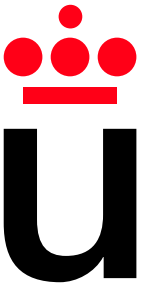


Universidad
Rey Juan Carlos

Escuela Técnica Superior
de Ingeniería Informática

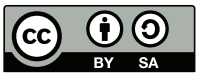
Grado en Ingeniería Informática

Curso 2024-2025

Trabajo Fin de Grado

HEALTHCARER: UNA APLICACIÓN WEB ORIENTADA A LA GESTIÓN Y SEGUIMIENTO DE LOS TRATAMIENTOS MÉDICOS

Autor: Eva Ruiz Aguado
Tutor: Michel Maes Bermejo



Resumen

La aplicación web diseñada para la gestión y el seguimiento de tratamientos médicos surge de la necesidad de mejorar la adherencia a los tratamientos farmacológicos. La aplicación incorpora y fusiona funcionalidades de plataformas líderes como MyTherapy y Medisafe, destacando su sistema de notificaciones y la claridad del historial de tomas.

La arquitectura de la aplicación se basa en un modelo de tres capas (frontend, backend y base de datos) y sigue el patrón de diseño Modelo-Vista-Controlador (MVC). En el backend, se utilizan Java y Spring Boot, con Spring Security para la seguridad y JPA/Spring Data para la persistencia. El frontend está construido con HTML, CSS, JavaScript y Mustache. En cuanto a las bases de datos, se emplea MySQL en producción y H2 para el desarrollo. El proyecto implementa pruebas unitarias, de integración y End-to-End (E2E) con JUnit, Mockito y Selenium. El desarrollo sigue la metodología de Trunk Based Development para facilitar la integración y entrega continuas, automatizadas con GitHub Actions. Finalmente, el despliegue se realiza mediante contenedores Docker y utilizando Docker Compose.

El trabajo detalla las distintas fases del proyecto, comenzando por la motivación inicial y la definición de los objetivos, pasando por el análisis de tecnologías y metodologías, y el diseño e implementación de la aplicación. También se incluyen las pruebas y su automatización, el empaquetado con Docker, y el despliegue final. El documento concluye con un análisis de los resultados y propuestas para futuros trabajos.

Palabras clave:

- Aplicación web
- Java y Spring Boot
- Base de datos MySQL
- Docker y Docker Compose
- Pruebas de software
- Integración continua (CI/CD)
- Trunk Based Development

Índice de contenidos

Índice de figuras

Índice de códigos

1 Introducción

Hoy en día, muchas personas sufren de enfermedades, usualmente crónicas, que requieren seguir un tratamiento médico estricto y continuado. Esto implica que deben tomar medicamentos en dosis específicas y en momentos concretos del día para no empeorar su estado de salud. No obstante, la adherencia a los tratamientos farmacológicos es un problema bastante común en la población actual.

La adherencia al tratamiento, según la OMS, se define como el grado en que la conducta de un paciente, en relación con la toma de medicación, el seguimiento de una dieta o la modificación de hábitos de vida, se corresponde con las recomendaciones acordadas con el profesional sanitario. Esto implica que la adherencia no solo se refiere a la toma de medicamentos, sino también a seguir las indicaciones médicas en general para mejorar la salud.

Según la Organización Mundial de la Salud (OMS), la adherencia al tratamiento es, en esencia, el compromiso de un paciente con el seguimiento de las indicaciones médicas recibidas por su especialista. Actualmente, es un problema de salud pública muy grave que afecta a todos los países y a todas las enfermedades. Se estima que la adherencia a los tratamientos crónicos en los países desarrollados es de, aproximadamente, el 50 % [1]. Esto significa que la mitad de los pacientes no siguen correctamente sus tratamientos, lo que puede acarrear consecuencias muy graves para su salud, desde la cronificación de enfermedades hasta la muerte.

En la elaboración de este proyecto, se han tomado como referencia las aplicaciones líderes en el seguimiento de tratamientos médicos: MyTherapy [2] y Medisafe [3].

Se pretende unificar las funcionalidades más destacadas de ambas para crear y ofrecer una aplicación completa que combine sus mejores funcionalidades y características. Por ejemplo, el sistema de notificaciones de MyTherapy, conocido por su fiabilidad y precisión, o la claridad del historial de seguimiento de Medisafe, que facilita a los usuarios y a los profesionales de la salud la revisión de la adherencia al tratamiento.

2

Objetivos

En este capítulo se presentan los objetivos, tanto específicos como generales, que se persiguen con el desarrollo de este proyecto.

2.1. Objetivos Generales

El objetivo general de este proyecto es proporcionar una aplicación web que permita a los usuarios seguir, organizar y gestionar sus tratamientos médicos de la manera más cómoda y fácil posible.

2.2. Objetivos Específicos

Los objetivos específicos que se persiguen con el desarrollo de este proyecto son los siguientes:

- Gestión de usuarios: Permitir a los usuarios registrarse, iniciar sesión, cerrar sesión y editar su perfil en la aplicación.
- Gestión de tratamientos: Permitir a los usuarios ver, añadir, editar y eliminar sus tratamientos.
- Gestión de medicamentos: Permitir a los usuarios ver, añadir, editar y eliminar sus medicamentos.

- Gestión de tomas: Permitir a los usuarios ver, añadir, editar y eliminar sus tomas.
- Base de datos: Utilización de la base de datos MySQL para el entorno de producción y H2 para el desarrollo y testing.
- Empaquetado: Creación de un pipeline para empaquetar y desplegar la aplicación en Docker.
- Pruebas: Desarrollo de pruebas unitarias, de integración y End-to-End, para el correcto control del funcionamiento de la aplicación haciendo uso de frameworks como JUnit, Mockikto y Selenium.
- Entrega continua y despliegue continuo(CI/CD): Creación de pipelines, tanto de integración continua como de despliegue continuo, para automatizar la construcción, el testing y el despliegue de la aplicación mediante el uso de GitHub Actions.

2.2.1. Objetivos de Aprendizaje

Los objetivos de aprendizaje tras realizar este proyecto son los siguientes:

- Seguridad: Aprender a implementar un sistema de autenticación y autorización para proteger los datos de los usuarios.
- Bases de datos: Aprender a manejar bases de datos para gestionar, almacenar y consultar los datos de los usuarios de manera eficiente.
- Notificaciones: Aprender a implementar un sistema de notificaciones por correo electrónico para mantener informados a los usuarios.
- Distribución de imágenes Docker: Aprender a empaquetar y distribuir la aplicación a través de contenedores para que los usuarios puedan disfrutar de la aplicación desde cualquier entorno.
- Pruebas: Aprender a hacer y ejecutar pruebas sobre la aplicación para asegurar la calidad y el correcto funcionamiento de la aplicación.

3

Tecnologías, Herramientas y Metodologías

Este capítulo describe los lenguajes, tecnologías, herramientas y metodologías utilizadas en el desarrollo de la aplicación.

3.1. Lenguajes

3.1.1. Backend

Para el desarrollo del Backend de la aplicación se ha utilizado Java [4] (Figura 3.1). Un lenguaje de programación orientado a objetos muy utilizado en el mundo. Su amplia comunidad y su gran cantidad de librerías hacen que sea una de las opciones más populares para el desarrollo de aplicaciones web.

3.1.2. Frontend

En cuanto a la tecnología utilizada para el desarrollo del Frontend, se han utilizado conjuntamente HTML, CSS y JavaScript [5]. Tres lenguajes de programación que se utilizan para crear aplicaciones web.

HTML (Figura 3.2a) es un lenguaje de marcado que se utiliza para definir y estructurar el contenido de las páginas web. CSS (Figura 3.2b) es un lenguaje de

Figura 3.1: Logo de Java.

estilo que se utiliza para dar formato y diseño a los elementos HTML. JavaScript (Figura 3.2c) es un lenguaje de programación que se utiliza para proporcionar interactividad y dinamismo a las páginas web.

(a) HTML (b) CSS (c) JavaScript

Figura 3.2: Logotipos de HTML, CSS y JavaScript

3.2. Tecnologías

A continuación, se presentarán y explicarán las tecnologías utilizadas para la elaboración del proyecto.

3.2.1. Frontend

Para el desarrollo del Frontend se ha utilizado Mustache [6]. Un motor de plantillas que permite generar HTML de forma dinámica a partir de datos. Separar la lógica de la presentación, lo que simplifica el mantenimiento y la escalabilidad del código.

3.2.2. Backend

Para el Backend se ha utilizado Spring [7]. Un framework de Java que ofrece inyección de dependencias y módulos tales como Spring Test o Spring Security, entre otros, y que ayuda a reducir el tiempo de desarrollo de aplicaciones.

También se ha utilizado Spring Boot [8] (Figura 3.3). Una herramienta que facilita la creación de proyectos de Spring. Se encarga de integrar todas las dependencias necesarias para el desarrollo de la aplicación. También simplifica la configuración y compilación del proyecto, lo que ayuda a ahorrar tiempo.

Figura 3.3: Logo de Spring.

