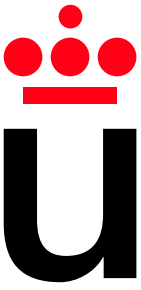


Universidad
Rey Juan Carlos

Escuela Técnica Superior
de Ingeniería Informática

Grado en Ingeniería del Software

Curso 2023-2024

Trabajo Fin de Grado

SMILELINK: UN SOFTWARE PARA LA GESTIÓN DE PETICIONES Y DOCUMENTOS DE LOS CLIENTES DE UNA CLÍNICA DENTAL

Autor: Sergio Cuadros Flores
Tutor: Michel Maes Bermejo

Agradecimientos

Quisiera expresar mi más sincero agradecimiento a todas las personas que han hecho posible la realización de este trabajo de fin de grado. En primer lugar, quiero agradecer a mi tutor Michel Maes Bermejo, por su increíble guía, paciencia y apoyo constante durante todo el proceso. Sus consejos y su profundo conocimiento sobre la materia han sido fundamentales para la realización de este proyecto.

Agradezco también a mis profesores y profesoras de la Universidad Rey Juan Carlos, quienes a lo largo de estos años me han brindado las herramientas necesarias para hacer que esto sea posible.

Un agradecimiento especial a mis compañeros y compañeras de clase, quienes con su colaboración y amistad han hecho de este camino una experiencia inolvidable.

No puedo terminar sin dejar de mencionar a mi familia, quienes siempre han estado a mi lado brindándome su amor y apoyo incondicional. Gracias por creer en mí y por ser mi pilar durante todos estos años. A mis padres, por su sacrificio y por enseñarme el valor del esfuerzo y la perseverancia. A mis amigos, por su comprensión y por estar siempre dispuestos a escucharme y animarme en los momentos más difíciles.

Resumen

En la era tecnológica nos encontramos con un problema que para unos puede ser irrelevante y para otros necesario. La información que las entidades sanitarias tienen sobre nuestra persona, y la capacidad que tenemos nosotros como usuarios para acceder a ella. De esta forma, SmileLink trata de dejar a disposición del usuario su información, historial y documentos clínicos, entre otras cosas, que la entidad posee sobre él.

Este Trabajo de Fin de Grado (TFG) se centra en el desarrollo de una aplicación web para la gestión clínica, denominada SmileLink. El objetivo principal es proporcionar una plataforma eficiente para la gestión de pacientes, clínicas, citas, historiales clínicos y documentos, utilizando tecnologías modernas como Spring Boot para el backend y Angular para el frontend.

Este documento aborda la problemática y motivación detrás del desarrollo de una aplicación web, proporcionando una visión general del proyecto. Se establecen los objetivos generales, específicos y técnicos que se quieren alcanzar con esta iniciativa. Se detallan las tecnologías y herramientas empleadas, así como las metodologías ágiles adoptadas para su desarrollo y despliegue. Asimismo, se profundiza en los requisitos de la aplicación para cada rol de usuario y a nivel de sistema, describiendo la arquitectura implementada y realizando un análisis de la web con LightHouse. Se incluye un análisis detallado de la implementación de cada función (Backend, Frontend, Base de datos), la realización de pruebas de código y su integración continua, junto con el despliegue continuo mediante AWS y GitHub Actions. Finalmente, se reflexiona sobre los resultados obtenidos y se proponen futuras mejoras y tareas para el avance de la aplicación.

Palabras clave:

- Spring Boot
- Angular
- AWS (Amazon Web Services)
- CI/CD
- SAAS (Software As A Service)
- Selenium
- RestAssured
- API REST
- Bases de datos

Índice de contenidos

Índice de figuras

Índice de códigos

1 Introducción y motivación

Este capıtulo explica de donde surge la motivación para la construcción del proyecto y su alcance.

1.1. Contexto y alcance

Hoy en día nuestros datos se mueven sin control por internet, pero, ¿qué cantidad de datos poseemos sobre nosotros mismos y cómo los gestionamos?. En el ámbito de la sanidad pública particularmente, la gestión de citas y documentos sigue siendo un proceso complejo, poco eficiente y poco adaptado a los tiempos que corren. SmileLink nace con el objetivo de dar solución al problema que nuestra sanidad pública nos presenta. La visualización y gestión de citas y documentos de manera digital por parte del paciente.

A lo largo de una vida, es común someterse a diversos análisis de sangre, radiografías, cirugías y empastes. Sin embargo, el acceso instantáneo a estos registros suele ser limitado, y es probable que se conserven pocos o ninguno. ¿Verdad?

La idea de este proyecto surge de una experiencia personal que refleja una situación común: la necesidad de ver unos análisis de sangre realizados hace unos meses. El proceso tradicional sería que pidas cita con tu médico de cabecera, solicites hacerte unos análisis, irte a tu casa, volver a los días a hacer la extracción, pedir cita de nuevo para que te de los resultados el médico, acudir, que te imprima los análisis el doctor en el caso en que quieras llevártelos y buscarle un lugar en tu casa para no olvidar dónde los dejaste. Este proceso está bien y es efectivo, pero, ¿cuánto tiempo y material hemos empleado en este proceso?

El tiempo de espera en el sector público sanitario es una problemática que cada vez está cogiendo más peso. Según un informe publicado por el Ministerio de Sanidad de España [1], la espera media para una consulta con el especialista es de 148 días, y de varias semanas para análisis de sangre o radiografías. Estas demoras se han normalizado de manera insólita, haciendo que a ciertas personas le generen frustración y ansiedad, además de que también, como es obvio, se retrasan los diagnósticos y tratamientos críticos. Además, en muchos casos los pacientes tienen que lidiar con la burocracia y protocolos que carecen de efectividad en la gestión de sus datos médicos, llevando a los pacientes a situaciones de estrés y malestar. Podemos ver en la figura 1.1 un gráfico del tiempo medio de espera en días por cada comunidad autónoma.

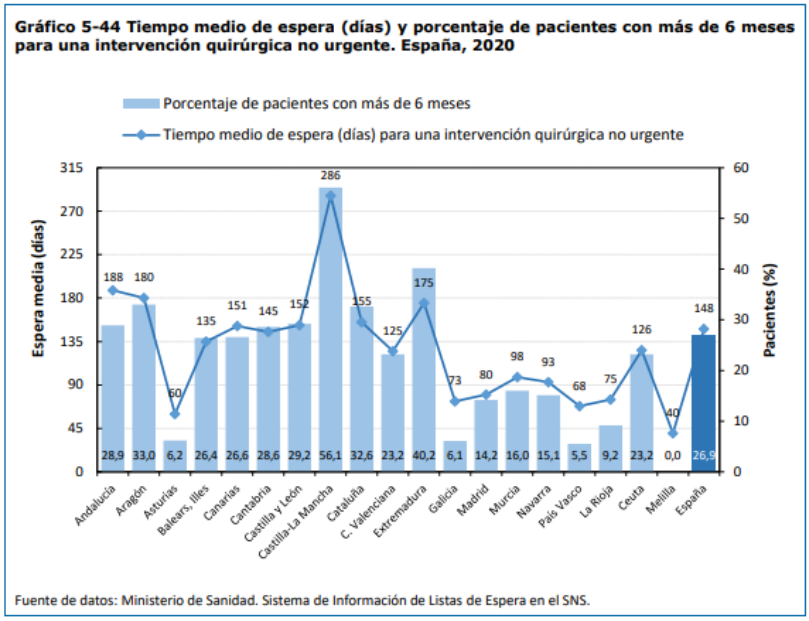


Figura 1.1: Tiempo medio de espera (días) y porcentaje de pacientes con más de 6 meses para una intervención quirúrgica no urgente. España, 2020

