidor**: Thymeleaf se procesa en el lado del servidor, lo que significa que las plantillas se renderizan en el servidor antes de ser enviadas al cliente. Esto facilita la generación dinámica de contenido en función de la lógica del servidor.



Al pulsar sobre el botón “Finish”, comienza a generar el proyecto, creando una estructura básica que posteriormente se ha aumentado al crear los diferentes paquetes del “Frontend” y las templates del “Bakcend”.

Como nota a destacar en este paso es el archivo “pom.xml”, el cual contiene la información esencial sobre el proyecto, como su identificación (nombre, versión, grupo, etc.), la URL del proyecto, el desarrollador principal, y otros detalles. Además, Maven gestiona las dependencias a través de este archivo, es decir, en él se encuentran todas las dependencias anteriores.

```
https://maven.apache.org/xsd/maven-4.0.0.xsd (xsi:schemaLocation)
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.1.0</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.invpharma</groupId>
    <artifactId>Invpharma</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>Invpharma</name>
    <description>Pharmaceutical Inventory of home</description>
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>
```

Integración con MySQL

Configuración de la conexión

Para poder guardar, actualizar o eliminar los datos conforme indique el usuario a través de la vista, se debe conectar la base de datos con el proyecto creado. Todo ello se hace mediante el archivo "*application.properties*", el cual se encuentra en el directorio "*src/main/resources*".

Una vez abierto el archivo, se han escrito los siguientes comandos para que quede configurada la conexión con la base de datos previamente creada:

- spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver: Especifica la clase del controlador JDBC que Spring Boot debe usar para conectarse a la base de datos.
- spring.datasource.url=jdbc:mysql://localhost:3306/pharma: Define la URL de conexión a la base de datos. En este caso, está en el localhost en el puerto 3306 y se llama "pharma".
- spring.datasource.username=root: Credenciales de usuario.
- spring.datasource.password=root: Contraseña para autenticarse en la base de datos.
- spring.jpa.hibernate.ddl-auto=update: Controla la estrategia de creación y actualización de la base de datos por parte de Hibernate. Al tener el comando "update", actualiza automáticamente la base de datos según los cambios en las entidades.
- spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect: Especifica el dialecto de Hibernate que se ha utilizado para interactuar con la base de datos
- spring.jpa.show-sql=true: Indica si se deben mostrar las consultas SQL generadas por Hibernate en la consola. Para la realización de pruebas y comprobar que las consultas son correctas es útil, una vez finalizada, se puede cambiar el valor a "false".

El primer paso de conexión con la base de datos se completa con esta fase, en el siguiente punto, a través de la creación de las los paquetes “Entity” y “Repository” se ha terminado de configurar la conexión y así queda todo conectado para un correcto funcionamiento.

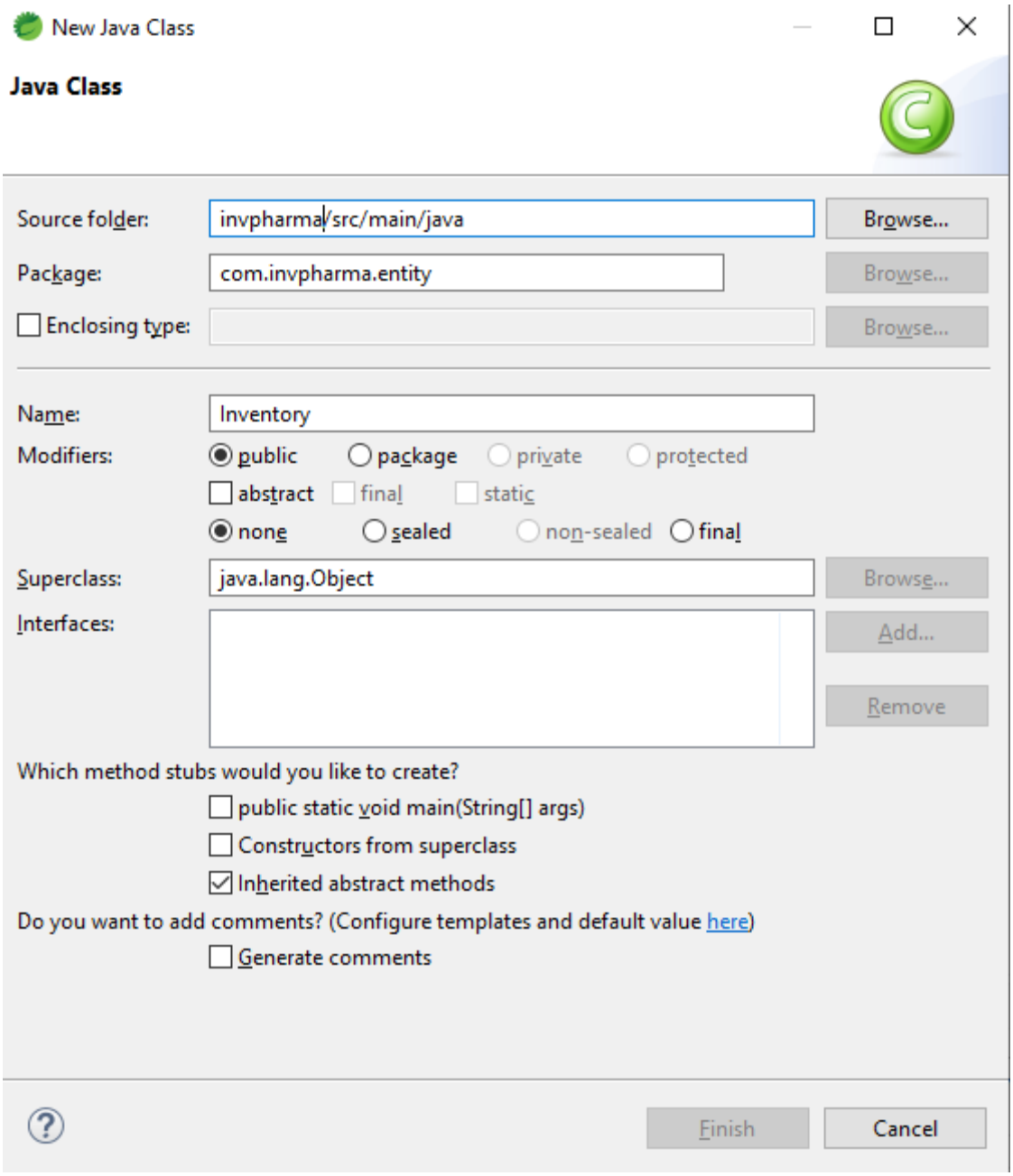
Desarrollo del Backend con Spring Boot.

Para el desarrollo de la aplicación basada en JPA y Spring MVC, las entidades, repositorios y controladores juegan roles específicos en el diseño y desarrollo de la aplicación.

A continuación, se procede a desglosar cada apartado del “Backend” y el código empleado para su desarrollo.

Entidades

Se han creado tres entidades, una por cada tabla de base de datos. Para ello se ha tenido que crear un paquete dentro de “*src/main/java*” llamado “*entity*”, y dentro del mismo las clases correspondientes a cada entidad.



INVENTORY.

```
@Entity
@Table(name="inventory")
public class Inventory {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Integer id;

	private String name;


	@Temporal(TemporalType.DATE)
	@DateTimeFormat(pattern = "yyyy-MM-dd") // Date format
	private Date date_create = new Date();
	@Temporal(TemporalType.DATE)
	@DateTimeFormat(pattern = "yyyy-MM-dd") // Date format
	private Date date_edit = new Date();
```

```
public Inventory() {
		super();
	}
	public Inventory(Integer id, String name, Date date_create, Date date_edit) {
		super();
		this.id = id;
		this.name = name;
		this.date_create = date_create;
		this.date_edit = date_edit;
	}
	/**
	 * @return the id
	 */
	public Integer getId() {
		return id;
	}
	/**
	 * @param id the id to set
	 */
	public void setId(Integer id) {
		this.id = id;
	}
	/**
	 * @return the name
	 */
	public String getName() {
		return name;
	}
	/**
	 * @param name the name to set
	 */
	public void setName(String name) {
		this.name = name;
	}
	/**
	 * @return the date_create
	 */
	public Date getDate_create() {
		return date_create;
	}
	/**
	 * @param date_create the date_create to set
	 */
	public void setDate_create(Date date_create) {
		this.date_create = date_create;
	}
	/**
	 * @return the date_edit
	 */
	public Date getDate_edit() {
		return date_edit;
	}
	/**
	 * @param date_edit the date_edit to set
	 */
	public void setDate_edit(Date date_edit) {
		this.date_edit = date_edit;
	}
    /**
	 * @return the listInvs
	 */
}
```

Lo más relevante a destacar son las anotaciones “@Entity” y “@Table”. La primera de ellas indica que la clase anotada es una entidad JPA (Java Persistence API), que se mapea a una tabla en una base de datos relacional. Por otro lado, la anotación “@Table(name = "inventory")” se utiliza para especificar el nombre de la tabla en la base de datos a la que se mapea la entidad.

También remarcar la anotación “@Id” puesto que indica que este campo es la clave primaria de la entidad.

El resto del código de la entidad son los “getters/setters” correspondientes a los campos.

LISTINV

```
@Entity
@Table(name="listinv")

public class ListInv {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Integer id;

	@Column(name = "id_inventory") // Database column name
	private int idInventory;

	@Column(name = "id_medicine") // Database column name
	private int idMedicine;

	@Temporal(TemporalType.DATE)
	@DateTimeFormat(pattern = "yyyy-MM-dd") // Date format
	private Date date_buy = new Date();

	@Temporal(TemporalType.DATE)
	@DateTimeFormat(pattern = "yyyy-MM-dd") // Date format
	private Date date_expiry;

	private int quantity;

	// The 'listinv' and 'medicine' tables are related by the 'id_medicine' field,
	// which matches the 'id' in the 'medicine' table
	@OneToOne
	@PrimaryKeyJoinColumn(name = "id_medicine")
	private Medicine medicine;

	public Medicine getMedicine() {
		return medicine;
	}

	public void setMedicine(Medicine medicine) {
		this.medicine = medicine;
	}

	public ListInv() {
		super();
	}
	public ListInv(Integer id, int id_inventory, int id_medicine, Date date_buy, Date date_expiry, int
quantity) {
		super();
		this.id = id;
		this.idInventory = id_inventory;
		this.idMedicine = id_medicine;
		this.date_buy = date_buy;
		this.date_expiry = date_expiry;
		this.quantity = quantity;
	}
	/**
	 * @return the id
	 */
	public Integer getId() {
		return id;
	}
	/**
	 * @param id the id to set
	 */
	public void setId(Integer id) {
		this.id = id;
	}
	/**
	 * @return the id_inventory
	 */
	public int getId_inventory() {
		return idInventory;
	}
	/**
	 * @param id_inventory the id_inventory to set
	 */
```

```
	public void setId_inventory(int id_inventory) {
		this.idInventory = id_inventory;
	}
	/**
	 * @return the id_medicine
	 */
	public int getId_medicine() {
		return idMedicine;
	}
	/**
	 * @param id_medicine the id_medicine to set
	 */
	public void setId_medicine(int id_medicine) {
		this.idMedicine = id_medicine;
	}
	/**
	 * @return the date_buy
	 */
	public Date getDate_buy() {
		return date_buy;
	}
	/**
	 * @param date_buy the date_buy to set
	 */
	public void setDate_buy(Date date_buy) {
		this.date_buy = date_buy;
	}
	/**
	 * @return the date_expiry
	 */
	public Date getDate_expiry() {
		return date_expiry;
	}
	/**
	 * @param date_expiry the date_expiry to set
	 */
	public void setDate_expiry(Date date_expiry) {
		this.date_expiry = date_expiry;
	}
	/**
	 * @return the quantity
	 */
	public int getQuantity() {
		return quantity;
	}
	/**
	 * @param quantity the quantity to set
	 */
	public void setQuantity(int quantity) {
		this.quantity = quantity;
	}
}
```

El procedimiento de esta entidad es similar a la explicada anteriormente con la salvedad de las anotaciones “***@OneToOne***” y “***@PrimaryKeyJoinColumn***”. Este tipo de anotaciones se utilizan para definir una relación, indicando que hay una relación uno a uno con la entidad “Medicine”. La anotación ***“@PrimaryKeyJoinColumn(name = "id_medicine"”*** especifica que la columna "*id_medicine*" se utilizará como clave primaria en la tabla de la entidad ListInv. La principal funcionalidad de usar esta anotación es ayudar a rescatar las características de los medicamentos y formar un listado con todos los datos.