3.2.3. Seguridad

Para gestionar la seguridad de la aplicación se ha utilizado Spring Security [9]. Un módulo de Spring que proporciona a las aplicaciones Java mecanismos personalizables de autenticación, para controlar el acceso a la aplicación y autorización, para gestionar los permisos de usuario y controlar el acceso a las URLs web.

3.2.4. Persistencia de datos y bases de datos

Para almacenar los datos de la aplicación se han utilizado JPA (Java Persistence API) [10], que define y gestiona la persistencia de datos, a través de una serie de interfaces, y Spring Data, un framework que facilita el uso de JPA en proyectos de Spring, proporcionando una capa de abstracción mediante el uso de repositorios para acceder a los datos de manera sencilla y rápida.

En cuanto a las bases de datos, se han utilizado dos diferentes:

- **MySQL** [11] (Figura 3.4a) para la persistencia de datos en producción. Se trata de una base de datos relacional muy popular y ampliamente utilizada en aplicaciones web.
- **H2** [12] **(Figura 3.4b)** para el desarrollo y pruebas de la aplicación ya que es más ligera y fácil de usar. Es una base de datos relacional en memoria, lo que permite realizar pruebas rápidas sin tener que configurar una base de

datos externa. Además, H2 es compatible con JPA y Spring Data, lo que facilita su integración.

(a) MySQL (b) H2

Figura 3.4: Logotipos de MySQL y H2

3.2.5. Notificaciones

Para el envío de notificaciones por correo electrónico se ha utilizado Spring Boot Mail [13]. Se trata de una dependencia que ofrece Spring, la cual facilita el envío de correos electrónicos. Para su configuración, basta con especificar las propiedades, como por ejemplo, el host o el puerto, en el archivo application.properties y Spring se encarga de crear la clase.

3.2.6. Pruebas

Para las pruebas de la aplicación se han utilizado JUnit [14] (Figura 3.5a) y Selenium [15] (Figura 3.5b).

JUnit es un framework de pruebas unitarias para Java que permite ejecutar pruebas automatizadas del código fuente de la aplicación. Se ha utilizado para realizar las pruebas unitarias y de integración de los diferentes componentes de la aplicación, como los controladores, servicios y repositorios, lo que ha permitido detectar errores de forma temprana.

Por otra parte, para las pruebas E2E se ha utilizado Selenium. Un software de código abierto que permite simular, de manera automática, la interacción del usuario con la aplicación.

(a) JUnit (b) Selenium

Figura 3.5: Logotipos de JUnit y Selenium

3.2.7. Empaquetado

Para distribuir la aplicación se ha utilizado Docker [16] (Figura 3.6). Una plataforma que permite crear, desplegar y ejecutar aplicaciones en contenedores.

Los contenedores son entornos aislados que permiten ejecutar aplicaciones de manera independiente del sistema operativo subyacente, lo que aprovecha mejor la infraestructura y aporta mayor seguridad. Al estar basados en imágenes, también mejora su portabilidad y escalabilidad.

Figura 3.6: Logo de Docker.

3.3. Herramientas

Esta sección describe las herramientas utilizadas en el desarrollo de la aplicación.

3.3.1. IDE

Para el desarrollo de la aplicación se ha utilizado IntelliJ IDEA [17] (Figura 3.7). Un IDE (Integrated Development Environment) de Java que ofrece una extensa gama de funcionalidades, como el autocompletado, refactorización y depuración de código, lo que agiliza el desarrollo. A su vez, tiene una integración muy buena con Spring, lo que permite aprovechar al máximo sus beneficios.

Figura 3.7: Logo de IntelliJ IDEA.

3.3.2. Compilación y construcción

Para compilar y construir la aplicación se ha utilizado Maven [18] (Figura 3.8). Un sistema de gestión de librerías de Java que se encarga automáticamente de gestionar las dependencias, compilar y empaquetar la aplicación. Su configuración se hace a través del archivo pom.xml.

Figura 3.8: Logo de Maven.

3.3.3. Control de versiones

Para el control de versiones se han utilizado Git [19](Figura 3.9a) y GitHub [20] (Figura 3.9b).

Git es un sistema de control de versiones distribuido que permite trabajar de manera colaborativa y remota de forma sencilla.

GitHub es una plataforma de desarrollo colaborativo que gestiona las aplicaciones que usan el sistema de Git y que permite guardar proyectos y trabajar con otros desarrolladores.

(a) Git (b) GitHub

Figura 3.9: Logotipos de Git y GitHub

3.3.4. Base de datos

Para la gestión de la base de datos se ha utilizado MySQL Workbench [21] (Figura 3.10). Una herramienta visual que ofrece un entorno integrado en el que

poder visualizar, diseñar, modelar y administrar bases de datos MySQL. Esto permite gestionar de manera más intuitiva y fácil los esquemas, las estructuras complejas y las consultas SQL.

Figura 3.10: Logo de MySQL Workbench.

3.3.5. GitHub Actions

Para implementar la CI/CD se ha usado GitHub Actions [22]. Se trata de una herramienta de automatización integrada en GitHub que permite crear y personalizar flujos de trabajo para que se ejecuten según los eventos que ocurran en la aplicación. Esto ayuda a automatizar los procesos de construcción, prueba y despliegue.

3.4. Metodologías

A continuación se explicará la metodología seguida durante la realización del proyecto.

3.4.1. Trunk Based Development

Durante la realización del proyecto se ha seguido la metodología Trunk Based Development (TBD) [23]. Se trata de un modelo de control de versiones en el que se fusionan a la rama principal pequeñas actualizaciones de forma frecuente. La idea central es mantener la rama principal siempre en un estado desplegable, lo que significa que el software está listo para ser liberado en cualquier momento.

Las principales ventajas del TBD son:

- **Integración continua y entrega continua (CI/CD)**: Facilita la incorporación de la CI/CD, ya que los cambios se fusionan a la rama principal varias veces al día. Esto permite minimizar los conflictos y asegurar que el código está siempre actualizado.

- **Velocidad**: El desarrollo se vuelve más rápido ya que se evitan los conflictos de fusión y los extensos ciclos de revisión que son comunes en otros modelos de desarrollo como Git Flow.
- **Calidad**: Las continuas pruebas y las pequeñas fusiones ayudan a detectar errores de forma temprana, lo que reduce el riesgo de introducir errores importantes en el código base.
- **Menos "merge conflicts"**: Al fusionar pequeños cambios de manera frecuente, es mucho menos probable que haya conflictos de código, y cuando ocurren, debido a que la diferencia entre las ramas es mínima, son mucho más fáciles de resolver.

A continuación, se muestra la estructura de ramas de un proyecto basado en la metodología TBD (Figura 3.11):

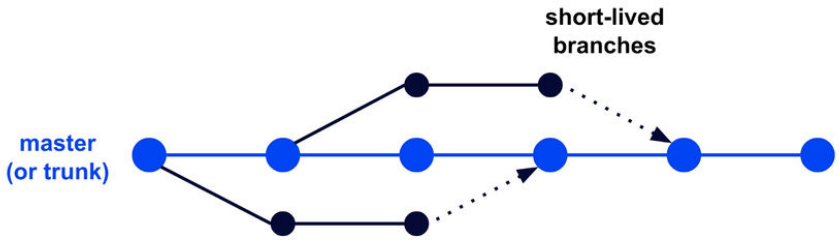


Figura 3.11: Estructura de ramas de un proyecto con TBD.

Se aprecia un flujo mucho más lineal y fácil de seguir. Todas las contribuciones van directamente a la rama principal (main), con pequeñas ramas temporales que se fusionan rápidamente de vuelta.

4

Descripción Informática

En este capítulo se ofrece una visión integral de la aplicación, cubriendo sus requisitos, su arquitectura y el proceso de implementación y despliegue.

Se detallan los requisitos funcionales, que incluyen la gestión de usuarios, medicamentos, tratamientos e historial de tomas, así como la capacidad de enviar notificaciones por correo electrónico. Los requisitos no funcionales se centran en la usabilidad, seguridad, confiabilidad, portabilidad y eficiencia de la aplicación.

La arquitectura se basa en un enfoque de tres capas (frontend, backend y base de datos) y sigue el patrón MVC (Modelo-Vista-Controlador). Se especifica el uso de Docker para la contenerización, facilitando la portabilidad y escalabilidad.

También describe el proceso de desarrollo, mencionando la implementación del frontend con tecnologías web estándar y el backend con Spring Boot en Java. Se explica cómo se utilizan GitHub Actions para implementar un flujo de Integración y Entrega Continua (CI/CD), que automatiza las pruebas (unitarias, de integración y E2E) y el despliegue de la aplicación. Por último, se detalla el uso de Docker Compose para simplificar la ejecución de toda la infraestructura de la aplicación.

4.1. Requisitos

En esta sección se presentan los requisitos iniciales de la aplicación.

4.1.1. Requisitos Funcionales

Gestión de Usuarios y Autenticación

- RF1. Como usuario no registrado, puedo darme de alta en la aplicación.
- RF2. Como usuario no registrado, puedo iniciar sesión en la aplicación.
- RF3. Como usuario registrado, debo poder ver mi perfil en la aplicación.
- RF4. Como usuario registrado debo poder cerrar sesión en la aplicación.
- RF5. Como usuario registrado puedo editar mi perfil en la aplicación.