SmileLink se postula como un sistema alternativo y eficiente para la gestión de información médica, en este caso, en el sector dental, por centrarnos en un sector y no abarcar tantos campos. La aplicación permitirá a los pacientes solicitar citas online, ver su historial médico, sus intervenciones y documentación, cumpliendo con las acciones básicas necesarias para el paciente, desde una única plataforma. Además, esta digitalización no solo ahorra tiempo, sino que también reduce significativamente el uso de papel, contribuyendo a un entorno más sostenible.

En España se consumió 7.038.927 toneladas de papel en el 2022 [2], mostrando un crecimiento del 2 % respecto al año 2021. Esto infiere en la problemática del consumo de papel por persona que registró un consumo de 141,4 kg per cápita, situando a España en el sexto mayor consumidor de la Unión Europea. Este dato

nos hace reflexionar sobre el impacto ambiental de nuestros hábitos comunes y la necesidad de adoptar medidas alternativas. La reducción del uso del papel no solo tiene beneficios ambientales, sino que también un significante ahorro económico para las arcas del Estado. En la figura 1.2 podemos ver la evolución del consumo del papel en España.

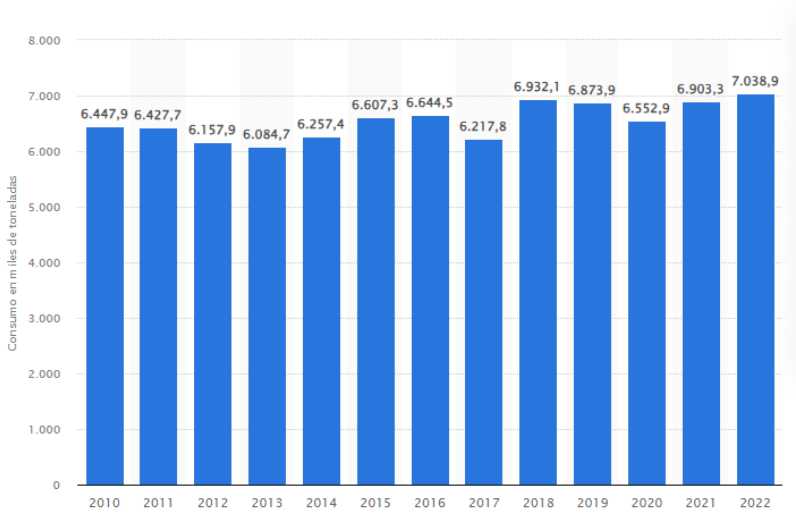


Figura 1.2: Consumo de papel en España (miles de toneladas) [3]

SmileLink busca facilitar la vida de los pacientes proporcionando acceso inmediato a su información dental, eliminando la necesidad de recordar o almacenar documentos físicos, además de reducir las visitas innecesarias al médico contribuyendo a la reducción de la alta tasa de espera.

Un aspecto crucial del desarrollo de SmileLink ha sido la colaboración con profesionales del sector y pacientes durante el proceso de diseño y prueba. Esto no solo asegura que la aplicación cumpla con los requisitos técnicos sino también que sea intuitivo y fácil de usar por los usuarios finales, que son el público objetivo.

Desde una perspectiva tecnológica y no mera administrativa, la aplicación utiliza un sistema basado en la nube para almacenar y gestionar los datos de los pacientes, garantizando que la información de los pacientes esté disponible en cualquier momento y lugar con acceso a internet. Además, según un estudio realizado por la spin-off de la UOC Open Evidence, RAND Europe, y BDI Research [4], el intercambio de información sobre salud ha aumentado y posiciona a España como uno de los cinco países donde mejor está funcionando y se está migrando al uso de la era digital, donde según las encuestas realizadas a 5,793 médicos de 27 países de la Unión Europea, el 36 % se muestran realistas ante la llegada de estas tecnologías, seguidos de un 27 % que se muestran entusiastas, consiguiendo más del 70 % junto con los que sienten indiferencia (23 %).

En conclusión, SmileLink representa una innovación e impulso para la gestión de datos médicos, con beneficios tanto para los pacientes como para el sistema sanitario en general.

2

Objetivos

En este capítulo se abordarán los objetivos genrales y específicos sobre el proyecto.

2.1. Objetivos Generales

El objetivo general de este proyecto es desarrollar y validar una aplicación que permita a los pacientes gestionar de manera eficiente sus citas y documentos médicos, enfocándonos especialmente en el sector dental, con la posibilidad de expandirse a otras áreas médicas en el futuro mediante pequeños cambios. El alcance del proyecto incluye:

1. **Diseño de la Interfaz de usuario (UI):** Creación de una interfaz intuitiva, atractiva y fácil de usar que permita a los pacientes navegar y acceder a sus datos sin complicaciones.

2. **Implementación de funcionalidades clave:** Desarrollo de funcionalidades para la gestión de citas, intervenciones, usuarios, entidades y documentos.

3. **Seguridad:** Implementación de un sistema de autenticación y autorización para controlar los permisos.

2.2. Objetivos Específicos

2.2.1. Gestión de usuarios

Se ha de desarrollar funcionalidades de inicio de sesión, gestión de dado de alta y gestión de perfiles, diferenciando roles de usuario (administrador, doctor y paciente).

Además, se debe implementar un sistema de autenticación y autorización basado en JWT para asegurar que solo los usuarios autorizados puedan acceder a determinadas funcionalidades.

2.2.2. Gestión de citas

Crear una sección donde los pacientes puedan pedir, modificar y cancelar sus citas de manera sencilla.

En el caso de los doctores, estos poseen un panel donde pueden visualizar todas las citas de la clínica, además de un dashboard [1] donde se va actualizando con los datos de la clínica y los futuros pacientes en un corto periodo de tiempo.

2.2.3. Gestión de intervenciones

Desarrollar una funcionalidad para añadir intervenciones a citas previamente creadas, con la posibilidad de incluir documentos para su posterior almacenamiento.

Integrar la posibilidad de generar un informe en tiempo real que incluya los detalles de la intervención sobre el paciente.

2.2.4. Gestión de documentos

Desarrollar una sección donde los pacientes pueden acceder a todo su historial médico, así como los documentos proporcionados por el doctor en relación a las intervenciones realizadas.