MEDICINE

```
@Entity
@Table(name="medicine")
public class Medicine {
    public enum MedicineFormat {
        Pomada("mg"), Liquido("ml"), Pastillas(""), Sobres("");

        private final String unit;
```

```
    MedicineFormat(String unit) {
        this.unit = unit;
    }

    public String getUnit() {
        return unit;
    }
}

     @Id
     @GeneratedValue(strategy = GenerationType.IDENTITY)
     private Integer id;

     @Column(name = "name") // Database column name
     private String name;

     @Enumerated(EnumType.STRING)
private MedicineFormat format;
     private String composition;
     private String treatment;

     public Medicine() {
              super();
     }
     public Medicine(Integer id, String name, MedicineFormat format, String composition, String treatment) {
              super();
              this.id = id;
              this.name = name;
              this.format = format;
              this.composition = composition;
              this.treatment = treatment;
     }
     /**
      * @return the id
      */
     public Integer getId() {
              return id;
     }

     /**
      * @return the name
      */
     public String getName() {
              return name;
     }
     /**
      * @param name the name to set
      */
     public void setName(String name) {
              this.name = name;
     }
     /**
      * @param id the id to set
      */
     public void setId(Integer id) {
              this.id = id;
     }
     /**
      * @return the format
      */
     public MedicineFormat getFormat() {
              return format;
     }
     /**
      * @param format the format to set
      */
     public void setFormat(MedicineFormat format) {
              this.format = format;
     }
     /**
      * @return the composition
      */
     public String getComposition() {
              return composition;
     }
              this.treatment = treatment;
     }
     /**
      * @param composition the composition to set
      */
     public void setComposition(String composition) {
              this.composition = composition;
     }
```

```
	/**
	 * @return the treatment
	 */
	public String getTreatment() {
		return treatment;
	}
	/**
	 * @param treatment the treatment to set
	 */
	public void setTreatment(String treatment) {
		this.treatment = treatment;
	}
	@Override
	public String toString() {
		return "Medicine [id=" + id + ", name=" + name + ", format=" + format + ", composition=" +
composition + ", treatment=" + treatment
				+ "]";
	}

}
```

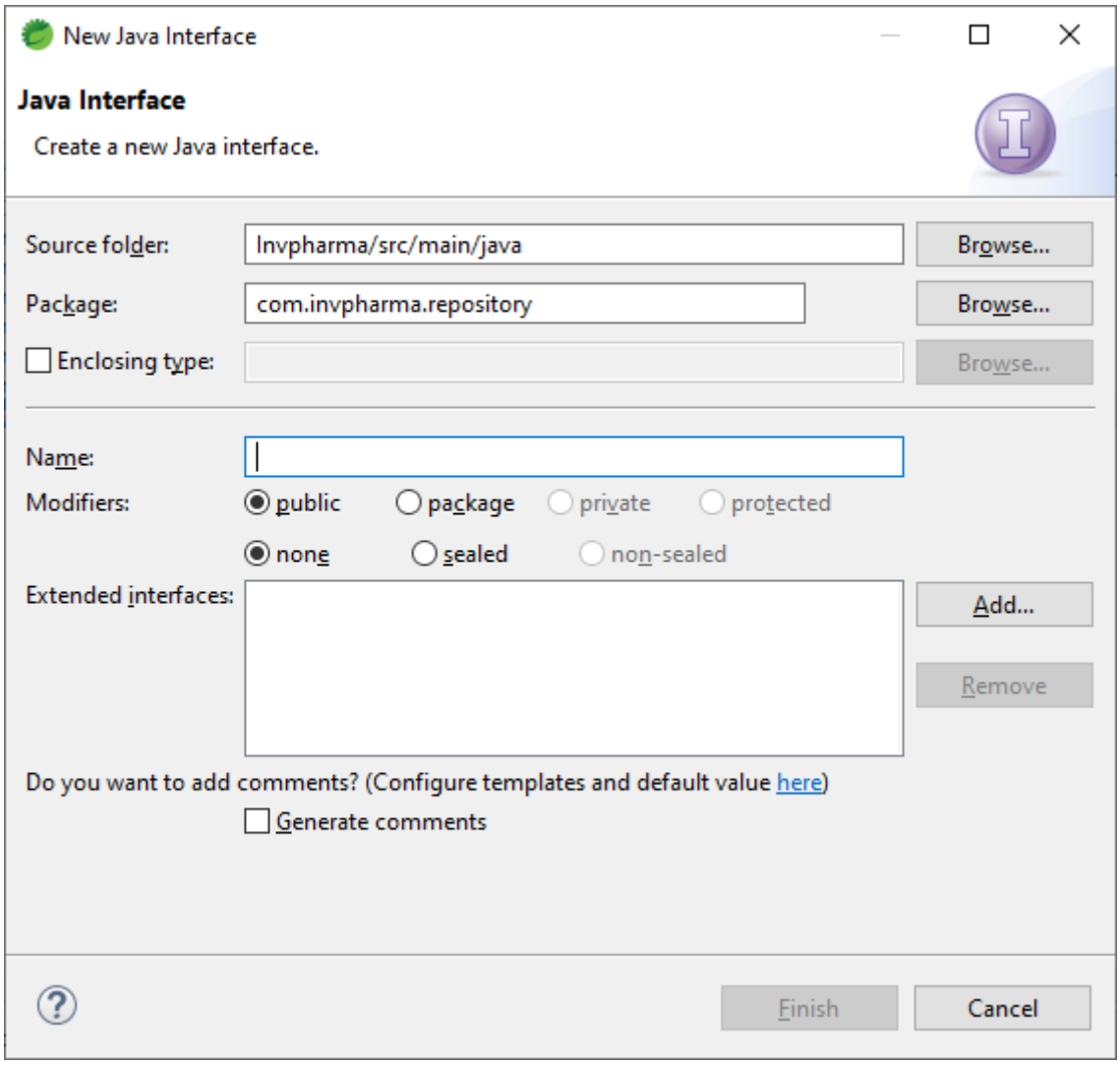
La última entidad trata sobre las características de los medicamentos, teniendo diferente de las anteriores la anotación "***@Enumerated***" que afecta al campo "*MedicineFormat*". El uso de esta anotación se debe a que el campo formato está limitado a una sería de valores predefinidos, los cuales están en la declaración de tipo de dato "*public enum MedicineFormat*".

Otra de las particularidades de este tipo de dato es que cada constante *enum* (Pomada, Líquido) tiene un atributo adicional que almacena la unidad asociada. El añadir esta opción es para distinguir en el "Frontend" la unidad de medida y dependiendo la que sea su funcionalidad visual se modifica.

Como última aclaración, "*EnumType.STRING*" se utiliza para indicar que se debe almacenar el nombre de la constante del *enum* como un String en la columna de la base de datos, ya que es el tipo de datos que tiene ese campo en la BBDD visto anteriormente.

Repositorios

Al igual que las entidades, se crean tres repositorios, los cuales son.



INVENTORY REPOSITORY

```
@Repository
public interface InventoryRepository extends JpaRepository<Inventory, Integer> {

}
```

El más básico, y está compuesto de estas partes a destacar:

- ✓ "***@Repository***": Esta anotación se utiliza para indicar que la interfaz es un bean de repositorio Spring. Los repositorios son componentes de Spring que proporcionan métodos para acceder y manipular datos en una base de datos.
- ✓ "***extends JpaRepository<Inventory, Integer>":*** Indica que hereda de JpaRepository, que es una interfaz proporcionada por Spring Data JPA. La interfaz JpaRepository proporciona métodos CRUD estándar (como save, findById, findAll, etc.) y otras operaciones comunes para trabajar con la base de datos.
  Respecto a los parámetros que contiene, el primero (<Inventory>) especifica la entidad asociada con este repositorio, en este caso, la clase Inventory.

El segundo parámetro (<Integer>) especifica el tipo del identificador principal (id) de la entidad, en este caso, se asume que el identificador es de tipo Integer.

LISTINV REPOSITORY

```
@Repository
public interface ListinvRepository extends JpaRepository<ListInv, Integer> {
        // Check if the medicine already exists in any inventory
    @Query("SELECT l.date_expiry, l.idInventory, m.name, i.name, m.treatment " +
            "FROM ListInv l " +
            "INNER JOIN Medicine m ON l.idMedicine = m.id " +
            "INNER JOIN Inventory i ON l.idInventory = i.id " +
            "WHERE m.name LIKE :name_medicine")
    List<Object[]> findInventoryInfoByMedicineName(@Param("name_medicine") String name_medicine);

        // Check if there is an entry with the combination of id_inventory and id_medicine
        @Query("SELECT CASE WHEN COUNT(li) > 0 THEN true ELSE false END FROM ListInv li WHERE li.idInventory = ?1
AND li.idMedicine = ?2")
        boolean existsByInventoryAndMedicine(int idInventory, int idMedicine);

        // Check for expired, soon-to-expire, or low stock medicine
        @Query("SELECT mi.name, mi.format, inv.name, li.date_expiry, li.quantity " +
                        "FROM ListInv li " +
                        "INNER JOIN Medicine mi ON li.idMedicine = mi.id " +
                        "INNER JOIN Inventory inv ON li.idInventory = inv.id " +
                        "WHERE li.date_expiry <= :expirydate " +
                        "OR (mi.format IN ('Pomada') AND li.quantity < 10) " +
                        "OR (mi.format IN ('Liquido') AND li.quantity < 50)" +
                        "OR (mi.format IN ('Sobres', 'Pastillas') AND li.quantity < 5)")
                List<Object[]> findExpiredOrLowQuantityMedicines(@Param("expirydate") Date expirydate);

        // Search by idInventory
    List<ListInv> findByIdInventory(int idInventory);

    // Search by idMedicine
    List<ListInv> findByIdMedicine(int idMedicine);

}
```

Este repositorio es el más desarrollado de los tres, puesto que implementa varias búsquedas específicas que se detallan a continuación:

- ✓ ***findInventoryInfoByMedicineName*:** Su objetivo es buscar información sobre el nombre de medicamento que se pasa como parámetro ***['@Param("name_medicine")']*** devolviendo una lista de objetos con la información solicitada de las diferentes tablas, en caso de existir.
- ✓ ***existsByInventoryAndMedicine:*** Se usa como validación previa a guardar un nuevo medicamento, comprobando si el que se intenta guardar (*idMedicine*) ya se encuentra almacenado en el inventario (*idInventory*), devolviendo un booleano a true si existe y false en caso contrario.
- ✓ ***findExpiredOrLowQuantityMedicines:*** La funcionalidad de esta búsqueda es rescatar los medicamentos caducados, próximos a caducar ['***@Param("expirydate")***'] o cuyas existencias o cantidad sea inferior a un límite preestablecido dependiendo del tipo de formato, devolviendo la consulta un listado de objetos con la información seleccionada de cada tabla.
- ✓ ***findByIdInventory:*** Busca el identificador único de ListInv que cumpla el parámetro de búsqueda ("*int idInventory*"), devolviendo una lista que posteriormente se usa para borrar inventarios con sus medicamentos o modificarlos.

- ✓ ***findByIdMedicine:*** Busca el identificador único de ListInv que cumpla el parámetro de búsqueda (“*int idMedicine*”), devolviendo una lista que se usa para borrar medicamentos de la BBDD, los cuales solo se podrán eliminar en caso de que dicho medicamento no se encuentre en ninguna línea de inventario.