Gestión de Medicamentos

- RF6. Como usuario registrado puedo ver la lista de mis medicamentos.
- RF7. Como usuario registrado puedo añadir un medicamento a mi lista.
- RF8. Como usuario registrado puedo consultar un medicamento de mi lista.
- RF9. Como usuario registrado puedo editar un medicamento de mi lista.
- RF10. Como usuario registrado puedo eliminar un medicamento de mi lista.
- RF11. Como usuario registrado debo recibir correos cuando las existencias de uno de mis medicamento sean bajas.

Gestión de Tratamientos

- RF12. Como usuario registrado puedo ver la lista de mis tratamientos.
- RF13. Como usuario registrado puedo añadir un tratamiento a mi lista.
- RF14. Como usuario registrado puedo consultar un tratamiento de mi lista.
- RF15. Como usuario registrado puedo editar un tratamiento de mi lista.
- RF16. Como usuario registrado puedo eliminar un tratamiento de mi lista.
- RF17. Como usuario registrado debo poder recibir correos recordándome la toma de mis medicamentos.

Gestión de Historial de Tomas

- RF18. Como usuario registrado puedo ver el historial de mis tomas.
- RF19. Como usuario registrado puedo añadir una toma a mi historial.
- RF20. Como usuario registrado puedo consultar una toma de mi historial.
- RF21. Como usuario registrado puedo editar una toma de mi historial.
- RF22. Como usuario registrado puedo eliminar una toma de mi historial.

4.1.2. Requisitos No Funcionales

RNF1. Usabilidad

- RNF1.1. La aplicación debe contar con una interfaz de usuario intuitiva y fácil de usar.
- RNF1.2. La aplicación debe proporcionar mensajes de error adecuados al usuario.

RNF2. Seguridad

- RNF2.1. La aplicación debe cifrar y almacenar correctamente las contraseñas y datos sensibles de los usuarios.

RNF3. Confiabilidad

- RNF3.1. La aplicación debe estar siempre disponible para los usuarios.
- RNF3.2. La aplicación debe ser tolerante a fallos.

RNF4. Portabilidad

- RNF4.1. La aplicación debe poder usarse en distintas plataformas.

RNF5. Eficiencia

- RNF5.1. La aplicación debe recuperar y mostrar la información en un tiempo máximo de 5 segundos.

4.2. Arquitectura y Análisis

En esta sección se describirá la arquitectura de la aplicación, así como sus componentes más relevantes.

La figura 4.1 representa un diagrama de la arquitectura de la aplicación, el cual muestra los distintos componentes que forman parte de ella y cómo se relacionan entre sí.

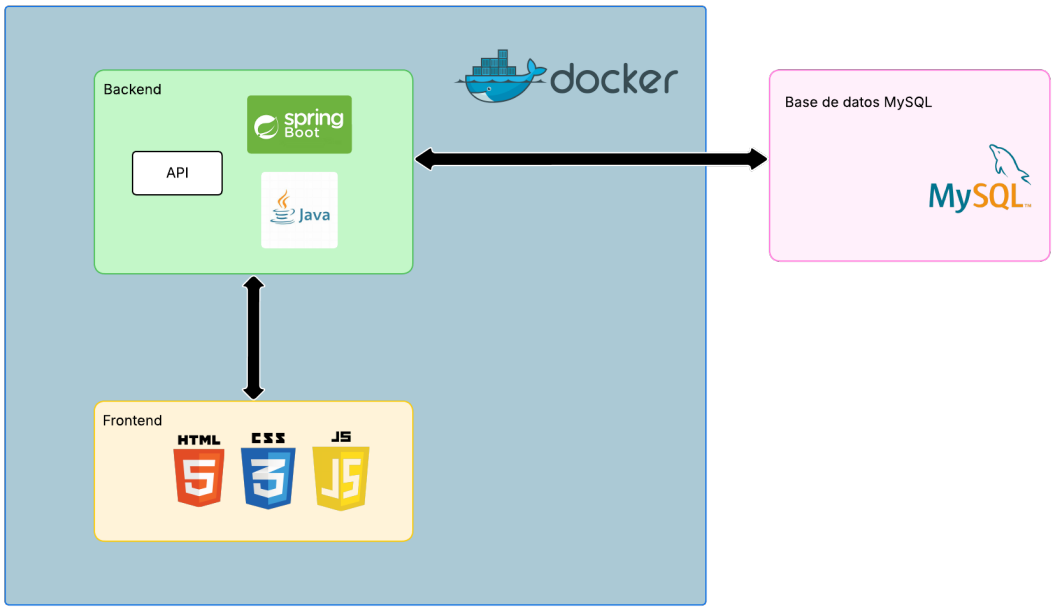


Figura 4.1: Diagrama de arquitectura de la aplicación

Se puede observar que la aplicación está dividida en tres partes principales: el frontend, el backend y la base de datos.

Tanto el frontend como el backend utilizan el patrón de arquitectura MVC (Modelo-Vista-Controlador) para organizar el código y separar las responsabilidades. También están contenerizadas en Docker, lo que facilita la portabilidad y escalabilidad de la aplicación.

4.2.1. Frontend

El frontend está implementado con HTML, CSS y JavaScript, utilizando el motor de plantillas de Mustache para generar las vistas dinámicamente. En la figura 4.2 se muestra la estructura de carpetas.

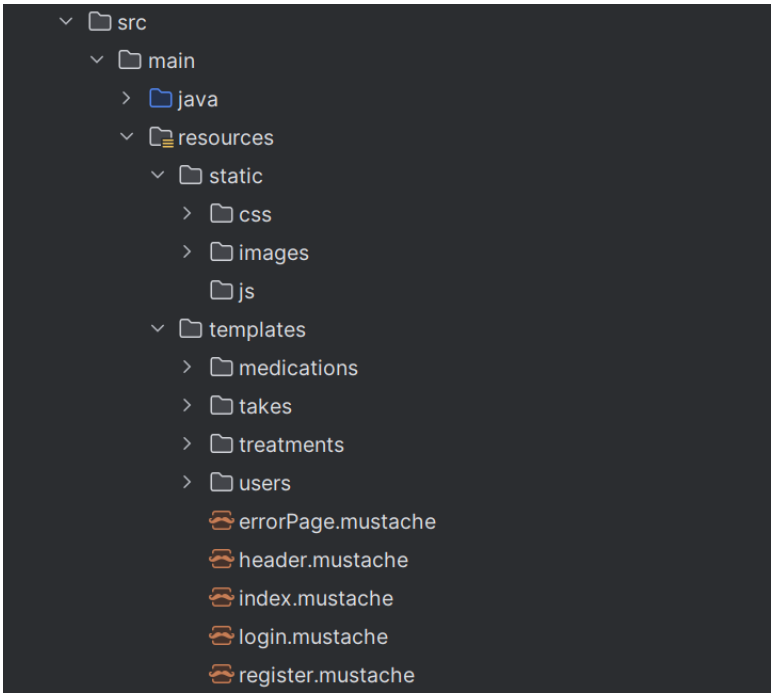


Figura 4.2: Estructura de carpetas del frontend

Tenemos las siguientes carpetas principales:

- **static**: Contiene los recursos estáticos, que son los archivos que se sirven directamente al navegador, en este caso, imágenes, hojas de estilo y scripts.
- **templates**: Contiene las vistas, que son los archivos HTML que se envían al navegador. A su vez, dentro de la carpeta templates, podemos distinguir las siguientes subcarpetas:
  - **users**: Contiene las vistas relacionadas con los usuarios.
    - **user-profile**: Muestra la información del usuario, como su nombre y correo electrónico.
    - **editProfile**: Permite al usuario modificar su información personal.
  - **medications**: Contiene las vistas relacionadas con los medicamentos:
    - **medication-list**: Muestra la lista de medicamentos del usuario.
    - **medication**: Muestra un medicamento concreto.
    - **createMedication**: Permite al usuario crear un nuevo medicamento.
    - **editMedication**: Permite al usuario editar un medicamento existente.
  - **takes**: Contiene las vistas relacionadas con las tomas:

- **takes**: Muestra el historial de tomas del usuario.
- **take**: Muestra una toma concreta.
- **createTake**: Permite al usuario crear una nueva toma.
- **editTake**: Permite al usuario editar una toma existente.

- **treatments**: Contiene las vistas relacionadas con los tratamientos:
  - **treatments**: Muestra la lista de tratamientos del usuario.
  - **treatment**: Muestra un tratamiento concreto.
  - **createTreatment**: Permite al usuario crear un nuevo tratamiento.
  - **editTreatment**: Permite al usuario editar un tratamiento existente.