[1] Pantalla principal/Panel

2.2.5. Interfaz de usuario

Hacer uso de un diseño y desarrollar la interfaz para que sea intuitiva y responsive, proporcionando una experiencia de usuario (UX)[2] fluida en dispositivos móviles y de escritorio.

Para ello se utilizarán los framework de Angular para el frontend y Spring Boot para el backend, siguiendo el patrón de diseño MVC.[3]

2.2.6. Bases de datos

Utilización de bases de datos relacionales como es MySQL en el despligue de la aplicación en producción, y haciendo uso de H2, para el desarrollo y/o testing.

2.2.7. Contenerización

Desarrollo de un pipeline para contenerizar la aplicación para su posterior despliegue en producción mediante una tecnología de contenedores como es Docker.

2.2.8. Pruebas y Despliegue continuo

Desarrollo de pruebas automatizadas para asegurar la calidad del código, incluyendo pruebas de integración y End-to-End.

Configuración de un pipeline de integración continua, entrega continua y despliegue continuo (CI/CD), utilizando el soporte de GitHub Actions, para automatizar el proceso de construcción, *testeo*, y despliegue de la aplicación.

2.2.9. Despliegue en la nube

Despliegue de la aplicación en producción en la nube, utilizando AWS EC2, asegurando la disponibilidad y el manejo del tráfico.

[2]UX: User Experience
[3]MVC: Model, View, Controller

2.3. Resultados esperados

Al finalizar el proyecto, se espera haber desarrollado una aplicación web completa y funcional que cumpla con los objetivos planteados, proporcionando una herramienta eficaz para la gestión de citas, intervenciones y documentos médicos, lista para su despliegue en producción y con la capacidad de ser utilizada de manera activa en el sector dental.

3

Tecnologías, Herramientas y Metodologías

En este capítulo se describen las diversas tecnologías, herramientas y metodologías empleadas para el desarrollo del proyecto.

3.1. Lenguajes

En esta sección se van a comentar los lenguajes utilizados para las diferentes partes de la aplicación.

3.1.1. Frontend

En cuanto al desarrollo del Frontend se ha utilizado TypeScript (Figura 3.1) [5] que es un lenguaje que extiende a JavaScript, el cual destaca por el uso del tipado estático que es de mucha utilidad dado que permite definir los tipos explícitos de las variables y funciones, haciendo que en caso de error de ejecución, entre una variable que no es la deseada, salte la excepción. Además este lenguaje es altamente integrable con Angular, framework del que hablaremos en la próxima sección.

Figura 3.1: Logotipo de TypeScript

Para el desarrollo de la interfaz se ha utilizado HTML5 y SCSS con el fin de crear una interacción de páginas y estilos favorable y óptima para los navegadores actuales, además de la capacidad de integración con diferentes herramientas como Angular, del que posteriormente hablaremos.

3.1.2. Backend

Para el desarrollo del Backend se ha utilizado Java (Figura 3.2) [6], un lenguaje de programación multiplataforma orientado a objetos conocido y utilizado por millones de personas. Java es un lenguaje robusto y eficaz que posee múltiples librerías y frameworks como es el caso de Spring, utilizado para el desarrollo de nuestra aplicación web.

Figura 3.2: Logotipo de Java

3.1.3. Compilación de la aplicación en la nube

Para el correcto despliegue en la nube de AWS se ha creado un script de shell interno en Bash (Bourne Again Shell) (Figura 3.3) [7]. Bash es un lenguaje de scripting de Unix que se usa comúnmente en sistemas operativos basados en Unix y Linux para automatizar tareas y escribir scripts.

Figura 3.3: Logotipo de Bash

3.2. Tecnologías

3.2.1. Frontend

En la parte del Frontend se ha utilizado en conjunto con TypeScript, Angular(Figura 3.4c) [8], que es un framework de desarrollo de aplicaciones web de código abierto ampliamente utilizado por la comunidad gracias a la libertad de uso en la construcción de aplicaciones web dinámicas y SPA (Single Page Application), eficientes y escalables, de forma rápida y sencilla.

Una SPA (Single Page Application)[9] cada vez está más en uso. Se trata de un tipo de aplicación web donde todas las pantallas (vistas) se muestran en la misma página. Aunque nosotros pensemos que estamos navegando por diferentes páginas, en realidad no nos hemos movido del punto de entrada que generalmente se relaciona con el archivo *index.html*. De esta forma lo que se hace es añadir y quitar componentes dependiendo de la acción, pero realmente no nos estamos moviendo de la pagina inicial haciendo que no se tenga que recargar la página. Con la implementación de este tipo se mejora la eficiencia de la comunicación entre el cliente y el servidor, dado que los datos pesan muy poco a pesar de que los archivos HTML y SCSS sean extensos, produciendo que la experiencia del cliente sea muy agradable debido a su baja tasa de latencia.

Gracias al uso de Angular se han utilizado ciertas librerías como Angular Material que nos proporciona ciertos componentes como *Form Control, Buttons o Navigation* facilitando la integración en el desarrollo web. Además, para la realización de popups de alerta y confirmación se ha añadido la librería sweetalert2 (Figura 3.4a) [10] en la versión 11.10.1 y bootstrap (Figura 3.4b) [11] en la versión 5.3.2.

(a) SweetAlert (b) Bootstrap (c) Angular

Figura 3.4: Logotipo de SweetAlert, Bootstrap y Angular

3.2.2. Backend

En cuanto a las tecnologías propias para desarrollar el Backend, se ha optado por usar Spring Boot con la versión 2.7.18, haciendo pleno uso del framework de Spring.

Spring (Figura 3.5a)[12] es un framework orientado a la programación de aplicaciones empresariales desarrolladas bajo el lenguaje de programación Java. Spring proporciona un soporte increíble en el desarrollo dado que ofrece una gran variedad de servicios que pueden ser utilizados de forma libre gracias a su licencia de software libre. Gracias a Spring se ha facilitado el desarrollo de servicios REST, la integración con la base de datos, securización, realización de pruebas, integración de envíos de correos electrónicos y el patrón de diseño MVC, entre otros.

La integración con la base de datos se ha realizado utilizando Spring Data que nos facilitaba el uso de consultas a la base de datos relacional mediante repositorios heredando de las acciones de JPA Repository. Así como la securización mediante el uso de Spring Security y, la librería MailSender para el envío de correos electrónicos con credenciales de inicio de sesión o notificaciones en caso de olvido de la contraseña.

La base de datos se ha implementado de forma diferente en dos entornos. En el entorno de producción se ha hecho uso de MySQL (Figura 3.5b) que tiene un largo recorrido en las bases relacionales, y en el entorno de desarrollo y testing se ha utilizado H2 Database (Figura 3.5c) [13], una base de datos ultraligera basada en Java que permite correr en memoria, permitiéndonos realizar consultas y construir pruebas que requieran acceso a datos de una forma sencilla y muy rápida.