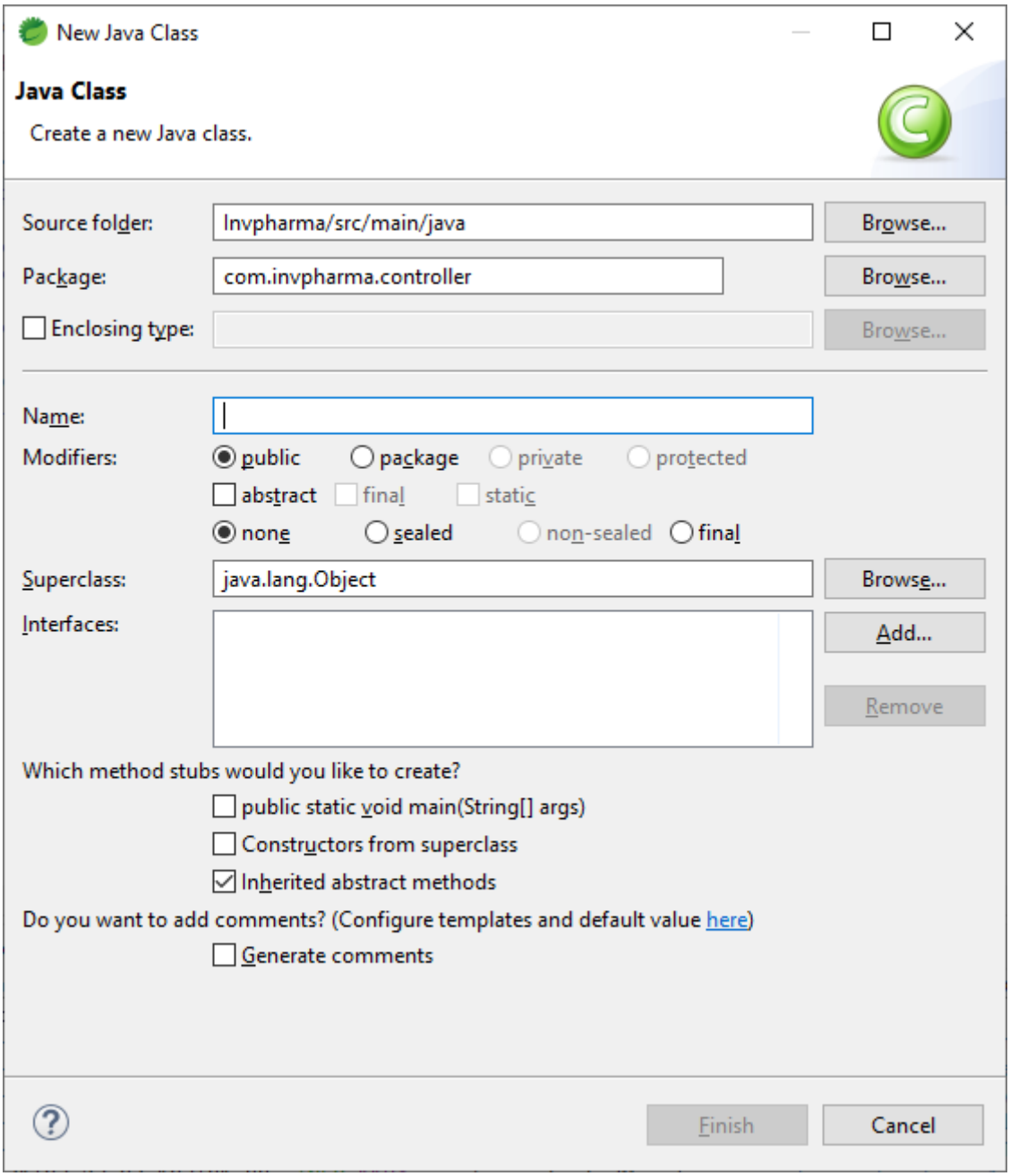
MEDICINE REPOSITORY

```
@Repository
public interface MedicineRepository extends JpaRepository<Medicine, Integer>{

	// Check if a medicine with the specific name and format already exists.
	@Query("SELECT CASE WHEN COUNT(me) > 0 THEN true ELSE false END FROM Medicine me WHERE
me.name = ?1 AND me.format = ?2")
	boolean existsByMedicine(String name,  MedicineFormat format);
}
```

Por último, en este repositorio se realiza una búsqueda para comprobar si un medicamento ya se encuentra guardado con el nombre y formato pasados como parámetros, devolviendo un booleano con valor true en caso de existir, no permitiendo duplicar el inventario de medicinas, o false añadiéndolo a BBDD.

Controladores.

Como último apartado del “Backend”, están a los controladores, los cuales también se han divido en tres partes que se explican seguidamente:



MEDICINE CONTROLLER

Controlador que se encarga de todas las operaciones relacionadas con el inventario exclusivo de medicamentos a la base de datos. Sus principales funciones son:

- ***Mostrar listado de medicamentos [‘@GetMapping("/medicine")’]***. Muestra en un listado ordenado alfabéticamente los medicamentos con sus características.
- ***Crear nuevo medicamento [‘@GetMapping("/create_medicine")’].*** Redirecciona a la página web que tiene los campos correspondientes para crear una nueva medicina.
- ***Guardar nuevos medicamentos [‘@PostMapping("/save_medicine")’].*** Teniendo en cuenta una validación previa antes de guardarlo, que consiste en no permitir duplicar medicamentos con el mismo nombre y formato, mostrando un mensaje de

error si se da dicha coincidencia. Mediante el siguiente comando, a través del repositorio, se guarda en base de datos "***medicineRepository*.save".**

- ***Editar los medicamentos existentes ['@GetMapping("/edit_medicine/{id}")'].***
- ***Eliminar medicamentos ['@GetMapping("/delete_medicine/{id}")'***]. La eliminación solo se realiza mientras el medicamento no se esté usando en alguno de los inventarios del hogar, evitando así una incongruencia y fallo en base de datos. El comando que permite la eliminación mediante el repositorio es ***"medicineRepository.delete".***
- ***Funcionalidades especiales del controller:***
    - **Ventana modal ['@GetMapping("")']:** Al iniciarse el programa se chequeará si existe en algún inventario algún medicamento caducado, próximo a caducarse o con falta de existencias, mostrando un listado de los mismos.
    - **Búsqueda de medicamento['@PostMapping("/result")']:** En la ventana inicial se permite realizar una búsqueda de un medicamento concreto para conocer si se tiene en alguno de los inventarios, evitando así el gasto innecesario de volver a comprarlo. Si se encuentra el medicamento se muestra un listado indicando el inventario donde se encuentra, pudiendo acceder directamente pulsando el enlace.

```java
@Controller
@RequestMapping("/home") //http:localhost:8080/medicine
public class MedicineController {

	@Autowired
	private MedicineRepository medicineRepository;

	@Autowired
	private ListinvRepository listInvRepository;


	@GetMapping("")
	public String home(Model model, HttpSession session) {

		// Calculate the upcoming expiration date (7 days)
		LocalDate expiryLimitDate = LocalDate.now().plusDays(7);

		// Change the format from LocalDate to Date
		Date expirydate =
Date.from(expiryLimitDate.atStartOfDay().atZone(ZoneId.systemDefault()).toInstant());

		// Search in ListInv for data with expiration dates approaching or already expired
		List<Object[]> expiredMedicinesorLowQuantity =
listInvRepository.findExpiredOrLowQuantityMedicines(expirydate);

		model.addAttribute("expiredMedicinesorLowQuantity", expiredMedicinesorLowQuantity);
	    return "home";
	}

	@GetMapping("/medicine") //http:localhost:8080/home/medicine
	public String medicine(Model model, HttpSession session) {
		List<Medicine> unsortedMedicines = medicineRepository.findAll();
		List<Medicine> medicine =
unsortedMedicines.stream().sorted(Comparator.comparing(Medicine::getName)).collect(Collectors.toList());

		// Display an error message if attempting to delete the medicine while it is still in any
inventory
	    String errorMessage = (String) session.getAttribute("errorMessage");
		model.addAttribute("medicine", medicine);
		model.addAttribute("errorMessage", errorMessage);

	    // Remove the error message from the session to prevent it from being displayed again
	    session.removeAttribute("errorMessage");
		return "medicine";
	}
	@GetMapping("/create_medicine") //http:localhost:8080/home/create_medicine
	public String create_medicine() {
		return "create_medicine";
	}
```

```
        @PostMapping("/save_medicine")
        public String save_medicine(HttpServletRequest request,@ModelAttribute("medicine") Medicine medicine,
RedirectAttributes redirectAttributes) {
                // Check if the medicine already exists in any inventory
            boolean exists = medicineRepository.existsByMedicine(medicine.getName(), medicine.getFormat());

            if (exists && request.getParameter("createSubmit") != null) {
                // // Check that a new medicine is being created and does not already exist
                 redirectAttributes.addFlashAttribute("messageError", "Existe el medicamento con esas
características");

                return "redirect:/home/create_medicine";
            } else {
                 medicineRepository.save(medicine);

                 return "redirect:/home/medicine";
            }
        }

        @GetMapping("/edit_medicine/{id}")
        public String edit_medicine(@PathVariable Integer id, Model model) {
                 Medicine med_aux = medicineRepository.getReferenceById(id);
                 model.addAttribute("medicamento",med_aux);
                 return "edit_medicine";
        }

        @GetMapping("/delete_medicine/{id}")
        public String delete_medicine(@PathVariable Integer id, HttpSession session) {
                  // Search in ListInv through the ID_medicine field to see if that medicine is in any inventory.
            List<ListInv> listInvWithMedicine = listInvRepository.findByIdMedicine(id);

            if (!listInvWithMedicine.isEmpty()) {
                // If exist a medicine, display an message error.
                 String errorMessage = "El medicamento no puede eliminarse. Debe eliminarlo de los inventarios
primero.";
                session.setAttribute("errorMessage", errorMessage);
            }else {
                 Medicine med_aux = medicineRepository.getReferenceById(id);
                 medicineRepository.delete(med_aux);
            }
                 return "redirect:/home/medicine";
        }

        @PostMapping("/result")
        public String search_medicine(@RequestParam("name_medicine") String name_medicine, Model model) {
                 // Modify the value of name_medicine to make it a pattern
            name_medicine = "%" + name_medicine + "%";
                 // Develop the medicine search
                  List<Object[]> inventoryInfo =
listInvRepository.findInventoryInfoByMedicineName(name_medicine);
            if (inventoryInfo.isEmpty()) {
                // If no data is found, display a message.
                model.addAttribute("errorMessage", "El medicamento no fue encontrado.");
            } else {
                        model.addAttribute("inventoryInfo", inventoryInfo);
            }
        return "result";
         }
}
```

INVENTORY CONTROLLER

Este controlador se encarga de manejar los diferentes inventarios de medicinas que se tienen, es decir, por cada vivienda donde se puedan almacenar una serie de medicamentos, se genera un nuevo inventario. Sus función es mostrar los inventarios que se pueden tener, añadiendo, eliminando o editando los mismos.

- ***Mostrar listado de inventarios ['@GetMapping("/inventory")'].*** Su función es acceder a la base de datos y recuperar de la tabla correspondiente los inventarios existentes para mostrarlos ordenados alfabéticamente.
- ***Crear nuevo inventario ['@GetMapping("/create_inventory")'].*** Redirecciona a la página web que se debe mostrar para rellenar los datos de un nuevo inventario.