4.2.2. Backend

El backend de la aplicación está implementado en Java utilizando el framework Spring Boot, el cual proporciona una serie de anotaciones como *@Controller*, *@Service*, *@Repository* y *@Entity*, entre otras, que permiten definir los distintos componentes de la aplicación y su relación entre sí. La siguiente figura 4.3 representa la estructura y organización de carpetas:

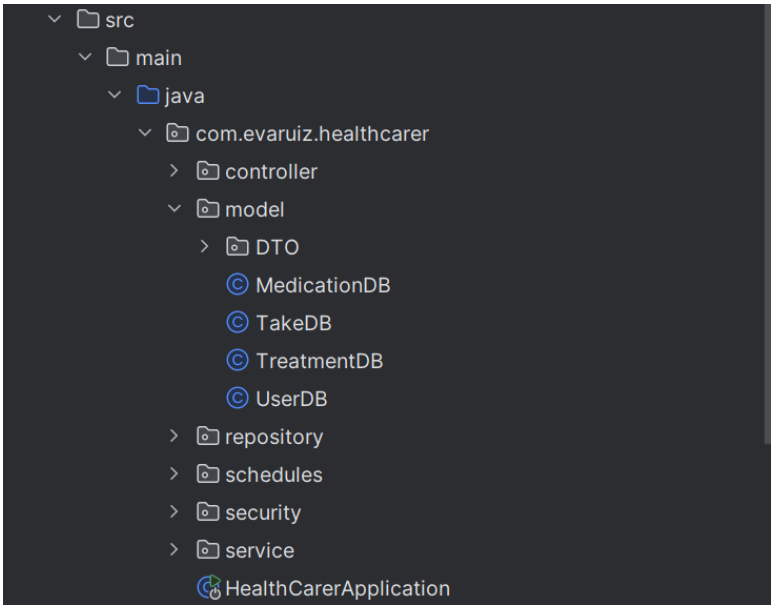


Figura 4.3: Estructura de carpetas del backend

Tenemos las siguientes carpetas principales:

- **controller**: Contiene los controladores que gestionan las peticiones HTTP y la lógica de la aplicación.

- **MedicationController**: Se encarga de gestionar las peticiones relacionadas con los medicamentos, como añadir, editar, eliminar y consultar medicamentos.

- **TakeController**: Se encarga de gestionar las peticiones relacionadas con las tomas de medicamentos, como añadir, editar, eliminar y consultar tomas.

- **TreatmentController**: Se encarga de gestionar las peticiones relacionadas con los tratamientos, como añadir, editar, eliminar y consultar tratamientos.

- **UserController**: Se encarga de gestionar las peticiones relacionadas con los usuarios, como el registro, inicio de sesión, cierre de sesión, edición del perfil y visualización del perfil.

- **model**: Contiene los modelos que representan las entidades de la aplicación, tanto en la base de datos como en la lógica de negocio en forma de DTOs (Data Transfer Objects).

  - **DTO**: Contiene los objetos de transferencia de datos que se utilizan para enviar y recibir información entre el frontend y el backend.

  - **MedicationDB**: Contiene el modelo de datos para los medicamentos.

  - **TakeDB**: Contiene el modelo de datos para las tomas de medicamentos.

  - **TreatmentDB**: Contiene el modelo de datos para los tratamientos.

  - **UserDB**: Contiene el modelo de datos para los usuarios.

A continuación, en la figura 4.4, se muestra un diagrama de clases que representa los modelos de datos y sus relaciones:

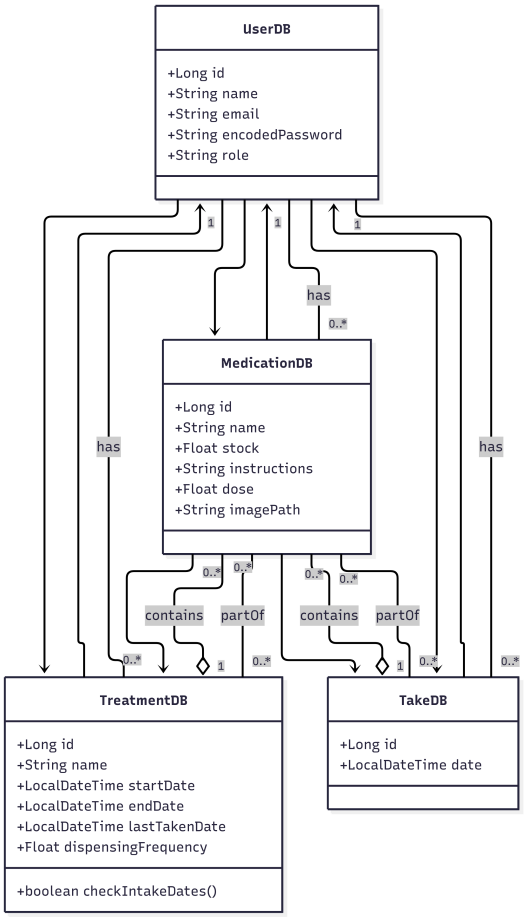


Figura 4.4: Diagrama de clases

- **repository**: Contiene las interfaces y las implementaciones de los repositorios que gestionan el acceso y la persistencia de los datos.

  - **MedicationRepository**: Repositorio para gestionar los medicamentos.
  - **TakeRepository**: Repositorio para gestionar las tomas de tomas.
  - **TreatmentRepository**: Repositorio para gestionar los tratamientos.
  - **UserRepository**: Repositorio para gestionar los usuarios.

  Los repositorios se encargan de la comunicación con la base de datos, permitiendo realizar operaciones CRUD (del inglés, Creación, Lectura, Actualización, Eliminación). La figura 4.5 muestra el esquema relacional de la base de datos utilizada en el proyecto.

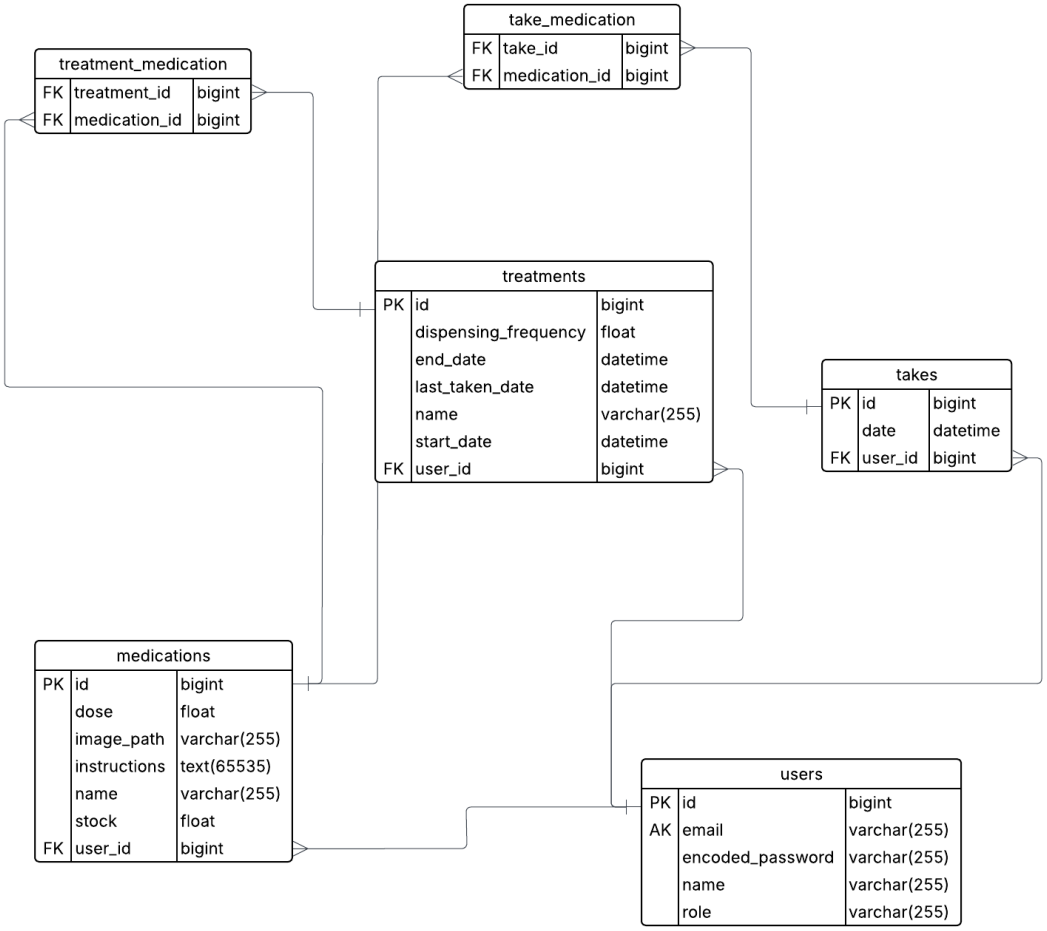


Figura 4.5: Esquema relacional de la base de datos

- **schedules**: Contiene las tareas programadas que se ejecutan periódicamente, en este caso, el envío de correos electrónicos.
  - **ScheduledTasks**.
- **security**: Contiene la configuración de seguridad de la aplicación, incluyendo la autenticación y autorización de usuarios.
  - **SecurityConfiguration**: Se encarga de definir las reglas de seguridad, como qué rutas son accesibles para usuarios no autenticados y cuáles requieren autenticación.
- **service**: Contiene los servicios que encapsulan la lógica de la aplicación y se comunican con los repositorios.
  - **DatabaseInitializer**: Se encarga de inicializar la base de datos con datos de prueba.