(a) Spring Boot (b) MySQL (c) H2

Figura 3.5: Logotipo de Spring Boot, MySQL y H2

3.2.3. Informes sobre intervenciones

Para la generación de archivos tanto para el cliente, como para hacer uso de ellos los doctores, se han utilizado varias librerías.

Dentro del Backend se ha utilizado para la generación de PDFs la librería de iText[14], con la cual se pueden generar informes médicos, especificando el estilo que se quiere de manera sencilla. De otra forma, se ha posibilitado la extracción de los clientes y citas que posee una clínica mediante dos formas: Excel (xlsx) o PDF. Para llevar a cabo este proceso se ha utilizado Apache POI[15], y Lowagie[16]. Apache POI es una librería que permite y faclita la extración de datos y escritura en celdas de excel. De la misma manera pero en formato PDF actúa Lowagie, donde se ha creado una tabla en el que se han introducido los campos deseados.

Para la visualización de los informes generados o documentos añadidos de forma externa se ha implementado un visualizador integrado en el Frontend llamado PDFViewerModule[17] en la versión 10.0.0, cuya funcionalidad es la de renderizar el documento de forma integrada en la aplicación sin tener que hacer uso de aplicaciones externas.

3.2.4. Pruebas

Para esta sección se ha utilizado JUnit (Figura 3.6c) [18] con la versión 5, que mejora su antigua versión 4 en la optimización del uso de Java, se han agregado nuevas características para describir, ordenar, organizar y ejecutar pruebas y además, está organizado en bibliotecas, haciendo que se puedan importar solo las funciones que se necesiten.

Gracias a este framework podemos utilizar acciones como son @AfterEach, @BeforeEach, @Order, @Test y @BeforeAll pare especificar en los métodos la acción que se quiere realizar. Los Each hacen que se ejecute ese método por cada test, uno antes y otro después, en cambio, @BeforeAll se ejecuta una única vez cuando compila la clase completa. Por otro lado, se pueden hacer aserciones para verificar que se cumple el dominio requerido, por ejemplo, comprobar que el

contenido devuelto por una petición API es el que se requiere.

Para la realización de los test de integración se ha empleado el framework de Rest Assured (Figura 3.6b) [19], que utiliza JUnit para realizar pruebas de API REST. Gracias a esta librería se pueden ejecutar consultas a la API de forma rápida, ya sea para obtener, insertar, actualizar o eliminar datos, podremos comprobar que realmente se hace lo que se pide. Además, esta librería permite realizar aserciones para verificar el comportamiento de la consulta. En el momento en que la prueba falle, se podrá identificar el error que se ha producido de manera rápida.

Para las pruebas End-to-End se ha empleado el framework de Selenium (Figura 3.6a) [20] con el driver del navegador Chrome. Con este framework podremos realizar las acciones que el usuario realizaría de manera común en el navegador web y comprobar que las interacciones entre componentes son correctas.

(a) Selenium (b) Rest Assured (c) JUnit 5

Figura 3.6: Logotipo de Selenium, Rest Assured y JUnit 5

3.2.5. Contenerización

Se ha usado Docker (Figura 3.7) [21] como tecnología de contenerización. Docker es una plataforma que permite empaquetar, distribuir y ejecutar aplicaciones en contenedores. Los contenedores Docker encapsulan una aplicación y sus dependencias en un solo paquete que se puede ejecutar en cualquier entorno, garantizando la consistencia, portabilidad, prueba y despliegue.

Un contenedor es una unidad que empaqueta el código de la aplicación junto con todas sus dependencias, de manera que la aplicación pueda ejecutarse en cualquier entorno de la misma manera. Se podría ejecutar de igual forma sobre una maquina virtual (VM) pero lo que ocurre es que este sistema es mucho más pesado y lento. Además, cabe destacar que los contenedores se ejecutan en el núcleo del sistema operativo del host y están aislados del resto de aplicaciones. Solo incluyen la aplicación y sus dependencias, haciéndolos sumamente rápidos.

El uso del contenedor es el siguiente: Se crea una imagen Docker de la aplicación, que incluye la aplicación y todas sus dependencias. Esta imagen se sube a Docker Hub, repositorio de imágenes de Docker y desde entonces se puede usar la misma imagen en diferentes entornos que se ejecutará de la misma manera,

evitando los famosos problemas de que en una máquina funcione y en otra no.

Figura 3.7: Logotipo de Docker

3.3. Herramientas

En esta sección se hablará de las herramientas utilizadas para el desarrollo del proyecto.

3.3.1. Control de versiones y repositorio

Se ha empleado el software de control de versiones Git, además de GitHub (Figura 3.8), que ha sido utilizado como repositorio[1] en remoto accesible desde cualquier parte del mundo y como ejecutor de automatizaciones con GitHub Actions.

GitHub Actions[22] es una plataforma de integración y despliegue continuo que permite automarizar la compilación, ejecución de pruebas y despliegues creando flujos de trabajo (workflows). Además, GitHub proporciona las maquinas virtuales de Linux, Windows y MacOS para que se ejecuten los flujos de trabajo creados.

Git tiene como propósito registrar las modificaciones en los archivos y facilitar la coordinación del trabajo que varias personas realizan en archivos compartidos dentro de un repositorio de código.

[1] `https://github.com/scuadrosf/TFG-Software.git`

Figura 3.8: Logotipo de Git y GitHub

3.3.2. Construcción de la aplicación

Para la construcción de la aplicación se ha hecho uso de la herramienta Maven (Figura 3.9) [23].

Maven es una herramienta de gestión que facilita la contrucción del proyecto en Java y el manejo de dependencias. Además, se pueden automatizar ciertas tareas como la ejecución de pruebas. Todas las librerías junto con sus versiones se incluyen en un archivo llamado *pom.xml* asegurando que todas las dependencias se incluyan automáticamente y se descarguen desde repositorios centralizados.

Figura 3.9: Logotipo de Maven

3.3.3. Visualización de base de datos

En relación a la visualización y gestión de la base de datos se ha utilizado MySQL Workbench (Figura 3.10) [24]. Workbench es una herramienta de diseño, desarrollo y administración de bases de datos que proporciona una interfaz para trabajar con MySQL, permitiendo crear diagramas, esquemas, ejecutar scripts SQL, consultas, etc.

Figura 3.10: Logotipo de MySQL Workbench

3.3.4. Peticiones API

Para las pruebas y documentación de las APIs se ha utilizado la herramienta Postman (Figura 3.11) [25]. Esta herramienta es ampliamente utilizada por desarrolladores y equipos de software para facilitar la creación de APIs. Gracias a Postman se pueden crear y enviar peticiones HTTP (GET, POST, PUT, DELETE, etc) a APIs mediante una interfaz amigable y sencilla que facilita la configuración de parámetros, headers y cuerpos de las peticiones.