- ***Guardar nuevo inventario [‘@PostMapping("/save_inventory")’].*** Guardará los datos del nuevo inventario generado.
- ***Edición del inventario [‘@GetMapping("/edit_inventory/{id}")’].*** Recuperará todas las medicinas que se encuentran asociadas a ese inventario para mostrarlas en un listado ordenado alfabéticamente.
- ***Eliminar un inventario [‘@GetMapping("/delete_inventory/{id}")’].*** Previo a eliminar el inventario, elimina los medicamentos que se encuentran asociados al mismo, para posteriormente eliminar de la tabla correspondiente el inventario.

```
@Controller
@RequestMapping("/home") //http:localhost:8080/home
public class InventoryController {

	@Autowired
	private InventoryRepository inventoryRepository;

	@Autowired
	private MedicineRepository medicineRepository;

	@Autowired
	private ListinvRepository listinvRepository;

	@GetMapping("/inventory") //http:localhost:8080/home/inventory
	public String inventory(Model model) {
		//Unordered list
		List<Inventory> unsortedInventory = inventoryRepository.findAll();
		//List sorted by the inventory name field
		List<Inventory> inventory =
unsortedInventory.stream().sorted(Comparator.comparing(Inventory::getName)).collect(Collectors.toList());
		model.addAttribute("inventory", inventory);
		return "inventory";
	}
	@GetMapping("/create_inventory") //http:localhost:8080/home/create_inventory
	public String create_inventory(Model model) {
		model.addAttribute("inventory", new Inventory());
		return "create_inventory";
	}

	@PostMapping("/save_inventory")
	public String save_inventory(@ModelAttribute("inventory") Inventory inventory) {
		inventoryRepository.save(inventory);
		return "redirect:/home/inventory";
	}

	@GetMapping("/edit_inventory/{id}")
	public String edit_inventory(@PathVariable Integer id, Model model, HttpSession session) {

		// Return the medicines associated with that inventory
		final List<ListInv> listinv = listinvRepository.findByIdInventory(id);

		// Return the medicine name from the 'medicine' table using the 'id_medicine'
		for (ListInv i: listinv) {
			final Optional<Medicine> medicine = medicineRepository.findById(i.getId_medicine());
			i.setMedicine(medicine.orElse(null));
		}

		//List sorted by the medicine name field
	    listinv.sort(Comparator.comparing(li -> li.getMedicine().getName()));
		model.addAttribute("listinv", listinv);
		session.setAttribute("id", id);

		return "edit_inventory";
	}
	@GetMapping("/delete_inventory/{id}")
	public String delete_inventory(@PathVariable Integer id) {
		Inventory inv_aux = inventoryRepository.getReferenceById(id);
		final List<ListInv> listinv = listinvRepository.findByIdInventory(id);
		listinvRepository.deleteAll(listinv);
		inventoryRepository.delete(inv_aux);
		return "redirect:/home/inventory";
	}

}
```

INVENTORY LINE CONTROLLER

Este es el controlador más complejo del programa, no tanto por las operaciones que realiza, que siguen siendo las básicas, sino por el código que contiene, debido a que debe acceder a las diferentes bases de datos para rescatar los datos de forma coherente y así poder mostrarse y realizar las diferentes acciones posibles.

- ***Crear un nuevo medicamento al inventario [‘@GetMapping("/create_inventoryLine")’].*** En este apartado se redirecciona a la página web que muestra los campos que añaden un nuevo medicamento, de los que previamente se han creado (Controlador de Medicine → [‘***@PostMapping("/save_medicine")***’]), a los inventarios ya existentes (Controlador de Inventory → [‘@PostMapping("/save_inventory")’]).
- ***Guardar nuevo medicamento [‘@PostMapping("/save_inventoryLine")].*** Guarda un nuevo medicamento al inventario, teniendo en cuenta que dicho medicamento no exista ya en ese mismo inventario.
- ***Editar medicamento de inventario [‘@GetMapping("/inventoryLine/{id}")].*** Se recuperan los datos correspondientes del medicamento en ese inventario, como la cantidad y fecha de caducidad, actualizando los mismos para posteriormente guardarlos.
- ***Eliminar medicamento de inventario [‘@GetMapping("/delete_inventoryLine/{id}")’].*** Se elimina el medicamento del inventario correspondiente.

```
@Controller
@RequestMapping("/home") //http:localhost:8080/home
public class InventoryLineController {

	@Autowired
	private InventoryRepository inventoryRepository;
	@Autowired
	private MedicineRepository medicineRepository;

	@Autowired
	private ListinvRepository listinvRepository;

	@GetMapping("/create_inventoryLine") //http:localhost:8080/home/create_inventory
	public String create_inventoryLine(Model model, HttpSession session) {
		// Medicine and inventory Unordered
		List<Inventory> unsortedInventories = inventoryRepository.findAll();
		List<Medicine> unsortedMedicines = medicineRepository.findAll();

		////Lists sorted by the inventory name and medicine name fields
		List<Inventory> inventories =
unsortedInventories.stream().sorted(Comparator.comparing(Inventory::getName)).collect(Collectors.toList());
		List<Medicine> medicines =
unsortedMedicines.stream().sorted(Comparator.comparing(Medicine::getName)).collect(Collectors.toList());

		model.addAttribute("listinv", new ListInv());
		model.addAttribute("inventoryAux", inventories);
		model.addAttribute("medicineAux", medicines);
		// The listnv ID is provided so that it can return to the corresponding inventory list by
pressing the back button
		Integer id = (Integer) session.getAttribute("id");
		model.addAttribute("id", id);
		return "create_inventoryLine";
	}
}
```

```
        @PostMapping("/save_inventoryLine")
        public String save_inventoryLine(HttpServletRequest request,@ModelAttribute("listinv") ListInv
listinv,RedirectAttributes redirectAttributes) {
                // Check if there is an entry with the combination of id_inventory and id_medicine
            boolean exists = listinvRepository.existsByInventoryAndMedicine(listinv.getId_inventory(),
listinv.getId_medicine());

            // Check that a new medicine is being added to the inventory and does not already exist
            if (exists && request.getParameter("createSubmit") != null) {
                // If an entry exists, display the following error message
                redirectAttributes.addFlashAttribute("messageError", "Ya existe ese medicamento en el
inventario");

            } else { //Add or udapte the inventory list
                listinvRepository.save(listinv);
                Inventory inventory = inventoryRepository.getReferenceById(listinv.getId_inventory()); //Update
date_edit of Inventory
                inventory.setDate_edit(new Date());
                inventoryRepository.save(inventory);
                // If it's edited, it returns to another URL (/home/inventoryLine)
                String referrer = request.getHeader("Referer");
                if (referrer != null && !referrer.isEmpty()) {
                        return "redirect:" + referrer;
                }
            }
            return "redirect:/home/create_inventoryLine";
        }

        @GetMapping("/inventoryLine/{id}") //http:localhost:8080/home/inventoryLine
        public String inventory(@PathVariable Integer id, Model model, HttpSession session) {
                // Edit a line of inventory
                ListInv list_aux = listinvRepository.getReferenceById(id);
                model.addAttribute("inventoryLine",list_aux);
                model.addAttribute("medicineAux",
medicineRepository.getReferenceById(list_aux.getId_medicine()));
                model.addAttribute("inventoryAux",
inventoryRepository.getReferenceById(list_aux.getId_inventory()));
                session.setAttribute("id", id);
                return "inventoryLine";
        }

        @GetMapping("/delete_inventoryLine/{id}")
        public String delete_inventoryxLine(@PathVariable Integer id,RedirectAttributes redirectAttrs) {
                ListInv inventoryLine_aux = listinvRepository.getReferenceById(id);
                Integer id_inventory = inventoryLine_aux.getId_inventory();
                listinvRepository.delete(inventoryLine_aux);

                // Add the value of id_inventory to enable redirection to the correct inventory.
                redirectAttrs.addAttribute("id_inventory", id_inventory);
                return "redirect:/home/edit_inventory/{id_inventory}";
        }

}
```

Desarrollo del Frontend con Thymeleaf y Bootstrap

El “Frontend” está compuesto por un total de diez plantillas, explicando la funcionalidad de cada una con el código que le acompaña a continuación.

Cada una de los plantillas han sido encabezadas por enlaces de Bootstrap, que agregan funcionalidades adicionales y el estilo CSS, usado para dar diseño y apariencia a la página, siendo diferenciados en el código al utilizar expresiones como “***btn***”, “***container***”, “***row***”, etc. Asimismo, también se ha añadido un enlace que indica que se está utilizando Thymeleaf, y por lo tanto, permite usar sus funciones, distinguiéndose en la codificación por estar precedidas por la expresión “***th***”.

En resumen, uso de Bootstrap y Thymeleaf permite mejorar la apariencia y funcionalidad de la página web.

HOME

Página de inicio, en ella se añaden unas imágenes que hacen más agradable la visual de la misma. Las opciones que se pueden implementar son:

- ***Botón de" Gestión de medicamentos":*** Se redirecciona a una nueva página donde se puede gestionar el inventario de todos los medicamentos utilizados.
- ***Botón de "Gestion de inventarios":*** Redirecciona a la página que muestra el listado de inventarios para poder administrarlos.
- ***Menú de búsqueda:*** Campo que permite la búsqueda rápida de un medicamento para comprobar si ya está guardado en alguno de los inventarios existentes. El resultado se muestra en un nuevo template.
- ***Ventana modal:*** Al iniciar está página, en el "Backend" se valida si hay algún medicamento en los inventarios que esté caducado, próximo a caducarse o a agotarse, y en el caso de existir, se abre y muestra en que inventario se encuentra, así como la fecha de caducidad y cantidad actual que posee

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
         <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
         <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
    <meta charset="UTF-8">
    <title>Inventario medicamentos</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
    <style>
        .card-img-top {
            object-fit: cover;
            width: 100%;
            height: 200px;
        }
    </style>
</head>
<body>

<div class="container">
    <div class="row justify-content-center mt-5">
        <div class="col-md-4 mb-4">
            <div class="card">
                <img src="/medicamentos.jpg" class="card-img-top" alt="Imagen medicamentos">
                <div class="card-body">
                    <h5 class="card-title">Gestión medicamentos</h5>
                    <a th:href="@{/home/medicine}" class="btn btn-primary">Ir a la gestión de medicamentos</a>
                </div>
            </div>
        </div>
        <div class="col-md-4 mb-4">
            <div class="card">
                <img src="/inventario.jpg" class="card-img-top" alt="Imagen inventarios">
                <div class="card-body">
                    <h5 class="card-title">Gestión inventarios</h5>
                    <a th:href="@{/home/inventory}" class="btn btn-info">Ir a la gestión de inventarios</a>
                </div>
            </div>
        </div>
        <div class="text-center mb-4">
         <form th:action="@{/home/result}" method="post">
                    <input type="text" name="name_medicine" placeholder="Nombre medicamento">
          <input class="btn btn-primary" type="submit" value="Buscar">
                          </form>
                    </div>
    </div>
</div>
```

```
<!-- Modal window -->
<div th:if="${expiredMedicinesorLowQuantity}">
        <!-- Ventana modal -->
        <div class="modal fade" id="expirationModal" tabindex="-1" aria-labelledby="expirationModalLabel" aria-
hidden="true">
            <div class="modal-dialog">
                <div class="modal-content">
                    <div class="modal-header">
                        <h5 class="modal-title" id="expirationModalLabel">Medicamentos Caducados, Próximos a
Caducar o Agotarse </h5>
                        <button type="button" class="btn-close" data-bs-dismiss="modal" aria-
label="Close"></button>
                    </div>
                    <div class="modal-body">
                        <!-- Lista de medicamentos próximos a caducar o caducados -->
                        <table class="table">
                            <thead>
                                <tr>
                                    <th scope="col">Nombre del Medicamento</th>
                                    <th scope="col">Nombre del Inventario</th>
                                    <th scope="col">Fecha de Caducidad</th>
                                    <th scope="col">Cantidad</th>
                                    <th scope="col">Formato</th>
                                </tr>
                            </thead>
                            <tbody>
                                <tr th:each="expiredOrQuantity : ${expiredMedicinesorLowQuantity}">
                                            <td th:text="${expiredOrQuantity[0]}"></td>
                                                <td th:text="${expiredOrQuantity[2]}"></td>
                                                <td th:text="${#dates.format(expiredOrQuantity[3],
'dd/MM/yyyy')}"></td>
                                                <td th:text="${expiredOrQuantity[4]}"></td>
                                                <td th:if="${expiredOrQuantity[1].getUnit() != ''}"
th:text="${expiredOrQuantity[1].name() + ' (' + expiredOrQuantity[1].getUnit() + ')'}"></td>
                                                                                <td
th:if="${expiredOrQuantity[1].getUnit() == ''}" th:text="${expiredOrQuantity[1].name()}"></td>
                                            </tr>
                            </tbody>
                        </table>
                    </div>
                    <div class="modal-footer">
                        <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Cerrar</button>
                    </div>
                </div>
            </div>
        </div>
    </div>