- **EmailServiceImpl**: Se encarga de enviar correos electrónicos a los usuarios, como recordatorios de tomas de medicamentos o notificaciones de existencias bajas.
- **ImageService**: Se encarga de gestionar las imágenes de los medicamentos, permitiendo subir, editar o eliminar imágenes asociadas a los mismos.
- **MedicationService**: Se encarga de la lógica relacionada con los medicamentos, como añadir, editar o eliminar medicamentos.
- **TakeService**: Se encarga de la lógica relacionada con las tomas de medicamentos, como añadir, editar o eliminar tomas.
- **TreatmentService**: Se encarga de la lógica relacionada con los tratamientos, como añadir, editar o eliminar tratamientos.
- **UserService**: Se encarga de la lógica relacionada con los usuarios, como el registro, inicio de sesión y gestión del perfil.

4.3. Diseño e Implementación

El código de la aplicación implementada puede encontrarse en `https://github.com/Mandolin03/HealthCarer`

4.3.1. Frontend

Cabe destacar la implementación de un servicio especial para la gestión de imágenes, ya que la aplicación permite a los usuarios subir una imagen para identificar mejor la medicación.

El código 4.1 muestra dicho servicio:

```
1  @Service
2  public class ImageService {
3
4      private final String UPLOAD_DIRECTORY = "/uploads/";
5
6      public String uploadImage(MultipartFile imageFile) throws IOException {
7          String originalFileName = imageFile.getOriginalFilename();
8          String fileExtension = "";
9          int dotIndex = originalFileName != null ?
               originalFileName.lastIndexOf('.') : -1;
10         if (dotIndex > 0 && dotIndex < originalFileName.length() - 1) {
11             fileExtension = originalFileName.substring(dotIndex);
12         }
13         String fileName = UUID.randomUUID() + fileExtension;
14         Path uploadPath = Path.of(UPLOAD_DIRECTORY);
15         if (!Files.exists(uploadPath)) {
16             Files.createDirectories(uploadPath);
17         }
18         Path filePath = uploadPath.resolve(fileName);
```

```
19          Files.copy(imageFile.getInputStream(), filePath,
                StandardCopyOption.REPLACE_EXISTING);
20          return fileName;
21      }
22
23      public void deleteImageFile(String imagePath) throws IOException {
24          if (imagePath == null || imagePath.isEmpty()) {
25              return;
26          }
27          String localFileName = imagePath.replace("/uploads/", "");
28          Path filePath = Path.of(UPLOAD_DIRECTORY).resolve(localFileName);
29          if (Files.exists(filePath)) {
30              Files.delete(filePath);
31          } else {
32              throw new IOException("No se ha encontrado la imagen: " +
                    imagePath);
33          }
34      }
35
36      public FileSystemResource getImageFile(String imagePath) {
37          if (imagePath == null || imagePath.isEmpty()) {
38              return null;
39          }
40          String localFileName = imagePath.replace("/uploads/", "");
41          Path filePath = Paths.get(UPLOAD_DIRECTORY, localFileName);
42          if (Files.exists(filePath)) {
43              return new FileSystemResource(filePath);
44          } else {
45              return null;
46          }
47      }
48
49  }
```

Código 4.1: Servicio de gestión de imágenes

El servicio cuenta con dos métodos:

`uploadImage(MultipartFile imageFile)` se encarga de subir una imagen a un directorio en el servidor. A continuación, se detalla su funcionamiento:

- **Genera un Nombre Único**: En lugar de usar el nombre original del archivo, crea un nombre aleatorio usando `UUID.randomUUID()`. Esto es crucial para la seguridad, ya que evita colisiones de nombres y ataques de "path traversal" (navegar a través de directorios).
- **Determina la Extensión**: Extrae la extensión del nombre original del archivo (ej. `.jpg`, `.png`).
- **Crea el Directorio**: Comprueba si el directorio de subida (`/uploads/`) existe. Si no es así, lo crea automáticamente.
- **Guarda el Archivo**: Copia el contenido del archivo subido (`imageFile`) al nuevo archivo con el nombre único dentro del directorio de subida.
- **Devuelve el Nombre**: Retorna el nombre único del archivo generado, que será la ruta que se guardará en la base de datos para futuras referencias.

`deleteImageFile(String imagePath)` elimina un archivo de imagen del servidor basándose en la ruta que se le proporciona.

- **Valida la Ruta**: Primero, verifica que la ruta de la imagen (`imagePath`) no sea nula o esté vacía para evitar errores.
- **Construye la Ruta Local**: A partir de la ruta completa, extrae el nombre del archivo y construye la ruta completa en el servidor.
- **Verifica la Existencia**: Comprueba si el archivo realmente existe en el disco.
- **Elimina el Archivo**: Si el archivo existe, lo borra.
- **Manejo de Errores**: Si el archivo no se encuentra, lanza una excepción (`IOException`) para notificar que la imagen que se intentó eliminar no existe.

A continuación se muestra la página que detalla un medicamento (Figura 4.6):

4.3.2. Backend

Aquí se explicará cómo se ha implementado el envío de correos electrónicos para notificar a los usuarios sobre la toma de medicamentos y las existencias bajas.

A continuación, se muestra el código (Código 4.2) de la función encargada de enviar los correos electrónicos.

```
1  public void sendSimpleMessage(String to, String subject, String text) {
2
3      SimpleMailMessage message = new SimpleMailMessage();
4      message.setFrom("noreply.healthcarer@gmail.com");
5      message.setTo(to);
6      message.setSubject(subject);
7      message.setText(text);
8      emailSender.send(message);
9
10 }
```

Código 4.2: Función de envío de correos

El proceso de envío de correos electrónicos se realiza mediante la clase `SimpleMailMessage` de Spring, que permite crear mensajes de correo electrónico simples. La función `sendSimpleMessage` recibe como parámetros el destinatario, el asunto y el texto del mensaje, y utiliza el objeto `emailSender`, el cual es una instancia de `JavaMailSender`, para enviar el mensaje.

Esta implementación permite enviar correos de forma generalizada, lo que favorece la reutilización del código en diferentes partes de la aplicación. Así mismo,

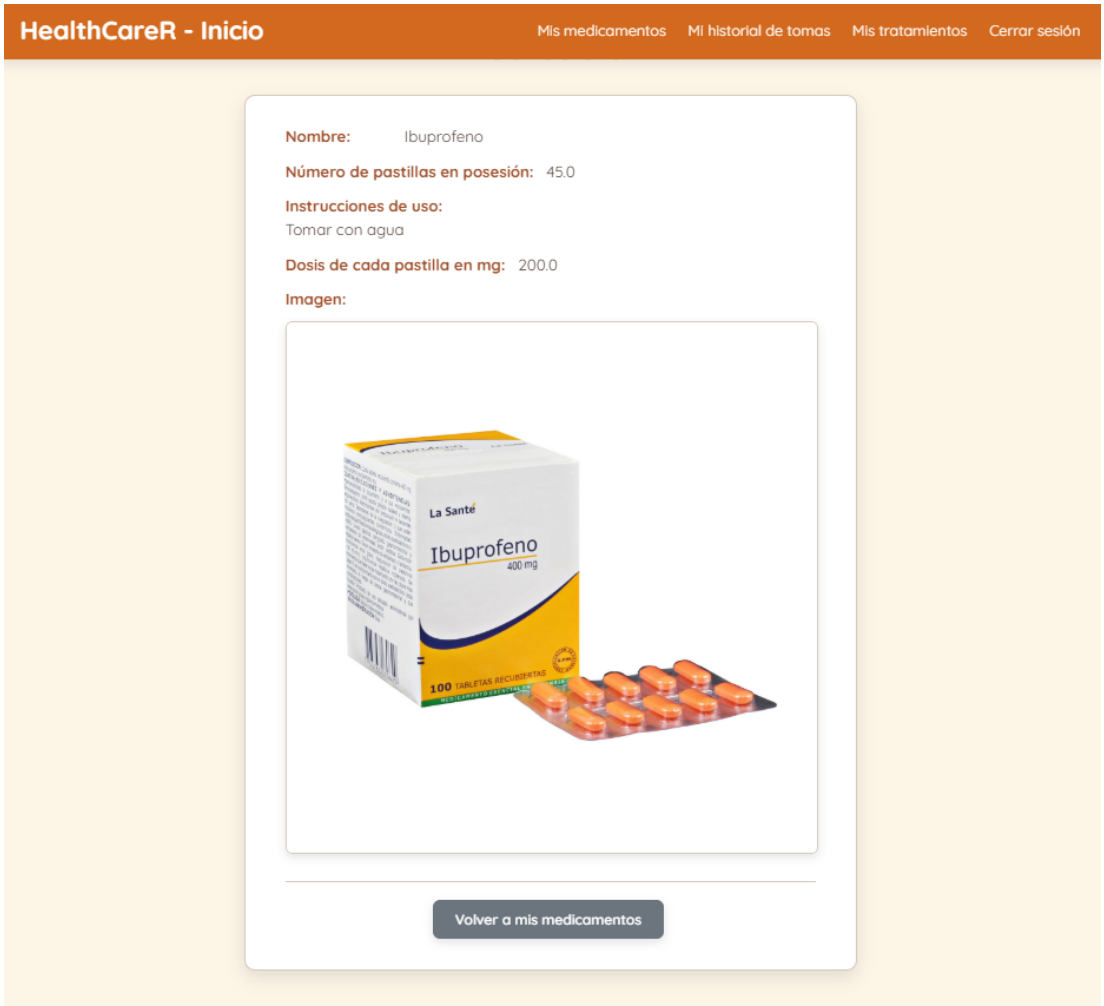


Figura 4.6: Página de detalle de un medicamento.

se ha utilizado para crear dos funciones principales: una para enviar recordatorios de toma de medicamentos y otra para notificar sobre el stock bajo de medicación.