Figura 3.11: Logotipo de Postman

3.3.5. Entorno de desarrollo integrado (IDE)

En este proyecto, Visual Studio Code se utiliza como el editor de código por defecto para desarrollar tanto el frontend como el backend de la aplicación. Además, su integración con Git facilita el manejo del control de versiones con extensiones como Git Graph, en la figura 3.12 podemos ver un ejemplo de uso en el desarrollo de SmileLink. De la misma forma que se ha integrado Git Graph, se ha hecho uso de diferentes extensiones como el paquete de Java o Angular para facilitar su uso y autocompletado de sintaxis, haciendo su desarrollo más rápido y preciso.

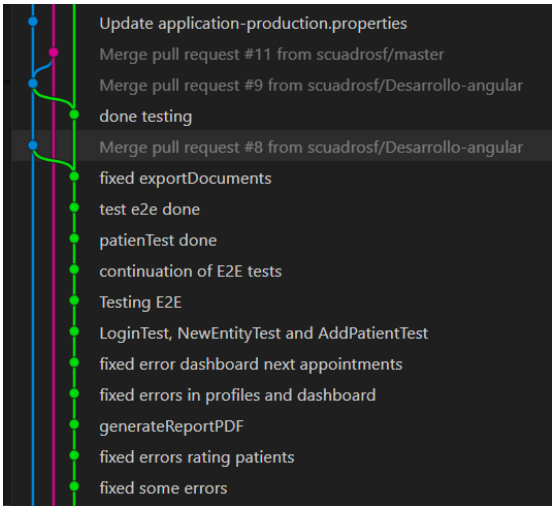


Figura 3.12: Control de flujo - Git Graph. La línea verde es el flujo de desarrollo, la azul es master y la rosa es un backup.

3.3.6. Organización y gestión del proyecto

Para llevar a cabo una organización y gestión sobre el proyecto se ha utilizado la herramienta Jira (Figura 3.13) [26].

Jira es una herramienta para la gestión de proyectos y tareas en la que se ha aplicado una metodología ágil como es Scrum, facilitando la planificación de sprints, posibilitando la asignación de tareas, y el seguimiento del progreso del proyecto. Además, Jira permite la generación de informes y dashboards para monitorizar el rendimiento en el proyecto como los burndown charts, velocity charts y cumulative flow charts.

Figura 3.13: Logotipo de Jira

3.3.7. Despliegue de la aplicación

Para el despliegue de la aplicación en el entorno de producción se ha utilizado el proveedor de cloud, Amazon Web Services (Figura 3.14a) [27], AWS en adelante.

AWS es una plataforma de servicios en la nube que proporciona herramientas para crear, implementar y gestionar aplicaciones y servicios de forma segura y eficiente. Entre los servicios que proporciona AWS, se ha hecho uso de RDS[2] (Figura 3.14b) y EC2[3] (Figura 3.14c).

RDS es un servicio que sirve para configurar, operar y escalar la base de datos en la nube. EC2, es un sistema de computación en la nube que trabaja bajo demanda, proporcionando escalabilidad y seguridad con la creación de máquinas virtuales (instancias) que pueden ser configuradas con diferentes sistemas operativos y especificaciones hardware dependiendo de la necesidad del proyecto.

La instancia de EC2 en el caso de este proyecto trabaja con el sistema operativo Ubuntu 22.04[28], una distribución de Linux basada en Debian. Esta es una de tecnologías más utilizas en el mundo del desarrollo y la administración de sistemas conocida por su estabilidad, seguridad y facilidad de uso.

Para profesionalizar el despliegue de la aplicación en producción se ha comercializado un dominio en IONOS (Figura 3.14d) [29] bajo el nombre de `smilelink.es`. IONOS es un proveedor de servicios en la nube, hosting web y certificados SSL, entre otros. De hecho, el dominio está cifrado bajo el certificado SSL proporcionado por IONOS, asegurando una mejor seguridad en el servicio.

(a) AWS

(b) RDS

(c) EC2

(d) IONOS

Figura 3.14: Logotipo de AWS, RDS, EC2 e IONOS.

[2]Relational Database Service
[3]Elastic Compute Cloud

3.4. Metodologías

Para el desarrollo del proyecto se han empleado metodologías ágiles, enfocándose en particular en Scrum. Las metodologías ágiles se centren en la entrega continua de valor al cliente mediante iteraciones cortas y revisiones frecuentes, pudiendo adaptase a cambios y mejoras necesarias y solicitadas por un supuesto stakeholder[4] primario, pero en este caso el cliente y el desarrollador es la misma persona, por tanto los cambios y mejoras son propuestas *motu proprio*.

Una metodología ágil se caracteriza por su flexibilidad y capacidad de respuesta a cambios, donde su principar prioridad es entregar software funcional aunque su documentación sea mínima. Además, para el seguimiento de este proyecto se ha empleado el modo de trabajo Scrum.

Scrum es una metodología dentro de las metodologías ágiles que se basa en sprints de una duración concreta. En el desarrollo de SmileLink la duración de los sprints han ido dependiendo de la carga de trabajo, desde dos hasta cuatro semanas, durante los cuales se ha llevado acabo una parte del proyecto.

Para gestionar este proceso iterativo se ha empleado la herramienta de Jira[5], una herramienta muy popular para la gestión de proyectos, que permite crear y gestionar tareas, historias de usuario, bugs y sprint de manera sencilla y eficiente. En la figura 3.15 podemos ver un ejemplo de gestión del tablero de Jira utilizado en SmileLink, donde se organiza en tres columnas, *To Do, In Progress* y *Done*.

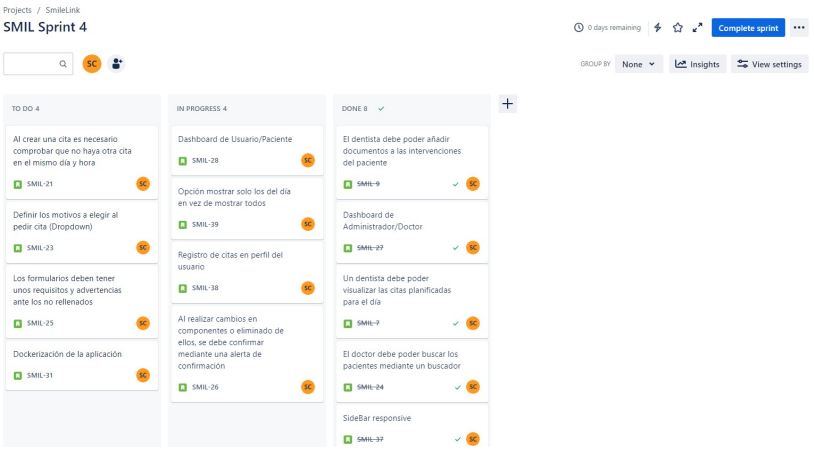


Figura 3.15: Pantalla con la gestión del tablero de Jira

Además, Jira ofrece la posibilidad de realizar reportes y métricas para evaluar

[4]Los stakeholders son aquellos individuos o grupos que tienen interés e impacto en una organización y en los resultados de sus acciones [30]

[5]`https://smilelink.atlassian.net/jira/software/projects/SMIL/list` (Link no accesible sin añadir los permisos de acceso)

el rendimiento y el progreso en el proyecto como se puede ver en la figura 3.16 en el que se muestra una gráfica *Burnup*, que muestra la comparación entre las tareas realizadas en el sprint y las totales, haciendo que esta gráfica ayude a llevar un seguimiento del progreso hasta terminar el sprint. De esta forma se puede ver que la linea roja indica el *Work Scope*, que indica el número de tareas para completar el sprint, la línea verde son las tareas que se han ido completando durante el sprint y la línea gris representa el ritmo ideal que se debe ir cumpliendo, por tanto, en este caso se puede ver que se sigue un ritmo razonable y acorde al ideal.