<script th:inline="javascript">
    $(document).ready(function() {
        // Check if the list expiredMedicinesorLowQuantity is empty
        var hasData = /*[[${expiredMedicinesorLowQuantity != null && expiredMedicinesorLowQuantity.size() > 0}]]*/
false;
        if (hasData) {
            // Activate the modal window if there is data
            var expirationModal = new bootstrap.Modal(document.getElementById('expirationModal'));
            expirationModal.show();
        }
      });
</script>
</body>
</html>
```

RESULT

Ventana que muestra un listado con los inventarios donde se encuentra el medicamento buscado. En caso de no existir, aparece un mensaje.

En el listado, presenta el nombre del inventario en formato de enlace, para, en caso de pulsarlo, redirigirse directamente al listado de medicinas del mismo.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Buscar medicamento</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
</head>
<body>
    <h1>Resultados de la Búsqueda</h1>

    <!-- Sección para mostrar un mensaje de error -->
         <div th:if="${errorMessage}" class="alert alert-danger">
         <p th:text="${errorMessage}"></p>
         </div>
       <table class="table">
        <thead>
<tr>
                <th scope="col">Nombre del Medicamento</th>
                <th scope="col">Nombre del Inventario</th>
                <th scope="col">Fecha de Caducidad</th>
                <th scope="col">Tratamiento</th>
            </tr>
        </thead>
        <tbody>
            <tr th:each="item : ${inventoryInfo}">
                <td th:text="${item[2]}"></td> <!-- Medicine name -->
                <td>
                <a th:text="${item[3]}" th:href="@{/home/edit_inventory/{id}(id=${item[1]})}"></a>
                </td> <!-- Inventory name -->
                <td th:text="${#dates.format(item[0], 'dd/MM/yyyy')}"></td><!-- Date expiry -->
                <td th:text="${item[4]}"></td> <!-- Treatment -->
            </tr>
        </tbody>
    </table>

    <!-- Redirect button -->
    <div style="position: fixed; bottom: 20px; right: 20px;">
        <a th:href="@{/home}" class="btn btn-primary">Ir a Home</a>
    </div>
</body>
</html>
```

MEDICINE

En esta página se muestra un listado con todos los medicamentos que se tiene almacenados en base de datos. Dicho inventario es el almacén de medicamentos que posteriormente se pueden añadir en los inventarios del hogar. Las funcionalidades principales son:

- ***Botón "Nuevo medicamento":*** Abre una nueva página que permite añadir un nuevo medicamento a la base de datos.
- ***Botón "Editar":*** Redirecciona a una nueva página para editar el medicamento donde se ha pulsado el botón.
- ***Botón "Eliminar":*** Realiza una petición al controlador para eliminar el medicamento de la base de datos.
  Si el medicamento que se intenta eliminar se encuentra guardado en alguno de los inventarios del hogar muestra un mensaje de error.
- ***Botón "Home":*** Al pulsarlo vuelve a la página inicial "home".

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
        <meta charset="UTF-8">
        <title>Lista de medicamentos</title>
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet">
        <style>
  .edit-btn {
    color: white;
    background-color: #FFC107;
    border-color: #FFC107;
  }

  .delete-btn {
    color: white;
    background-color: #DC3545;
    border-color: #DC3545;
  }
</style>
</head>
<body>

<div class="container">
        <h1>Lista de medicamentos</h1>

        <!-- Muestra un mensaje de error si el medicamento está asignado a un inventario-->
        <div th:if="${errorMessage != null}" class="alert alert-danger">
            <p th:text="${errorMessage}"></p>
        </div>

        <a th:href="@{/home/create_medicine}" class="btn btn-primary">Nuevo medicamento</a>

        <table class="table">
                <thead>
                        <tr>
                                <th style="display:none;">Id</th>
                                <th scope="col">Nombre</th>
                                <th scope="col">Formato</th>
                                <th scope="col">Composición</th>
                                <th scope="col">Tratamiento</th>
                                <th colspan="2" scope="col">Acciones</th>
                        </tr>
                </thead>
                        <tbody>
                                <tr th:each="medicamento, rowStat : ${medicine}" th:class="${rowStat.even} ?
'table-light'">
                                        <td style="display:none;" th:text="${medicamento.id}"></td>
                                        <td th:text="${medicamento.name}"></td>
                                        <td th:text="${medicamento.format}"></td>
                                        <td th:text="${medicamento.composition}"></td>
                                        <td th:text="${medicamento.treatment}"></td>
                                        <td><a th:href="@{/home/edit_medicine/{id} (id=${medicamento.id})}"
class="btn btn-warning edit-btn">Editar</a></td>
                                        <td><a th:href="@{/home/delete_medicine/{id}
(id=${medicamento.id})}" class="btn btn-danger delete-btn">Eliminar</a></td>
                                </tr>
                        </tbody>
        </table>
        <!-- Redirect button -->
        <div style="position: fixed; bottom: 20px; right: 20px;">
                <a th:href="@{/home}" class="btn btn-primary">Ir a Home</a>
        </div>
</div>
</body>
</html>
```

CREATE MEDICINE

Página que permite añadir un nuevo medicamento. En el caso de que dicho medicamento ya esté guardado, es decir, que tenga el mismo nombre y formato, aparece un mensaje de error.

Los campos que se deben rellenar son los siguientes:

- ✓ Nombre del medicamento (campo obligatorio).
- ✓ Formato (campo obligatorio): valores predefinidos (pomada, líquido, pastillas, sobres)
- ✓ Composición.
- ✓ Tratamiento (campo obligatorio). Forma que debe tomarse según prospecto o tratamiento médico.

Para guardar los datos se debe clickar el botón de "***Guardar***". En caso de no querer continuar, se pulsa el botón "***Volver***" que redirecciona al listado de la página de "medicine".

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Nuevo medicamento</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
    rel="stylesheet"
    integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
    crossorigin="anonymous">
</head>
<body>

    <div class="container">
        <h1> Añadir medicamento  </h1>

         <!-- Display an error message if that medication already exists-->
                  <div th:if="${messageError}" class="alert alert-danger">
                  <p th:text="${messageError}"></p>
                  </div>

        <form th:action="@{/home/save_medicine}" method="post">
            <div class="mb-3">
                <label for="name" class="form-label">Nombre medicamento*</label>
                <input type="text" class="form-control" id="name" name="name" required
                placeholder="Escriba el nombre del medicamento ">
            </div>
            <div class="mb-3">
                <label for="format" class="form-label">Formato*</label>
                <select class="form-select" id="format" name="format" required>
                           <option th:each="format : ${T(com.invpharma.entity.Medicine.MedicineFormat).values()}"
                                    th:value="${format}" th:text="${format.name() + (format ==
T(com.invpharma.entity.Medicine.MedicineFormat).Pomada ? ' (mg)' : (format ==
T(com.invpharma.entity.Medicine.MedicineFormat).Liquido ? ' (ml)' : ''))}"></option>
                           </select>
            </div>
            <div class="mb-3">
                <label for="composition" class="form-label">Composición</label>
                <input type="text" class="form-control" id="composition" name="composition"
                placeholder="Indique los compuestos que lo forman">
            </div>
            <div class="mb-3">
                <label for="treatment" class="form-label">Tratamiento*</label>
                <input type="text" class="form-control" id="treatment" name="treatment" required
                placeholder="Modo de administrar según el médico">
            </div>
             <button type="submit" name="createSubmit" class="btn btn-success">Guardar</button>
        </form>

        <!-- Redirect button -->
        <div style="position: fixed; bottom: 20px; right: 20px;">
            <a th:href="@{/home/medicine}" class="btn btn-primary">Volver</a>
        </div>
    </div>
</body>
</html>
```

EDIT MEDICINE

Página web que se abre para editar los datos del medicamento, siendo todos los campos modificables (Nombre, Formato, Composición y Tratamiento).

Los botones de la página web son:

- **Botón *"Actualizar".*** Al pulsarlo envía los datos al controlador para actualizar la base datos, previa validación que estén los campos obligatorios.
- **Botón *"Volver".*** Vuelve al listado de medicamentos.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Editar medicamento</title>
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet"
integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">
</head>
<body>
<div class="container">
                  <h1> Actualizar productos </h1>

         <form th:action="@{/home/save_medicine}" method = "post">
         <input type="hidden" name="id" th:value="${medicamento.id}">
                  <div class="mb-3">
                    <label for="medicamento" class="form-label">Nombre medicamento</label>
                    <input type="text" class="form-control" id="medicamento" name="name"
                    th:value="${medicamento.name}" required>
                  </div>
                  <div class="mb-3">
         <label for="format" class="form-label">Formato*</label>
         <select class="form-select" id="format" name="format" required>
                  <option th:each="formatOption : ${T(com.invpharma.entity.Medicine.MedicineFormat).values()}"
                                    th:value="${formatOption}"
                                    th:text="${formatOption.name() + (formatOption == 'Pomada' ? ' (mg)' :
(formatOption == 'Liquido' ? ' (ml)' : ''))}"
                                    th:selected="${formatOption == medicamento.format.name()}"></option>
                  </select>
        </div>
                  <div class="mb-3">
                    <label for="composition" class="form-label">Composición</label>
                    <input type="text" class="form-control" id="composition" name="composition"
                    th:value="${medicamento.composition}">
                  </div>
                  <div class="mb-3">
                    <label for="tratamiento" class="form-label">Tratamiento</label>
                    <input type="text" class="form-control" id="tratamiento" name="treatment"
                    th:value="${medicamento.treatment}" required>
                  </div>

                  <button type= "submit" class="btn btn-success"> Actualizar</button>
         </form>
</div>
<!-- Redirect button -->
<div style="position: fixed; bottom: 20px; right: 20px;">
         <a th:href="@{/home/medicine}" class="btn btn-primary">Volver</a>
</div>

</body>
</html>
```

INVENTORY

Página que muestra el listado de los inventarios existentes, es decir, aquellos lugares donde se puede tener medicamentos guardados.

Funcionalidades:

- ***Botón "Nuevo inventario".*** Redirecciona a una nueva página para poder añadir un inventario.

- ***Botón "Editar".*** Abre la página con el listado de los medicamentos que tiene ese inventario.
- ***Botón "Eliminar".*** Realiza una petición al controlador para eliminar el medicamento de la base de datos.
  Se eliminan todos los medicamentos que se encuentren asociados a ese inventario.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Listado Inventario</title>
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet"
integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">
</head>
<body>

        <div class="container">
                <h1>Lista de inventarios</h1>

                <a th:href="@{/home/create_inventory}" class="btn btn-primary float-left"> Nuevo inventario</a>

                <table class="table">
                  <thead>
                    <tr>
                      <th style="visibility:collapse; display:none;" scope="col">Id</th>
                      <th scope="col">Nombre</th>
                      <th scope="col">Fecha creación</th>
                      <th scope="col">Fecha modificación</th>
                      <th scope="col">Acción</th>
                      <th scope="col">Acción</th>
                    </tr>
                  </thead>
                  <tbody>
                    <tr th:each="inventarios:${inventory}">
                      <td style="visibility:collapse; display:none;" scope="row"
th:text="${inventarios.id}"></td>
                       <td th:text="${inventarios.name}"></td>
                            <td th:text="${#dates.format(inventarios.date_create, 'dd/MM/yyyy')}"></td>
                            <td th:text="${#dates.format(inventarios.date_edit, 'dd/MM/yyyy')}"></td>
                            <td><a th:href="@{/home/edit_inventory/{id} (id=${inventarios.id})}" class="btn btn-
warning"> Editar</a></td>
                            <td><a th:href="@{/home/delete_inventory/{id} (id=${inventarios.id})}" class="btn
btn-danger"> Eliminar</a></td>
                    </tr>

                        <!-- Redirect button -->
                        <div style="position: fixed; bottom: 20px; right: 20px;">
                                <a th:href="@{/home}" class="btn btn-primary">Ir a Home</a>
                  </tbody>
                </table>
        </div>
</body>
</html>
```

CREATE INVENTORY

Template que crea un nuevo inventario de aquel lugar donde se pueden tener medicamentos.

Tiene los siguientes campos:

- ✓ Nombre inventario. Nombre del lugar donde se va a crear el nuevo inventario, por ejemplo, la casa de la playa.
- ✓ Fecha de creación. Se rellena automáticamente con la fecha actual, pudiendo modificarse por la fecha que se desee.