A continuación, se muestra el código (Código 4.3) de la función encargada de enviar recordatorios de toma de medicamentos:

```
1  @Scheduled(fixedRate = 18000)
2  @Transactional
3  public void checkIntakeDates() {
4      List<TreatmentDB> treatments = treatmentRepository.findAll();
5      for (TreatmentDB treatment : treatments) {
6          boolean sendMail = treatment.checkIntakeDates();
7          if (sendMail) {
8              String subject = "Hora de tomar la medicaci n";
9              StringBuilder text = new StringBuilder("Debes tomar la siguiente
                   medicacion: ");
10             for (MedicationDB medication : treatment.getMedications()) {
11                 text.append(medication.getName()).append(" ");
12                 text.append(medication.getDose()).append("mg ");
13             }
14             emailService.sendSimpleMessage(treatment.getUser().getEmail(),
                   subject, text.toString());
```

```
15
16              for (MedicationDB medication : treatment.getMedications()) {
17                  try {
18                      medicationService.discountMedicationStock(medication.getId());
19
20                  } catch (IllegalStateException e) {
21                      String errorSubject = "Error al tomar la medicaci n";
22                      String errorText = "No hay suficiente stock para la
                            medicaci n: " + medication.getName();
23                      emailService.sendSimpleMessage(treatment.getUser().getEmail(),
                            errorSubject, errorText);
24                  }
25              }
26
27              TakeDB take = new TakeDB();
28
29              take.setMedications(new ArrayList<>(treatment.getMedications()));
30              take.setDate(LocalDateTime.now());
31              take.setUser(treatment.getUser());
32              takeRepository.save(take);
33
34
35              treatment.setLastTakenDate(LocalDateTime.now());
36              treatmentRepository.save(treatment);
37          }
38      }
39  }
```

Código 4.3: Envío del recordatorio de toma de medicamentos

Esta función se ejecuta cada 5 minutos (18000 milisegundos) y verifica si hay tratamientos con fechas de toma pendientes. Si encuentra alguno, envía un correo electrónico al usuario recordándole que debe tomar su medicación y descuenta el stock de los medicamentos correspondientes. Si no hay suficiente stock, envía un correo electrónico de error al usuario. Además, registra la toma en el historial de tomas del usuario y actualiza la fecha de la última toma del tratamiento.

La figura 4.7 muestra el correo recibido por el usuario:

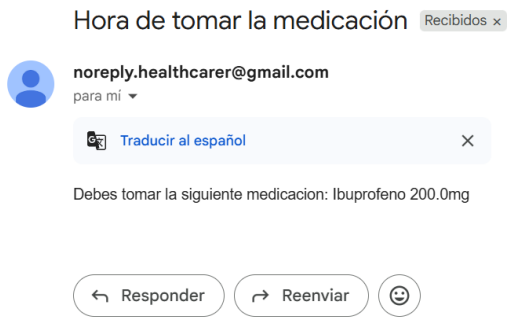


Figura 4.7: Correo de recordatorio de toma de medicamentos.

A continuación, se muestra el código (Código 4.4) de la función encargada de enviar notificaciones sobre el stock bajo de medicación:

```
@Scheduled(fixedRate = 43200)
@Transactional
public void checkMedicationStock() {
    List<TreatmentDB> treatments = treatmentRepository.findAll();
    for (TreatmentDB treatment : treatments) {
        for (MedicationDB medication : treatment.getMedications()) {
            if (medication.getStock() <= 5) {
                String subject = "Stock de medicaci n bajo";
                String text = "La medicaci n " + medication.getName() + "
                    est  por debajo del nivel m nimo de stock. Por favor,
                    reabastece lo antes posible.";
                emailService.sendSimpleMessage(treatment.getUser().getEmail(),
                    subject, text);
            }
        }
    }
}

}
```

Código 4.4: Envío del recordatorio de reposición de stock

Esta función se ejecuta cada 12 horas (43200 milisegundos) y verifica el stock de los medicamentos de todos los tratamientos. Si encuentra algún medicamento con un stock inferior o igual a 5 unidades, envía un correo electrónico al usuario notificándole que debe reabastecer la medicación.

La figura 4.8 muestra el correo recibido por el usuario:

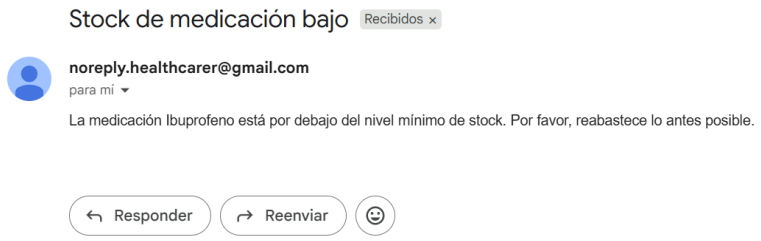


Figura 4.8: Correo de recordatorio de reposición de stock.

4.4. Pruebas

En esta sección se hablará de las pruebas realizadas a la aplicación, tanto pruebas unitarias como pruebas de integración y pruebas de E2E.

Cada clase de prueba tiene una estructura básica que incluye:

- Given: Configuración del entorno de prueba, incluyendo la creación de objetos necesarios y la configuración de mocks. El framework de Junit propor-

ciona anotaciones como @BeforeEach para ejecutar código antes de cada prueba.

- When: Ejecución del método que se va a probar. El framework de Junit proporciona anotaciones como @Test para marcar un método como una prueba.
- Then: Verificación de los resultados esperados, comprobando que el resultado del método coincide con lo esperado. El framework de Junit proporciona aserciones como assertEquals, assertTrue, etc., para verificar los resultados.

4.4.1. Pruebas Unitarias

Se han realizado un total de 28 tests utilizando JUnit 5 y Mockito. Estos tests prueban bloques pequeños y aislados de código para asegurar que funcionan y se comportan correctamente. Ayudan a identificar de forma rápida y aislada el origen de los errores.

A continuación se muestra un ejemplo de una prueba unitaria (Código 4.5):

```
1  @Test
2  @DisplayName("GET user by ID")
3  void userById() {
4  Long userId = existingUser.getId();
5  when(userRepository.findById(userId)).thenReturn(Optional.of(existingUser));
6
7  UserDB foundUser = userService.findById(userId);
8
9  assertNotNull(foundUser);
10 assertEquals(existingUser, foundUser);
11 verify(userRepository, times(1)).findById(userId);
12 verifyNoMoreInteractions(userRepository);
13 }
```

Código 4.5: Ejemplo de test unitario

La parte de Given se realiza mediante la creación de un usuario existente y la configuración del mock del repositorio de usuarios. La parte de When se realiza llamando al método findById del servicio de usuarios. La parte de Then se realiza verificando que el usuario encontrado no sea nulo y que sea igual al usuario existente. Finalmente, se verifica que el repositorio de usuarios haya sido llamado una vez con el ID del usuario y que no haya más interacciones con el repositorio.

4.4.2. Pruebas de Integración

Se han realizado 32 pruebas de integración utilizando JUnit 5 y Spring Boot Test. Estas pruebas aseguran que los diferentes componentes de la aplicación se comunican y funcionan correctamente entre sí.

A continuación se muestra un ejemplo de una prueba de integración (Código 4.6):

```
@Test
@DisplayName("GET user by email from DB")
void userByEmailFromDB() {
    UserDetails userDetails =
        userService.loadUserByUsername(savedUser.getEmail());

    assertNotNull(userDetails, "UserDetails should not be null");
    assertEquals(savedUser.getEmail(), userDetails.getUsername());
    assertTrue(passwordEncoder.matches("initialPass",
        userDetails.getPassword()), "Loaded password should match original");
    assertTrue(userDetails.getAuthorities().stream()
    .anyMatch(a -> a.getAuthority().equals(savedUser.getRole())), "Role should
        match");
}
```

Código 4.6: Ejemplo de test de integración

La parte de Given se realiza mediante la creación de un usuario guardado en la base de datos. La parte de When se realiza llamando al método loadUserByUsername del servicio de usuarios. La parte de Then se realiza verificando que los detalles del usuario no sean nulos, que el nombre de usuario coincida con el correo electrónico del usuario guardado, que la contraseña coincida con la contraseña original y que el rol del usuario coincida con el rol guardado.

4.4.3. Pruebas de E2E

Se han realizado 21 pruebas de E2E utilizando Selenium. Estas pruebas simulan los distintos caminos de flujo del usuario en la web. Se utilizan para verificar que las interacciones con la aplicación y la comunicación de datos son correctas.