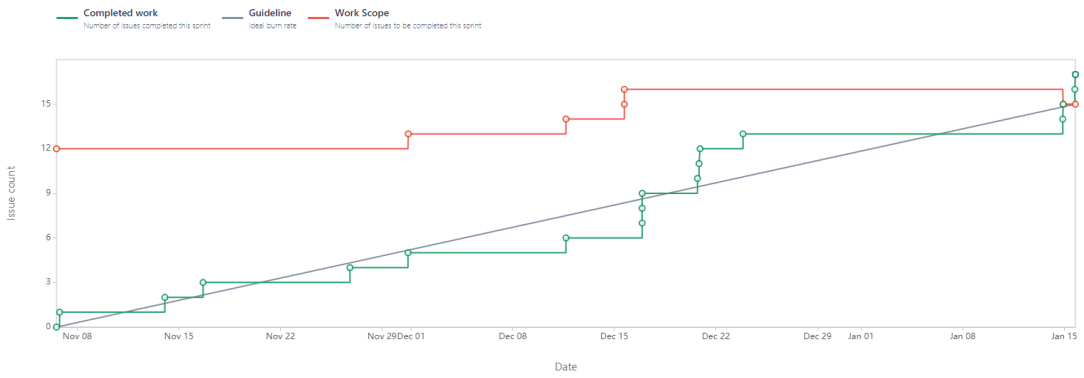


Figura 3.16: Reporte Burnup Jira Sprint 4

De la misma forma se tiene el diagrama *Burndown*. Esta gráfica que se puede ver en la figura 3.17 representa el trabajo restante para acabar el sprint, donde de igual forma que el anterior, la línea gris representa la escena ideal en la que equipo tiene que trabajar.

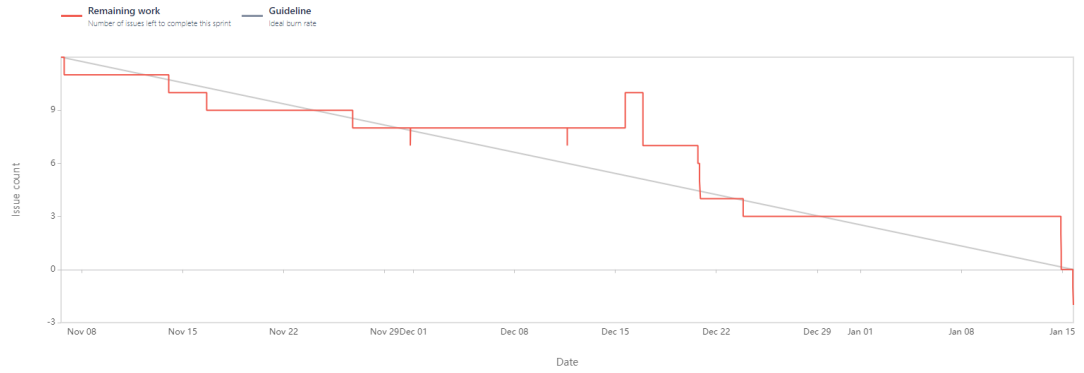


Figura 3.17: Reporte Burndown Jira Sprint 4

Por último, cabe destacar la gráfica *Cumulative Flow Diagram* que se puede ver en la figura 3.18, que representa el estado del proyecto en relación a las tareas a lo largo del tiempo, mostrando su evolución en las columnas del tablero. De esta gráfica se puede analizar un incremento constante del trabajo completado, pues

la sección verde que representa las tareas *Done*, muestra un incremento constante en el tiempo, lo que da indicios que se ha mantenido un ritmo constante de finalización de tareas, lo cual es positivo. Por otro lado, la sección azul representa las tareas en progreso (*In Progress*), que muestra ciertas variaciones pero se mantiene relativamente estable en el tiempo, lo que quiere decir que se ha hecho hincapié en evitar la excesiva carga de trabajo en curso. La sección morada indica las tareas pendientes por empezar, es decir, la columna *To Do*, donde se muestra también un incremento gradual, lo que viene a indicar que se ha ido aumentando el Backlog[6], lo que es normal, pero hay que gestionar con detenimiento para evitar futuras sobrecargas de tareas.

Hay que ser crítico, por tanto, cabe destacar que hay varias ocasiones donde se observan aumentos significativos en las tareas en progreso seguidos de incrementos en las tareas completadas, lo que puede indicar periodos de mayor actividad o sin movimiento en el tablero, por lo que habría que estudiar estos picos de actividad para futuros casos.

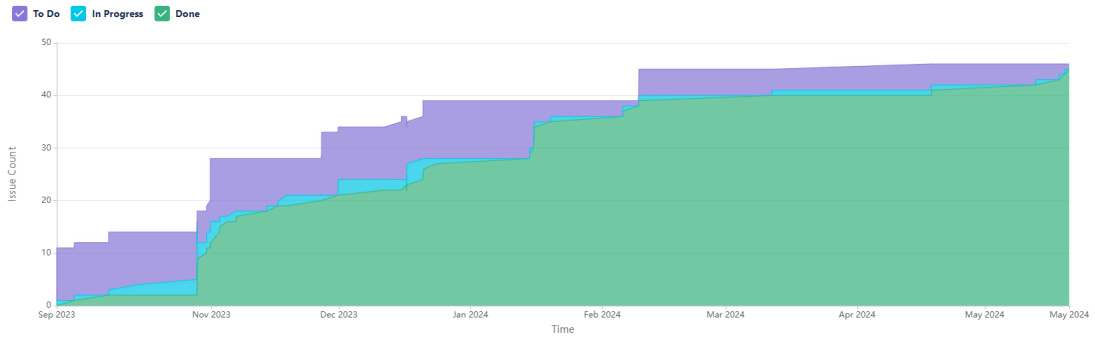


Figura 3.18: Reporte Cumulative Flow Diagram

[6]Lista de tareas ordenadas por prioridad para su uso en futuros sprints

4

Descripción informática

En este capítulo se describe el proyecto a alto nivel, desde un punto de vista más técnico en el que se hablará sobre los requisitos, la arquitectura, el diseño, alguna implementación que haya sido interesante, y por último, las pruebas realizadas.