Los botones a utilizar son:

- **Botón *"Guardar".*** Guarda el nuevo inventario, comprobando que ambos campos se encuentren rellenos.
- **Botón *"Volver".*** Vuelve al listado de inventarios.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
	<meta charset="UTF-8">
	<title>New Inventory</title>
	<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
	<link rel="stylesheet" href="//code.jquery.com/ui/1.13.2/themes/base/jquery-ui.css">
	<link rel="stylesheet" href="/resources/demos/style.css">
</head>

<body>


	<div class="container">
		<h1> Crear inventario </h1>

	<form th:action="@{/home/save_inventory}" th:object="${inventory}" method = "post">
		<div class="mb-3">
		  <label for="Inventory_name" class="form-label">Nombre inventario</label>
		  <input type="text" class="form-control" id="Inventory_name" name="name" required
		  placeholder="Escriba el nombre del inventario">
		</div>
		<div class="mb-3">
			<label for="date_create" class="form-label">Fecha creación</label>
			<input type="date" class="form-control" id="date_create" name="date_create"
th:field="*{date_create}"
			th:attr="placeholder=${#dates.format(inventory.date_create, 'yyyy-MM-dd')}" required>
		</div>
		<button type= "submit" class="btn btn-success"> Guardar</button>
	</form>
	<!-- Redirect button -->
	<div style="position: fixed; bottom: 20px; right: 20px;">
	<a th:href="@{/home/inventory}" class="btn btn-primary">Volver</a>
	</div>
</div>
</body>
</html>
```

EDIT INVENTORY

Página web donde se muestra un listado con todos los medicamentos almacenados en ese inventario y que dispone de las siguientes funcionalidades.

- **Botón *"Añadir medicamento".*** Redirecciona a una nueva página web para rellenar un formulario con los datos de un nuevo medicamento que se añade al inventario que se desee.
- **Botón *"Editar".*** Abre una nueva página web con los datos del medicamento, los cuales se editan para añadir, quitar cantidad del mismo o modificar su fecha de compra o caducidad.
- **Botón "*Eliminar".*** Elimina el medicamento desde donde se pulsa el botón de ese inventario.
- **Botón "*Volver*".** Vuelve a la página anterior, es decir, al listado de inventarios.

Una de las características especiales del listado, es el formato en color rojo del texto que aparece en las celdas de fecha de caducidad para aquellos medicamentos del inventario que estén caducados o próximos a caducarse o de las celdas de cantidad que estén cerca de agotarse o agotadas, es decir, cuya cantidad en pastillas o sobres esté por debajo de 5, en pomada sea inferior a 10 mg y de cualquier líquido que baje de 10 ml, siendo estos los valores por defecto.

```
!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Editar inventario</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet">
    <style>
        /* Estilos personalizados */
        .edit-btn {
            color: white;
            background-color: #FFC107;
            border-color: #FFC107;
        }

        .delete-btn {
            color: white;
            background-color: #DC3545;
            border-color: #DC3545;
        }

        /* Estilos personalizados para la tabla */
        .highlight-row {
            background-color: #F2F2F2;
        }
    </style>
</head>
<body>

<div class="container">
    <h1>Lista de medicamentos</h1>

    <a th:href="@{/home/create_inventoryLine}" class="btn btn-primary"> Añadir medicamento</a>

    <table class="table">
        <thead>
        <tr>
            <th style="display:none;">Id</th>
            <th style="display:none;">Inventario_id</th>
            <th scope="col">Medicamento</th>
            <th scope="col">Cantidad</th>
            <th scope="col">Fecha compra</th>
            <th scope="col">Fecha caducidad</th>
            <th colspan="2" scope="col">Acciones</th>
        </tr>
        </thead>
<tbody>
        <tr th:each="lineaInv, rowStat : ${listinv}" th:class="${rowStat.even} ? 'highlight-row'">
            <td style="display:none;" th:text="${lineaInv.id}"></td>
            <td style="display:none;" th:text="${lineaInv.id_inventory}"></td>
            <td th:text="${lineaInv.medicine.name}"></td>
            <td th:text="${lineaInv.quantity}" th:style="${(lineaInv.medicine.format.toString() == 'Pastillas' or
            lineaInv.medicine.format.toString() == 'Sobres') and (lineaInv.quantity < 5) or
            (lineaInv.medicine.format.toString() == 'Pomada') and (lineaInv.quantity < 10) or
            (lineaInv.medicine.format.toString() == 'Liquido') and (lineaInv.quantity < 50)} ? 'color: red;' :
''"></td>
            <td th:text="${#dates.format(lineaInv.date_buy, 'dd/MM/yyyy')}"></td>
            <td th:text="${#dates.format(lineaInv.date_expiry, 'dd/MM/yyyy')}" th:id="dateExpiryField">
            </td>
            <td><a th:href="@{/home/inventoryLine/{id} (id=${lineaInv.id})}" class="btn btn-warning edit-btn">
Editar</a></td>
            <td><a th:href="@{/home/delete_inventoryLine/{id} (id=${lineaInv.id})}" class="btn btn-danger delete-
btn"> Eliminar</a></td>
        </tr>
        </tbody>
    </table>

    <!-- Redirect button -->
    <div style="position: fixed; bottom: 20px; right: 20px;">
        <a th:href="@{/home/inventory}" class="btn btn-primary">Volver</a>
    </div>
</div>
```

```
<script>
    function changeColorForExpiryDate() {
        var dateExpiryFields = document.querySelectorAll("#dateExpiryField");

        dateExpiryFields.forEach(function(dateExpiryField) {
            var dateParts = dateExpiryField.textContent.trim().split('/');
            var formattedDate = dateParts[1] + '/' + dateParts[0] + '/' + dateParts[2]; // Change to MM/dd/yyyy
            var expiryDate = new Date(formattedDate);
            var currentDate = new Date();

            // Set the time of currentDate to 00:00:00 to avoid time zone differences.
            currentDate.setHours(0, 0, 0, 0);

            // Set the period of days for expiration warning (7 days)
            var thresholdDays = 7;

            // Calculate the difference in days
            var daysDifference = Math.floor((expiryDate - currentDate) / (24 * 60 * 60 * 1000));

            if (daysDifference <= thresholdDays) {
                dateExpiryField.style.color = "red"; // Change to red if it's close to expiration or expired
            } else {
                dateExpiryField.style.color = ""; // Default color
            }
        });
    }

    // Reload page
    window.onload = changeColorForExpiryDate;
</script>

</body>
</html>
```

CREATE INVENTORY LINE

Página web que muestra un formulario para añadir un nuevo medicamento. Los campos a rellenar son:

- ✓ Inventario: Indica el inventario donde se añade el medicamento. Es un "comboBox" cuyos datos son los inventarios existentes.
- ✓ Medicamento. Se trata de un "comboBox" con todos los medicamentos que se tienen almacenados en la base de datos.
- ✓ Fecha de compra. Se rellenar con el día de la compra del medicamento.
- ✓ Fecha de caducidad. Se debe rellenar con el día que indica el medicamento que caduca.
- ✓ Cantidad. Cantidad que dispone el fármaco, diferenciando entre ml, sobres, pastillas y pomada.

Respecto a los botones de la página, su funcionalidad es la siguiente:

- **Botón "*Guardar".*** Antes de enviar la orden al controlador para que guarde los datos realiza las siguientes validaciones:
    - Comprueba que todos los campos estén rellenos.
    - La fecha de caducidad sea posterior a la fecha de compra.
    - En el caso que el controlador compruebe que ya existe el medicamento en ese inventario, muestra un mensaje de error, impidiendo que se almacene.
- **Botón "*Volver".*** Vuelve a la página anterior, es decir, al listado de medicamentos del inventario que se está editando.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
	<meta charset="UTF-8">
	<title>AddMedicine</title>
	<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
	<link rel="stylesheet" href="//code.jquery.com/ui/1.13.2/themes/base/jquery-ui.css">
	<link rel="stylesheet" href="/resources/demos/style.css">
</head>

<body>

<div class="container">
	<h2>Añadir nuevo medicamento</h2>

	<!-- Display an error message if the medicine already exists in the inventory-->
	<div th:if="${messageError}" class="alert alert-danger">
		<p th:text="${messageError}"></p>
	</div>
	<!-- Message success-->
	<div th:if="${message}" class="alert alert-success">
		<p th:text="${message}"></p>
	</div>

<!--	Inventory Medication Addition Form -->
	<form th:action="@{/home/save_inventoryLine}" method="post">
		<div class="mb-3">
			<label for="idInventory" class="form-label">Inventario</label>
			<select class="form-control" id="idInventory" name="id_inventory" required>
				<option th:each="inventarios:${inventoryAux}"
					th:value="${inventarios.id}"
					th:text="${inventarios.name}"
					th:selected="${inventarios.id == id}"></option>
			</select>
		</div>
		<div class="mb-3">
			<label for="idMedicine" class="form-label">Medicamento</label>
			<select class="form-control" id="idMedicine" name="id_medicine" required>
				<option th:each="medicina:${medicineAux}"  th:value="${medicina.id}"
				th:text="${medicina.name}"></option>
			</select>
		</div>
		<div class="mb-3">
			<label for="date_buy" class="form-label">Fecha de compra</label>
			<input type="date" class="form-control" id="date_buy" name="date_buy"
			th:field="*{listinv.date_buy}"
			th:attr="placeholder=${#dates.format(listinv.date_buy, 'yyyy-MM-dd')}"  required>
		</div>
		<div class="mb-3">
			<label for="quantity" class="form-label">Cantidad</label>
			<input type="number" class="form-control" id="quantity" name="quantity" required>
		</div>
		<button type="submit" name="createSubmit" class="btn btn-success">Guardar</button>
	</form>
</div>
<!-- Redirect button -->
<div style="position: fixed; bottom: 20px; right: 20px;">
	<a th:href="@{/home/edit_inventory/{id} (id=${id})}" class="btn btn-primary">Volver</a>
</div>

<!-- Script that checks if the date_expiry is later than the date_buy -->
<script th:inline="javascript">
    document.addEventListener("DOMContentLoaded", function () {
        var form = document.querySelector("form");
        form.addEventListener("submit", function (event) {
            var dateBuy = new Date(document.querySelector("#date_buy").value);
            var dateExpiry = new Date(document.querySelector("#date_expiry").value);

            if (dateExpiry <= dateBuy) {
                var expiryError = document.querySelector("#expiry-error");
                expiryError.textContent = "La fecha de caducidad debe ser posterior a la fecha de compra";
                expiryError.style.display = "block";
                event.preventDefault(); // Prevent the form from submitting if validation fails.
            }
        });
    });
</script>
</body>
</html>
```

INVENTORY LINE

Formulario relleno con los datos del medicamento que se quiere editar. Los campos que se pueden modificar son las fechas de compra y caducidad y la cantidad.

Como particularidad de este template, es que rellena el campo de color rojo a los medicamentos caducados o próximos a caducarse y cuya cantidad baje por debajo de las 5 unidades en sobres o pastillas, 10 mg en pomada y 50 ml en líquidos. Además, si se actualiza la fecha de caducidad dentro de los 7 días posteriores a la fecha actual, muestra un mensaje de advertencia y si dicha fecha se intenta actualizar a una inferior a la de compra, muestra un pop-up con un mensaje de error, sin actualizar los datos.