A continuación se muestra un ejemplo de una prueba de E2E (Código 4.7):

```
@Test
@DisplayName("Login with existing user")
public void loginWithExistingUser() {
    driver.get("https://localhost:" + this.port + "/login");
    wait.until(ExpectedConditions.titleIs("Inicio de sesi n"));
    driver.findElement(By.id("email")).sendKeys("bob@example.com");
    driver.findElement(By.id("password")).sendKeys("password");
    driver.findElement(By.id("login")).click();
    wait.until(ExpectedConditions.titleIs("HealthCareR"));

}
```

Código 4.7: Ejemplo de test E2E

La parte de Given se realiza accediendo a la página de inicio de sesión. La parte de When se realiza introduciendo el correo electrónico y la contraseña del usuario existente y haciendo clic en el botón de inicio de sesión. La parte de Then se realiza verificando que el título de la página haya cambiado a "HealthCareR", lo que indica que el usuario ha iniciado sesión correctamente.

4.5. Integración Continua y Entrega Continua (CI/CD)

En esta sección se explicará cómo se ha implementado la integración continua y entrega continua (CI/CD) en el proyecto. Para ello, se describirá el uso de herramientas como GitHub Actions, así como la configuración de los pipelines de CI/CD para automatizar el proceso de pruebas y despliegue de la aplicación.

4.5.1. GitHub Actions

A continuación, se muestran algunos ejemplos de las pruebas automáticas ejecutadas durante el desarrollo del proyecto y que han permitido mejorar y acelerar su progreso:

La Figura 4.9 muestra el resumen de un flujo de trabajo de GitHub Actions que se ejecutó con éxito al hacer un commit.

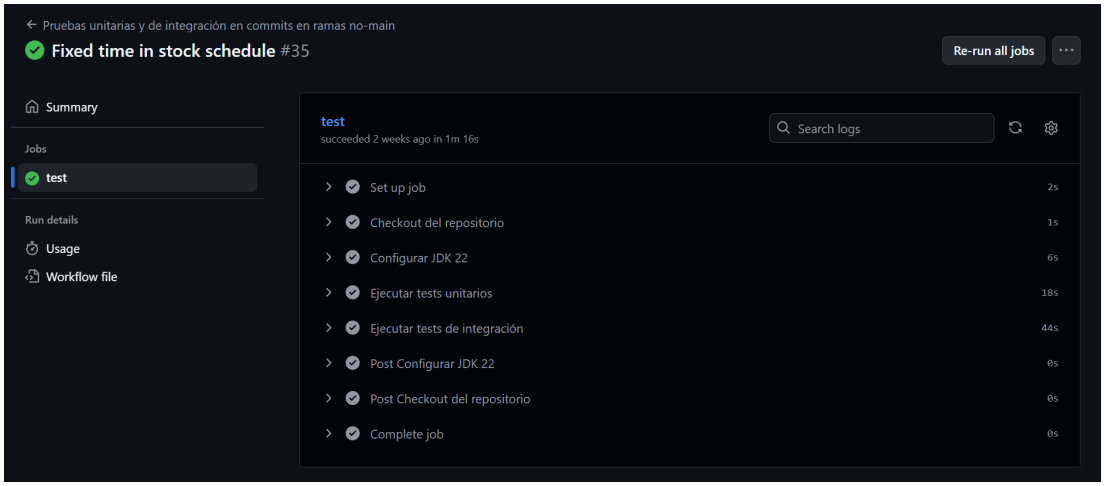


Figura 4.9: Ejecución de pruebas tras un commit.

Se pueden apreciar cada uno de los pasos individuales del trabajo, todos ejecutados correctamente:

- **Set up job**: Preparación del entorno de ejecución.
- **Checkout del repositorio**: Descarga del código del repositorio.
- **Configurar JDK 22**: Configuración del entorno de desarrollo.
- **Ejecutar tests unitarios**: Ejecución de las pruebas unitarias.
- **Ejecutar tests de integración**: Ejecución de las pruebas de integración.

- **Post Configurar JDK 22**: Pasos de limpieza posteriores a la configuración.
- **Post Checkout del repositorio**: Pasos de limpieza posteriores a la descarga.
- **Complete job**: Fin del proceso.

La figura 4.10 muestra el resumen de un flujo de trabajo de GitHub Actions que se ejecutó con éxito en un pull request.

Figura 4.10: Ejecución de pruebas tras un pull request.

Se pueden apreciar cada uno de los pasos individuales del trabajo, todos ejecutados correctamente:

- **Set up job**: Preparación del entorno de ejecución.
- **Checkout del repositorio**: Descarga del código del repositorio.
- **Configurar JDK 22**: Configuración del entorno de desarrollo.
- **Ejecutar tests unitarios**: Ejecuciñon de las pruebas unitarias.
- **Ejecutar tests de integración**: Ejecuciñon de las pruebas de integración.
- **Post Configurar JDK 22**: Pasos de limpieza posteriores a la configuración.
- **Post Checkout del repositorio**: Pasos de limpieza posteriores a la descarga.
- **Complete job**: Fin del proceso.

4.5.2. Integración Continua

La integración continua es una metodología la cual se basa en ejecutar las pruebas automáticas de la aplicación cada vez que haya un cambio en el código del repositorio, usualmente en sistemas de control como Git. Esto permite detectar errores de forma temprana y reduce la deuda técnica.

Para ello, se han configurado dos pipelines en GitHub Actions:

El código del primer pipeline es el siguiente (Código 4.8):

```
name: Pruebas unitarias y de integraci n en commits en ramas no-main

on:
  push:
    paths:
      - src/**
      - pom.xml
    branches-ignore:
      - main

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout del repositorio
        uses: actions/checkout@v3
      - name: Configurar JDK 22
        uses: actions/setup-java@v3
        with:
          distribution: 'temurin'
          java-version: '22'
      - name: Ejecutar tests unitarios
        run: mvn test -Dtest="**/unitary/*"
      - name: Ejecutar tests de integraci n
        run: mvn test -Dtest="**/integration/*"
```

Código 4.8: Workflow de ejecución de pruebas en commits en ramas no-main

Se encarga de ejecutar las pruebas unitarias y de integración cada vez que se realiza un commit en una rama que no sea la rama principal (main). Esto permite detectar errores en el código antes de que se realice un pull request a la rama principal. El pipeline se activa cuando se realizan cambios en los archivos del directorio ‘src/‘ o en el archivo ‘pom.xml‘, y se ejecuta en todas las ramas excepto en la rama ‘main‘.

El segundo pipeline es el siguiente (Código 4.9):

```
name: Pruebas unitarias/integraci n/e2e en pull-requests

on:
  workflow_dispatch:
  pull_request:
    branches:
      - main

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
```

```
13        - name: Checkout del repositorio
14          uses: actions/checkout@v3
15        - name: Configurar JDK 22
16          uses: actions/setup-java@v3
17          with:
18            distribution: 'temurin'
19            java-version: '22'
20        - name: Ejecutar tests unitarios
21          run: mvn test -Dtest="**/unitary/*"
22        - name: Ejecutar tests de integraci n
23          run: mvn test -Dtest="**/integration/*"
24        - name: Ejecutar tests de sistema
25          run: mvn test -Dtest="**/e2e/*"
```

Código 4.9: Workflow de ejecución de pruebas en pull-requests

Se encarga de ejecutar las pruebas unitarias, de integración y de sistema (end-to-end) cada vez que se realiza un pull request a la rama principal (main). Esto permite verificar que los cambios propuestos en el pull request no rompen la funcionalidad existente. El pipeline se activa manualmente o cuando se crea un pull request a la rama 'main'.

4.5.3. Entrega Continua

La entrega continua es una metodología de trabajo en la cual los cambios hechos en el repositorio de código se construyen, se prueban y se despliegan de forma automática. Esto evita que se introduzcan nuevos errores y/o regresiones.

Para ello, se ha configurado el siguiente pipeline en GitHub Actions (Código 4.10):

```
1  name: Build and Push Docker 'dev' Image
2
3  on:
4    push:
5      branches:
6        - master
7
8  jobs:
9    build_and_push_dev_image:
10     runs-on: ubuntu-latest
11
12     steps:
13       - name: Checkout Repository
14         uses: actions/checkout@v3
15
16       - name: Log in to Docker Hub
17         uses: docker/login-action@v2
18         with:
19           username: ${{ secrets.DOCKER_USERNAME }}
20           password: ${{ secrets.DOCKER_PASSWORD }}
21
22       - name: Build and Push Docker image
23         uses: docker/build-push-action@v4
24         with:
25           context: .
26           push: true
```

```
27          tags: ${{ secrets.DOCKER_USERNAME }}/healthcarer:dev
```

Código 4.10: Workflow de construcción y subida de la imagen docker

Este pipeline se encarga de construir y subir la imagen Docker de la aplicación a Docker Hub cada vez que se realiza un push a la rama 'master'. Esto permite que la última versión de la aplicación esté siempre disponible para su despliegue en entornos de desarrollo. El pipeline utiliza las credenciales almacenadas en los secretos de GitHub ('DOCKER_USERNAME' y 'DOCKER_PASSWORD') para autenticarse en Docker Hub y subir la imagen.