4.1. Requisitos

En esta sección se van a desarrollar los requisitos a nivel de historias de usuario que el sistema debe ofrecer. En primer lugar, se contempla una serie de historias de usuario "no funcionales", en el apartado 4.1.2, y, seguidamente unas historias de usuario "funcionales", a nivel de los diferentes roles, en los apartados 4.1.1 (Administrador Total), 4.1.1 (Administrador doctor), 4.1.1 (Doctor), 4.1.1 (Paciente).

4.1.1. Funcionales

Rol Administrador Total (ADMIN)

1. Yo como Administrador Total, quiero poder acceder a mi perfil mediante mis credenciales.
2. Yo como Administrador Total, quiero poder dar de alta entidades para

gestionar la clínica de manera eficiente

3. Yo como Administrador Total, quiero poder recuperar mi contraseña para en caso de olvido poder recuperar mi cuenta.

4. Yo como Administrador Total, quiero pode cambiar mi contraseña en cualquier momento para añadir más seguridad a mi cuenta.

Rol Doctor Administrador (ADMIN, DOCTOR)

1. Yo como doctor administrador, quiero poder acceder a mi perfil mediante mis credenciales.

2. Yo como doctor administrador, quiero tener unos tiempos definidos para las citas para poder reservar correctamente ese tiempo.

3. Yo como doctor administrador, quiero poder dar de alta a doctores en mi entidad para que puedan gestionar sus pacientes.

4. Yo como doctor administrador, quiero no poder tener dos citas en el mismo tramo horario.

5. Yo como doctor administrador, quiero poder ver a mis pacientes para llevar un seguimiento adecuado de sus casos.

6. Yo como doctor administrador, quiero poder ver los pacientes de toda mi entidad para colaborar con otros doctores.

7. Yo como doctor administrador, quiero poder dar de alta a nuevos pacientes proporcionando sus datos personales para que hagan uso del sistema.

8. Yo como doctor administrador, quiero poder realizar acciones CRUD sobre los pacientes para mantener su información actualizada.

9. Yo como doctor administrador, quiero poder realizar acciones CRUD sobre las citas para gestionar el calendario de mis pacientes.

10. Yo como doctor administrador, quiero poder realizar acciones CRUD sobre las intervenciones para gestionar las intervenciones mis pacientes.

11. Yo como doctor administrador, quiero poder generar informes de las citas y adjuntarlas como intervenciones para documentar el tratamiento de los pacientes.

12. Yo como doctor administrador, quiero poder generar un documento PDF y Excel de los pacientes asignados a mí para tener un registro detallado de ellos.

13. Yo como doctor administrador, quiero poder generar un documento PDF de las citas en mi entidad para tener un registro consolidado de todas las citas.

14. Yo como doctor administrador, quiero poder utilizar un buscador para encontrar a los pacientes por nombre, apellido o DNI para acceder rápidamente a su información.

15. Yo como doctor administrador, quiero poder recuperar mi contraseña para en caso de olvido poder recuperar mi cuenta.

16. Yo como doctor administrador, quiero pode cambiar mi contraseña en cualquier momento para añadir más seguridad a mi cuenta.

Rol Doctor (DOCTOR)

1. Yo como doctor, quiero poder acceder a mi perfil mediante mis credenciales.

2. Yo como doctor, quiero poder ver mis pacientes para llevar un seguimiento adecuado de sus casos.

3. Yo como doctor, quiero no poder tener dos citas en el mismo tramo horario.

4. Yo como doctor, quiero tener unos tiempos definidos para las citas para poder reservar correctamente ese tiempo.

5. Yo como doctor, quiero poder ver los pacientes de toda mi entidad para colaborar con otros doctores.

6. Yo como doctor, quiero poder dar de alta a nuevos pacientes proporcionando sus datos personales para que hagan uso del sistema.

7. Yo como doctor, quiero poder realizar acciones CRUD sobre los pacientes para mantener su información actualizada.

8. Yo como doctor, quiero poder realizar acciones CRUD sobre las citas para gestionar el calendario de mis pacientes.

9. Yo como doctor, quiero poder realizar acciones CRUD sobre las intervenciones para gestionar las intervenciones mis pacientes.

10. Yo como doctor, quiero poder generar informes de las citas y adjuntarlas como intervenciones para documentar el tratamiento de los pacientes.

11. Yo como doctor, quiero poder generar un documento PDF y Excel de los pacientes asignados a mí para tener un registro detallado.

12. Yo como doctor, quiero poder generar un documento PDF de las citas en mi entidad para tener un registro consolidado de todas las citas.

13. Yo como doctor, quiero poder utilizar un buscador para encontrar a los pacientes por nombre, apellido o DNI para acceder rápidamente a su información.

14. Yo como doctor, quiero poder recuperar mi contraseña para en caso de olvido poder recuperar mi cuenta.

15. Yo como doctor, quiero pode cambiar mi contraseña en cualquier momento para añadir más seguridad a mi cuenta.

Rol Paciente (USER)

1. Yo como paciente, quiero poder acceder a mi perfil mediante mis credenciales.

2. Yo como paciente, quiero poder que se me asigne automáticamente el doctor que me da de alta para saber quién me atenderá.

3. Yo como paciente, quiero poder ver todas mis intervenciones para llevar un control de mi historial médico.

4. Yo como paciente, quiero poder ver todo mi historial médico para estar informado sobre mi salud.

5. Yo como paciente, quiero poder realizar acciones CRUD sobre mis citas para gestionarlas de acuerdo a mis necesidades.

6. Yo como paciente, quiero poder actualizar mi perfil para mantener mi información personal al día.

7. Yo como paciente, quiero poder recibir un correo electrónico con mis credenciales.

8. Yo como paciente, quiero poder recibir un correo electrónico con nuevas credenciales en caso de que las haya olvidado.

9. Yo como paciente, quiero poder recuperar mi contraseña para en caso de olvido poder recuperar mi cuenta.

10. Yo como paciente, quiero pode cambiar mi contraseña en cualquier momento para añadir más seguridad a mi cuenta.

11. Yo como paciente, quiero que quien me pueda dar de alta sea únicamente un trabajador de la entidad a donde voy.

4.1.2. No funcionales

1. El sistema debe poder tener diferentes roles de usuario para poder hacer diferentes acciones dependiendo del rol.

2. El sistema debe tener una fácil usabilidad, siendo intuitiva y fácil de usar.

3. El sistema debe desarrollarse en Spring Boot y Angular.

4. El sistema debe poder tener un sistema de autenticación y autorización basada en JWT.

5. El sistema debe poder almacenar los datos en una base de datos relacional.

6. El sistema debe ser capaz de enviar correos electrónicos.

7. El sistema debe tener una herramienta de confirmación ante acciones decisivas.

8. El sistema debe ser accesible desde dispositivos de escritorio y móviles.

4.2. Arquitectura y Análisis

En esta sección se describirá la arquitectura de la aplicación, entrando en detalle en las diferentes partes que forman el proyecto.

4.2.1. Arquitectura de la aplicación

En la figura 4.1 podemos ver un diagrama en el que se describen las partes fundamentales a nivel general del proyecto que son: frontend, backend y la base de datos, desplegadas en un servicio en la nube de AWS EC2, y utilizando tecnologías como Ionos para la asignación IP de la máquina virtual.