Los botones que tiene está web son:

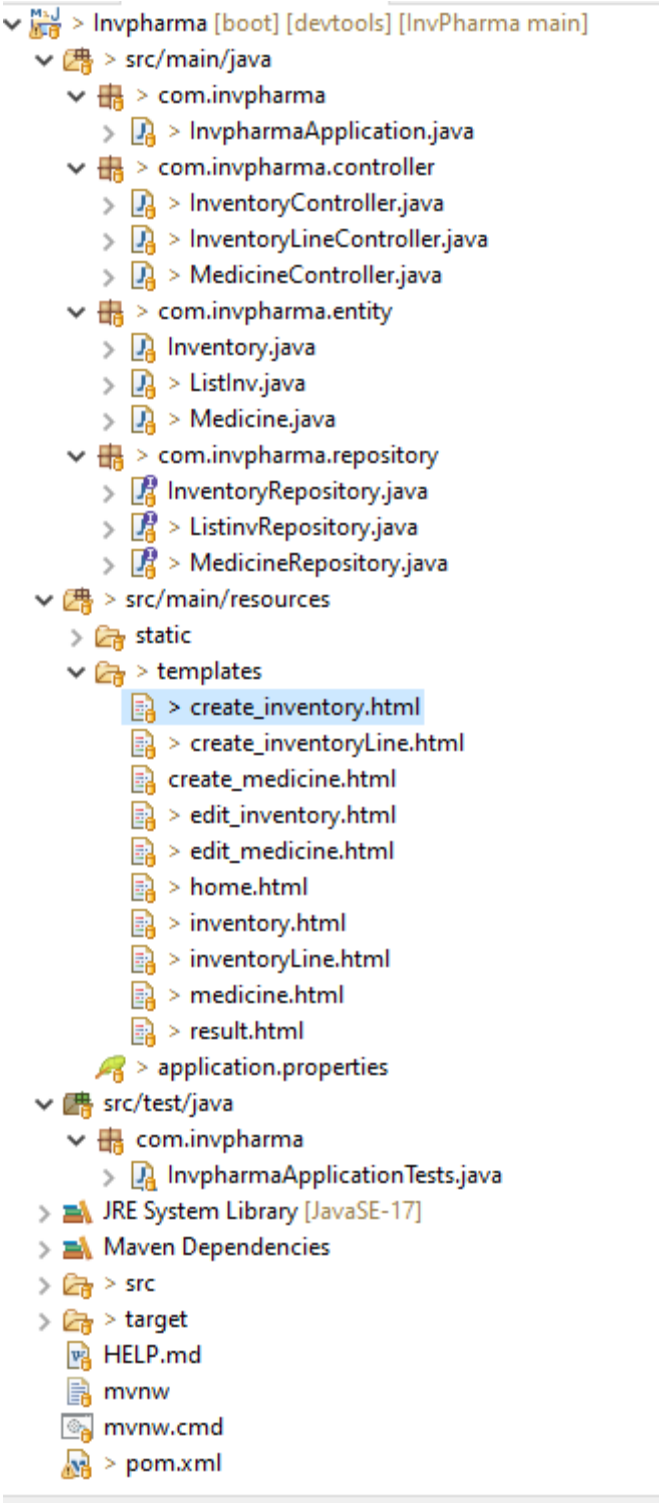
- **Botón "*Actualizar*".** Una vez pulsado envía los datos al controlador para que se almacene en base de datos, realizando las comprobaciones anteriores previamente.
- **Botón "*Volver*".** Vuelve a la página anterior, es decir, al listado de medicamentos del inventario que se está editando.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Editar línea inventario</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
</head>
<body>
<div class="container">
    <h2>Editar línea Inventario</h2>
    <form th:action="@{/home/save_inventoryLine}" method="post" onsubmit="return validateForm();">
        <input type="hidden" name="id" th:value="${inventoryLine.id}">
        <input type="hidden" name="id_medicine" th:value="${inventoryLine.id_medicine}">
        <input type="hidden" name="id_inventory" th:value="${inventoryLine.id_inventory}">

        <div class="mb-3">
            <label for="Inventary" class="form-label">Inventario</label>
            <input type="text" class="form-control" id="Inventary"
                   th:value="${inventoryAux.name}" readonly>
        </div>
        <div class="mb-3">
            <label for "Medicine" class="form-label">Medicamento</label>
            <input type="text" class="form-control" id="Medicine"
                   th:value="${medicineAux.name}" readonly>
        </div>
        <div class="mb-3">
            <label for="dateBuy" class="form-label">Fecha compra</label>
            <input type="date" class="form-control" id="dateBuy" name="date_buy"
                   th:value="${#dates.format(inventoryLine.date_buy,'yyyy-MM-dd')}">
        </div>
        <div class="mb-3">
            <label for="dateExpiry" class="form-label">Fecha caducidad</label>
            <input type="date" class="form-control" id="dateExpiry" name="date_expiry"
                   th:value="${#dates.format(inventoryLine.date_expiry,'yyyy-MM-dd')}"
                   th:id="dateExpiryField">
            <div id="expiry-error" class="text-danger"></div>
        </div>
```

```
        <div class="mb-3">
            <label for="quantity" class="form-label">Cantidad</label>
            <input type="number" class="form-control" id="quantity" name="quantity"
                   th:value="${inventoryLine.quantity}"
                   th:style="${(medicineAux.format.toString() == 'Pastillas' or medicineAux.format.toString() ==
'Sobres') and (inventoryLine.quantity < 5)
                   or (medicineAux.format.toString() == 'Pomada') and (inventoryLine.quantity < 10)
                   or (medicineAux.format.toString() == 'Liquido') and (inventoryLine.quantity < 50)} ?
'background-color: red;' : ''">
        </div>
        <button type="submit" class="btn btn-success">Actualizar</button>
    </form>
</div>

<!-- Redirect button -->
<div style="position: fixed; bottom: 20px; right: 20px;">
    <a th:href="@{/home/edit_inventory/{id}(id=${inventoryLine.id_inventory})}" class="btn btn-primary">Volver</a>
</div>
<script>

function changeColorForExpiryDate() {
        var dateExpiryField = document.getElementById("dateExpiryField");
        var expiryDate = new Date(dateExpiryField.value);
        var currentDate = new Date();

        // Set the time of currentDate to 00:00:00 to avoid time zone differences.
        currentDate.setHours(0, 0, 0, 0);

        // Set the period of days for expiration warning (7 days)
        var thresholdDays = 7;

        // Calculate the difference in days
        var daysDifference = Math.floor((expiryDate - currentDate) / (24 * 60 * 60 * 1000));