4.6. Distribución y despliegue

En esta sección se hablará de cómo se ha empaquetado la aplicación en Docker, así como de cómo puede utilizarse en un entorno propio.

4.6.1. Docker

La imagen docker de la aplicación se ha creado utilizando el siguiente Dockerfile (Código 4.11) que define los pasos necesarios para construir la imagen:

```
1  FROM maven:3.9.5-eclipse-temurin-21 AS builder
2
3  WORKDIR /app
4
5  COPY pom.xml .
6  COPY src ./src
7
8  RUN mvn clean package -Dmaven.test.skip=true
9
10 FROM openjdk:24
11
12 WORKDIR /app
13
14 COPY --from=builder /app/target/*.jar app.jar
15
16 EXPOSE 8443
17
18 ENTRYPOINT ["java", "-jar", "app.jar"]
```

Código 4.11: Archivo dockerfile

En este Dockerfile, se utiliza una imagen base de Maven para compilar la aplicación y luego se copia el archivo JAR resultante a una imagen base de OpenJDK. Esto permite que la aplicación se ejecute en un entorno Java sin necesidad de tener Maven instalado en el contenedor final. También se expone el puerto 8443, que es el puerto por defecto en el que se ejecuta la aplicación Spring Boot. Por último, se define el comando de entrada para ejecutar la aplicación Java.

4.6.2. Docker Compose

Para facilitar la gestión de la aplicación y sus dependencias, se ha creado el siguiente archivo docker-compose.yml (Código 4.12) el cual define los servicios necesarios para ejecutar la aplicación:

```yaml
version: '3.8'

services:
  mysql:
    image: mysql:8.0
    container_name: mysql
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: healthcarer
    volumes:
      - mysql_data:/var/lib/mysql
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "-h", "localhost", "-uroot", "-proot"]
      interval: 10s
      timeout: 5s
      retries: 5

  app:
    image: mandolin03/healthcarer:dev
    container_name: app
    ports:
      - "8443:8443"
    depends_on:
      mysql:
        condition: service_healthy
    environment:
      - SPRING_PROFILES_ACTIVE=dev
      - DB_HOST=mysql
      - DB_USER=root
      - DB_PASSWORD=root
    command: ["sh", "-c", "sleep 15 && java -jar /app/app.jar"]

volumes:
  mysql_data:
```

Código 4.12: Archivo docker-compose

Se ha definido un servicio para MySQL y otro para la aplicación. El servicio de MySQL utiliza la imagen oficial de MySQL y define las variables de entorno necesarias para configurar la base de datos, como la contraseña del usuario root y el nombre de la base de datos. Además, se ha añadido un volumen para persistir los datos de la base de datos y un chequeo de salud para asegurarse de que el servicio está listo antes de que la aplicación intente conectarse a él. El servicio de la aplicación utiliza la imagen creada anteriormente y expone el puerto 8443.

También se define una dependencia del servicio de MySQL, asegurando que la aplicación no se inicie hasta que MySQL esté saludable. Se han añadido variables de entorno para configurar el perfil de Spring y las credenciales de la base de datos. Además, se ha añadido un comando para esperar 15 segundos antes de iniciar la aplicación, lo que permite que MySQL tenga tiempo suficiente para arrancar y estar listo para aceptar conexiones.

4.6.3. Ejecución de la aplicación

Para ejecutar la aplicación, se debe tener instalado Docker y Docker Compose en el sistema. Una vez que se tiene todo instalado, se pueden seguir los siguientes pasos:

1. Clonar el repositorio de la aplicación desde GitHub.
2. Navegar al directorio del proyecto clonado.
3. Ejecutar el comando `docker-compose up --build` para construir y levantar los servicios definidos en el archivo docker-compose.yml.
4. Una vez que los servicios estén en funcionamiento, se puede acceder a la aplicación a través de un navegador web en la dirección `https://localhost:8443`.
5. Para detener los servicios, se puede utilizar el comando `docker-compose down`.

5

Conclusiones y trabajos futuros

Para finalizar, se presentan las conclusiones del trabajo realizado.

5.1. Objetivos cumplidos

El objetivo principal de este proyecto era desarrollar una aplicación web que permitiera a los usuarios gestionar sus medicamentos, tratamientos y tomas de forma eficiente y segura. A lo largo del desarrollo, se han cumplido los siguientes objetivos:

- Se ha implementado un sistema de registro e inicio de sesión para usuarios, garantizando la seguridad de los datos personales.
- Se ha desarrollado una interfaz intuitiva y fácil de usar, permitiendo a los usuarios gestionar sus medicamentos, tratamientos y tomas de manera eficiente.
- Se han implementado funcionalidades para añadir, editar y eliminar medicamentos, tratamientos y tomas, así como para consultar el historial de tomas.
- Se ha integrado un sistema de notificaciones por correo electrónico para recordar a los usuarios la toma de sus medicamentos y alertarles sobre existencias bajas.

- Se ha garantizado la seguridad de los datos sensibles mediante el cifrado de contraseñas y la implementación de medidas de seguridad adecuadas.

5.2. Mejoras futuras

A pesar de que los objetivos iniciales se han alcanzado con éxito, el proyecto presenta varias oportunidades para su desarrollo y optimización. Las áreas clave para futuras mejoras incluyen:

- **Escalabilidad mejorada a través de Docker Compose:** Introducir una configuración de Docker Compose con réplicas de los servicios del *backend* para facilitar la escalabilidad horizontal y simplificar el proceso de despliegue.
- **Despliegue nativo en la nube de AWS:** Realizar la transición a un entorno de nube dedicado, desplegando la aplicación en Amazon EC2 y la base de datos en Amazon RDS para un mejor rendimiento y una gestión más sencilla.
- **Infraestructura robusta en la nube:** Implementar servicios avanzados de AWS como **Auto Scaling Groups** y un **balanceador de carga** para asegurar una alta disponibilidad y una distribución eficiente del tráfico. Se realizarán pruebas de carga para validar la escalabilidad del sistema.
- **Aprovisionamiento y despliegue automatizados:** Adoptar un enfoque de **Infraestructura como Código (IaC)** utilizando AWS CloudFormation para automatizar el aprovisionamiento de recursos y establecer una sólida tubería de despliegue continuo.

Bibliografía

[1] W. H. Organization, "Adherence to long-term therapies : evidence for action," p. 196 p., 2003.

[2] "MyTherapy: aplicación de recordatorio de medicación y gestión de la salud." [Online]. Available: https://www.mytherapyapp.com/es

[3] Medisafe, "Digital Health • Platform • Adherence • Persistence • Solutions — Medisafe." [Online]. Available: https://www.medisafe.com/

[4] Oracle, "Java, lenguaje de programación," 1995. [Online]. Available: http://www.oracle.com/technetwork/java/index.html

[5] "Guía de Desarrollo Web - El proyecto MDN — MDN," 2025. [Online]. Available: https://developer.mozilla.org/es/docs/MDN/Guides

[6] J. D. Journal, "Mustache with Spring Boot — Java Development Journal," 2018. [Online]. Available: https://javadevjournal.com/spring-boot/spring-boot-mustache/

[7] "Spring framework." [Online]. Available: https://spring.io/projects/spring-framework

[8] "Spring boot." [Online]. Available: https://spring.io/projects/spring-boot

[9] "Spring Security." [Online]. Available: https://spring.io/projects/spring-security

[10] "Spring Data JPA." [Online]. Available: https://spring.io/projects/spring-data-jpa

[11] J. Erickson, "MySQL: Understanding What It Is and How It's Used," 2024. [Online]. Available: https://www.oracle.com/mx/mysql/what-is-mysql/

[12] C. Caules, "H2 Database Java y acceso remoto," 2023. [Online]. Available: https://www.arquitecturajava.com/h2-database-java-y-acceso-remoto/

[13] "Email :: Spring Framework." [Online]. Available: https://docs.spring.io/spring-framework/reference/integration/email.html

[14] "Overview (JUnit 5.13.4 API)." [Online]. Available: https://docs.junit.org/current/api/

[15] "Selenium." [Online]. Available: https://www.selenium.dev/

[16] S. Ratliff, "Docker: Accelerated Container Application Development," 2025. [Online]. Available: https://www.docker.com/

[17] "IntelliJ IDEA – the IDE for Pro Java and Kotlin Development." [Online]. Available: https://lp.jetbrains.com/intellij-idea-promo/?msclkid=05d92233d70f10a7b08f4aa1323efa72&utm_source=bing&utm_medium=cpc&utm_campaign=EMEA_en_ES_IDEA_Branded&utm_term=intellij&utm_content=intellij%20idea

[18] "Introduction – Maven." [Online]. Available: https://maven.apache.org/what-is-maven.html

[19] "Git." [Online]. Available: https://git-scm.com/

[20] "GitHub.com Documentación de ayuda." [Online]. Available: https://docs.github.com/es

[21] "MySQL :: MySQL Workbench." [Online]. Available: https://www.mysql.com/products/workbench/

[22] "GitHub Actions documentation - GitHub Docs," 2001. [Online]. Available: https://docs.github.com/en/actions

[23] Atlassian, "Trunk-based development — Atlassian." [Online]. Available: https://www.atlassian.com/continuous-delivery/continuous-integration/trunk-based-development