        // Check if expiry date is valid
         if (expiryDate <= currentDate) {
            dateExpiryField.style.backgroundColor = "red"; // Change to red if it's expired
```

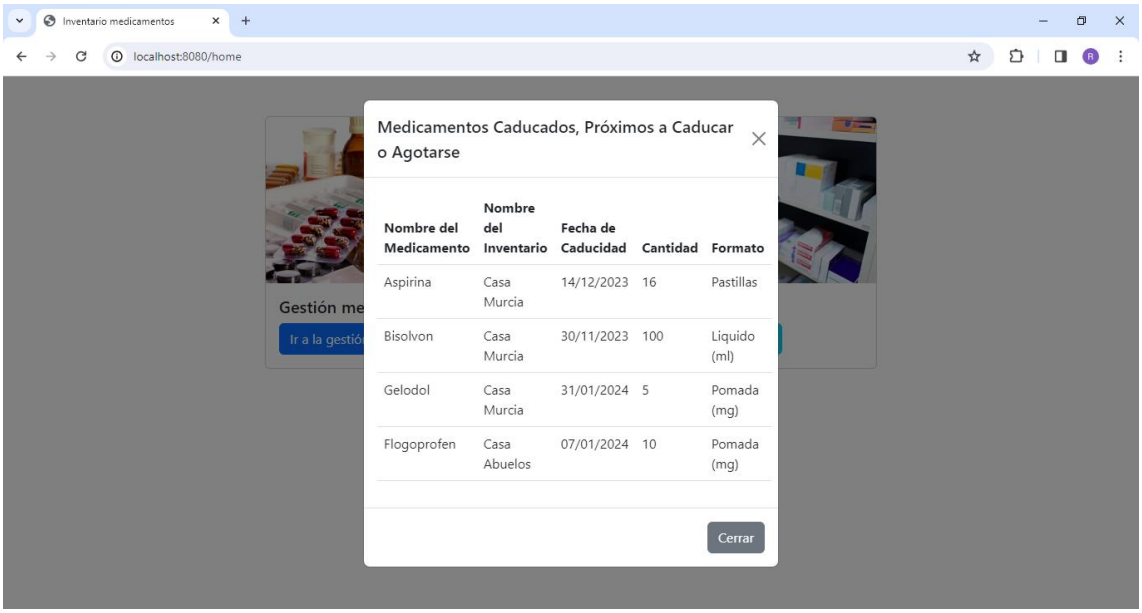
Estructura del proyecto

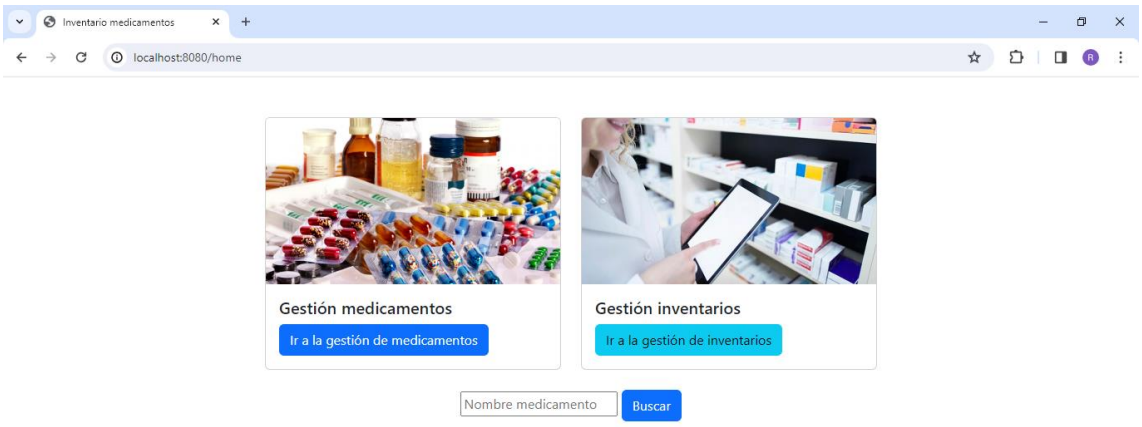
La estructura final queda de la siguiente manera una vez que se han explicado los paquetes que la forman.

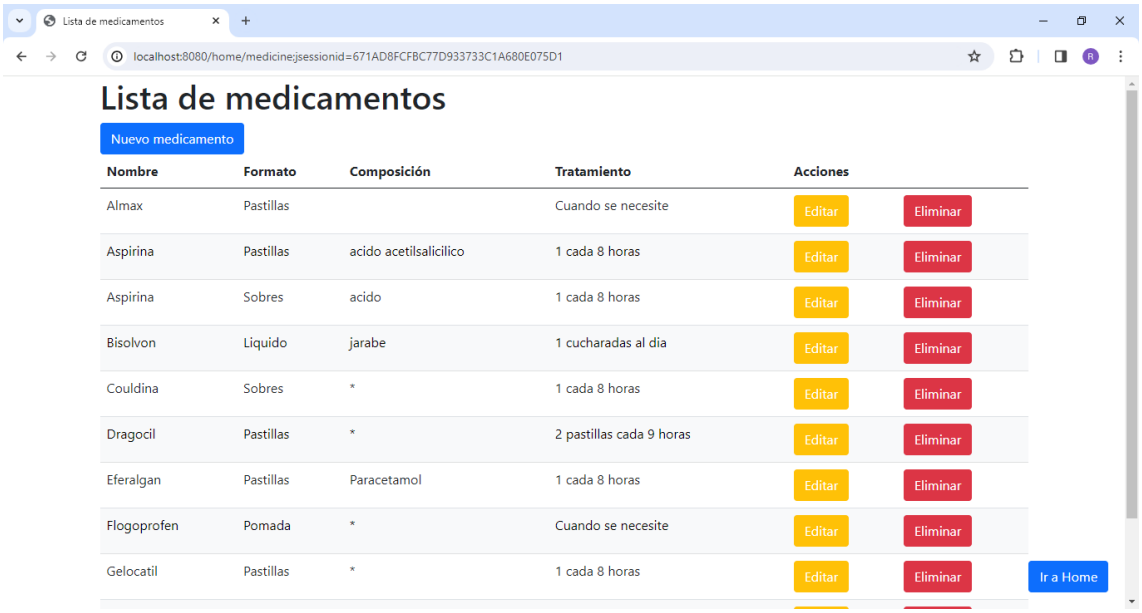

Invpharma [boot] [devtools] [InvPharma main]
src/main/java
com.invpharma
InvpharmaApplication.java
com.invpharma.controller
InventoryController.java
InventoryLineController.java
MedicineController.java
com.invpharma.entity
Inventory.java
ListInv.java
Medicine.java
com.invpharma.repository
InventoryRepository.java
ListinvRepository.java
MedicineRepository.java
src/main/resources
static
templates
create_inventory.html
create_inventoryLine.html
create_medicine.html
edit_inventory.html
edit_medicine.html
home.html
inventory.html
inventoryLine.html
medicine.html
result.html
application.properties
src/test/java
com.invpharma
InvpharmaApplicationTests.java
JRE System Library [JavaSE-17]
Maven Dependencies
src
target
HELP.md
mvnw
mvnw.cmd
pom.xml

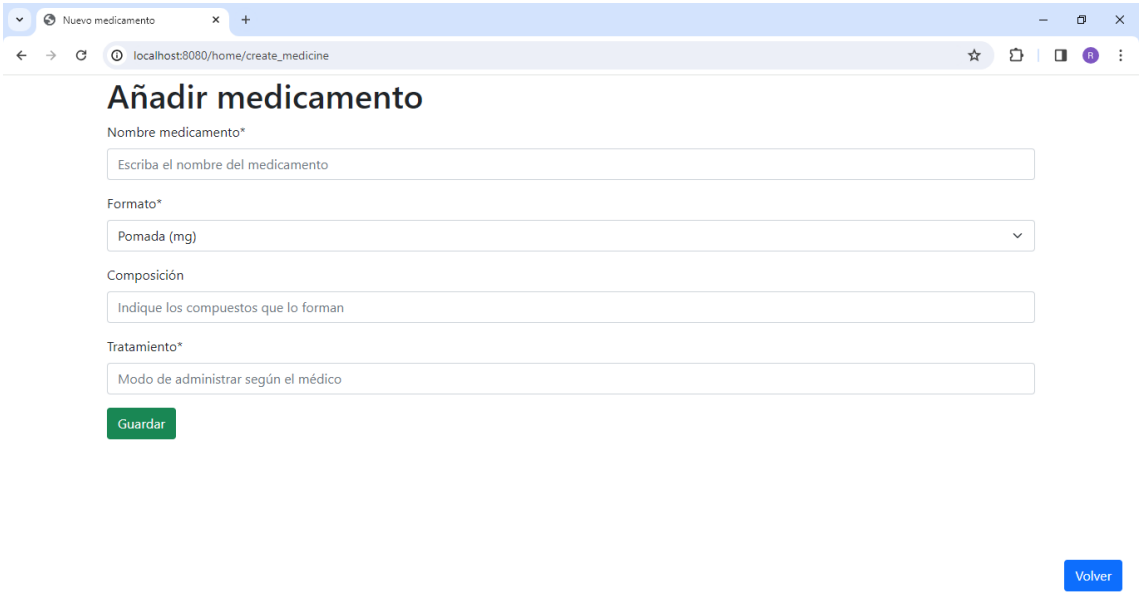
Anexos

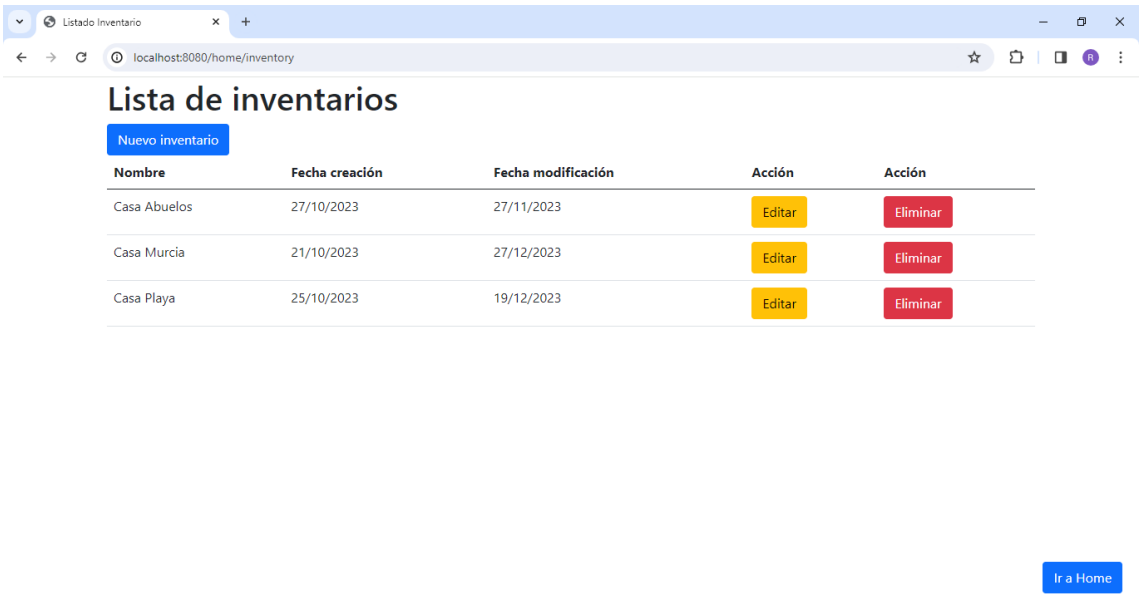
Capturas de pantalla



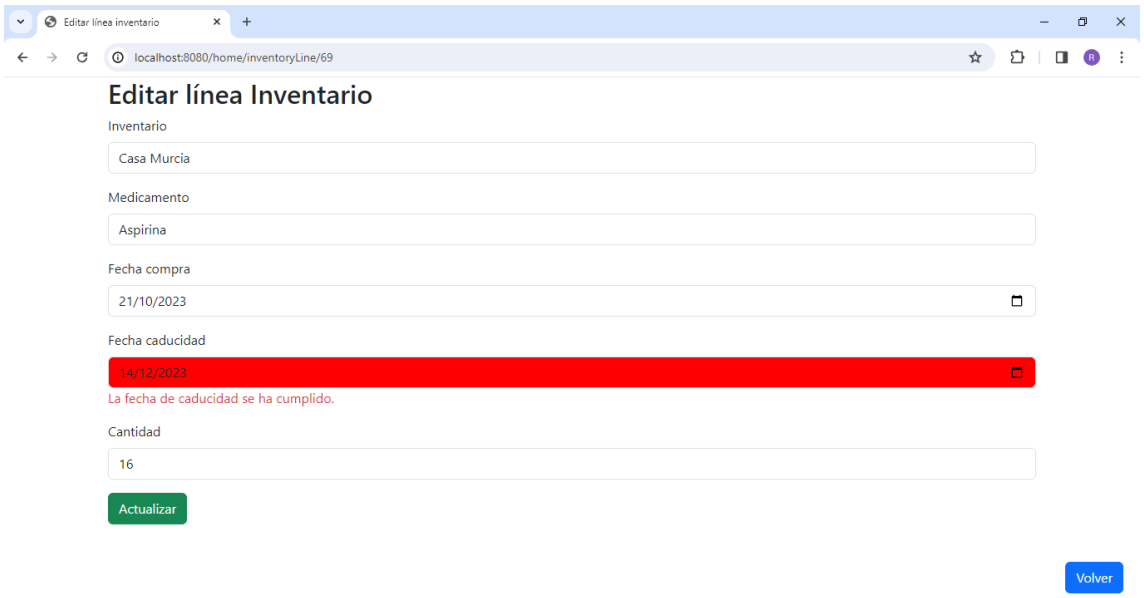

Inventario medicamentos
localhost:8080/home
Gestión medicamentos
Ir a la gestión de medicamentos
Gestión inventarios
Ir a la gestión de inventarios
Nombre medicamento
Buscar

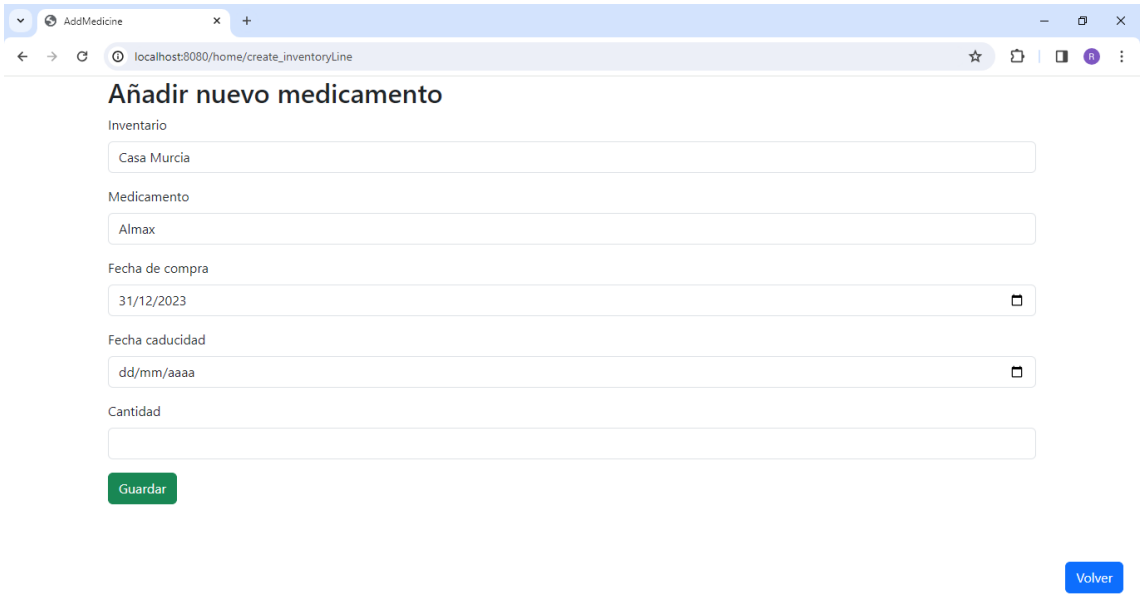

Lista de medicamentos
localhost:8080/home/medicine;jsessionid=671AD8FCFBC77D933733C1A680E075D1
Lista de medicamentos
Nuevo medicamento
Nombre
Formato
Composición
Tratamiento
Acciones
Almax
Pastillas
Cuando se necesite
Aspirina
Pastillas
acido acetilsalicilico
1 cada 8 horas
Aspirina
Sobres
acido
1 cada 8 horas
Bisolvon
Liquido
jarabe
1 cucharadas al dia
Couldina
Sobres
*
1 cada 8 horas
Dragocil
Pastillas
*
2 pastillas cada 9 horas
Eferalgan
Pastillas
Paracetamol
1 cada 8 horas
Flogoprofen
Pomada
*
Cuando se necesite
Gelocatil
Pastillas
*
1 cada 8 horas
Editar
Eliminar
Ir a Home


Nuevo medicamento
localhost:8080/home/create_medicine
Añadir medicamento
Nombre medicamento*
Escriba el nombre del medicamento
Formato*
Pomada (mg)
Composición
Indique los compuestos que lo forman
Tratamiento*
Modo de administrar según el médico
Guardar
Volver


Listado Inventario
localhost:8080/home/inventory
Lista de inventarios
Nuevo inventario
Nombre
Fecha creación
Fecha modificación
Acción
Acción
Casa Abuelos
27/10/2023
27/11/2023
Editar
Eliminar
Casa Murcia
21/10/2023
27/12/2023
Editar
Eliminar
Casa Playa
25/10/2023
19/12/2023
Editar
Eliminar
Ir a Home


Editar inventario
localhost:8080/home/edit_inventory/16
Lista de medicamentos
Añadir medicamento
Medicamento
Cantidad
Fecha compra
Fecha caducidad
Acciones
Aspirina
16
21/10/2023
14/12/2023
Editar
Eliminar
Bisolvon
100
21/10/2023
30/11/2023
Editar
Eliminar
Flogoprofen
20
21/10/2023
18/06/2025
Editar
Eliminar
Gelocatil
30
25/10/2023
25/10/2024
Editar
Eliminar
Gelodol
5
13/12/2023
31/01/2024
Editar
Eliminar
Volver





Bibliografía

[1] https://spring.io/

[2] https://es.wikipedia.org/wiki/Spring_Framework

[3] https://javadesde0.com/diferencias-entre-spring-y-spring-boot/

[4] https://openwebinars.net/blog/que-es-thymeleaf/

[5] https://blog.softtek.com/es/thymeleaf

[6] https://getbootstrap.com/

[7] https://raiolanetworks.es/blog/bootstrap/

[8] https://openwebinars.net/blog/que-es-mysql/

[9] https://blog.hubspot.es/website/que-es-mysql

[10] https://www.arsys.es/blog/mysql#Caracteristicas_y_ventajas_de_MySQL

[11] https://creatitproject.eu/curso/mock-up/lessons/lesson-1-what-is-a-mock-up-what-is-for/

[12] https://contributor.freepik.com/blog/es/disfruta-ventajas-disenar-mockups/

[13] https://ideandoazul.com/branding/mockup/