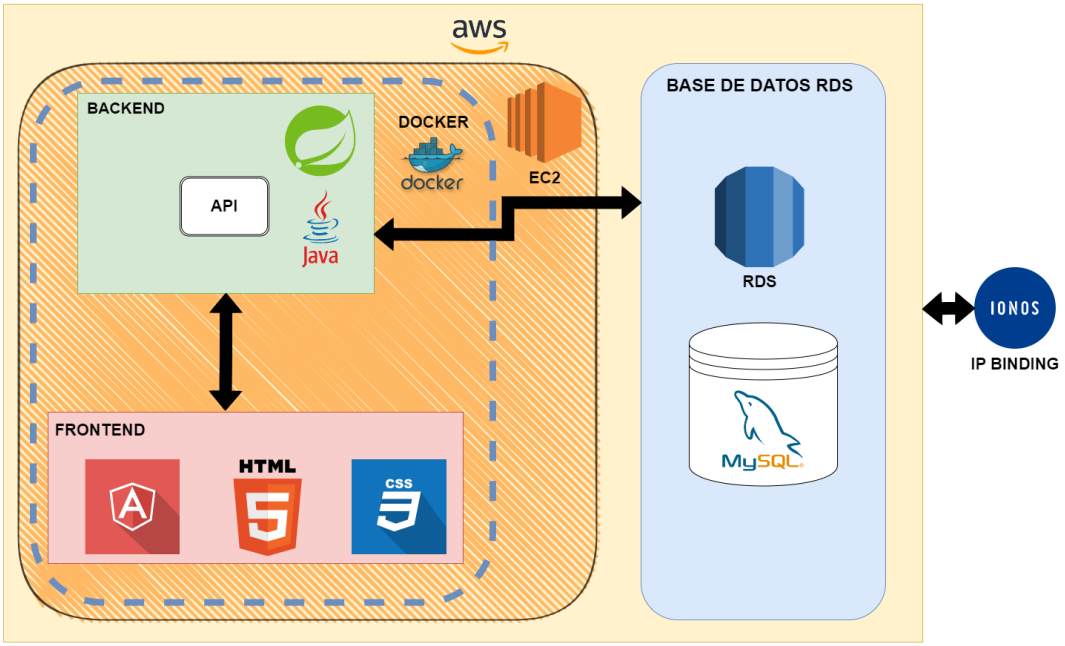


Figura 4.1: Arquitectura general

Componentes de la arquitectura

El frontend de la aplicación se ha desarrollado utilizando Angular, HTML5 y SCSS, en donde se representan las pantallas en una página única gracias al hacer uso del SPA. Este apartado en la arquitectura es de alta importancia dado que es donde el usuario va a interaccionar, por tanto, las transacciones de los componentes del frontend con el backend deben tener un funcionamiento correcto. Como se puede ver, el frontend únicamente se comunica con el backend, donde se dispondrá de una API REST, donde se irá haciendo solicitudes para ejecutar las acciones que se deseen. Estos dos componentes se encontrarán contenerizados mediante Docker en una imagen específica. Además, la API se comunicará con la base de datos MySQL soportada en la nube gracias a AWS RDS, una instancia específica de AWS, haciendo que los datos de la aplicación persistan en el tiempo. Todo estos procesos se mantienen contenidos en una instancia de una maquina virtual de AWS llamada EC2 bajo el sistema operativo de Ubuntu. A su vez, gracias al dominio contratado con el servicio de hosting de Ionos, se le asigna la dirección IP de la instancia EC2 al host para poder utilizar el dominio en uso. (`www.smilelink.es`).

4.2.2. Frontend

En el caso del frontend se sigue siguiendo el patrón MVC, donde en este caso los controladores son los componentes de la aplicación. Siguiendo una jerarquía de procesos, se empezará a detallar en relación de Componentes - Modelos - Servicios.

Los componentes se han estructurado por "vistas", de esta forma obtenemos la estructura que se puede ver en la figura 4.2.

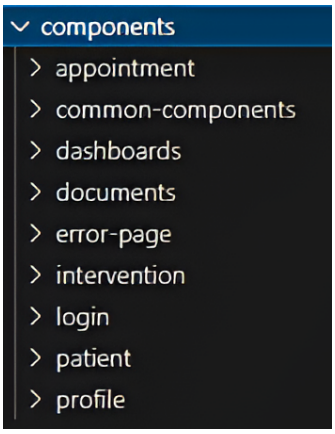


Figura 4.2: Estructura de componentes Angular

Como se observa se ha divido en diferentes pantallas haciendo relación al objetivo de la aplicación que es:

- Gestionar citas (appointment):
  - add-appointment: Contiene la pantalla con el formulario a rellenar para generar una cita.
  - add-appointment-main: Contiene la pantalla con los pacientes para dar una cita.
  - edit-appointment: Contiene la pantalla con la cita solicitada dispuesta a rellenar de nuevo con los cambios pertinentes.
  - appointment: Contiene la pantalla con la lista de citas pendientes.
- Gestionar intervenciones (*intervention*):
  - add-intervention: Contiene la pantalla con el formulario a rellenar tras la cita, donde se puede adjuntar un documento y escribir aclaraciones.
  - edit-intervention: Contiene la pantalla con la intervención que se quiere actualizar.

    - interventoin: Contiene una pantalla con un registro de todas las citas pendientes de un paciente para poder añadir intervención a la cita pertinente.

- Gestionar documentos (*documents*):

    - documents-of-intervention: Contiene una pantalla con un visor de PDF integrado en el que se muestra el archivo adjunto en el caso en que lo haya, además de las anotaciones incorporadas en la intervención.

- Gestionar pacientes (*patient*):

    - add-patient: Contiene una pantalla con el formulario a rellenar para dar de alta a un paciente.
    - edit-patient: Contiene una pantalla con el formulario del paciente que se quiere actualizar.
    - patient: Contiene una pantalla con una lista de los pacientes, divididos entre los de toda la entidad y los del doctor que ha iniciado sesión mediante un toggleButton[1].

No obstante, para un uso más completo se han añadido vistas como son el panel principal (*dashboards*), una pantalla para gestionar tu perfil (*profile*) o iniciar sesion (*login*), así como componentes compartidos (*common-components*) que contienen los headers y sidebar correspondientes para la navegabilidad en la aplicación.

En cuanto a los servicios gestionados por el frontend, son simples comunicaciones con el backend mediante el módulo *HttpClient* de Angular.

4.2.3. Backend

Organización

De la misma forma que en el Frontend, se sigue un patrón de diseño MVC, formado por *@RestController*, *@Entity*, *@Service*, además de una carpeta para gestionar la seguridad de la aplicación, y los permisos para hacer peticiones a los servicios REST.

De la misma forma que la seguridad, se ha creado una carpeta para con el contenido que favorece la gestión del acceso y almacenamiento de los datos en la base de datos con la ayuda del framework de Spring Data, *JpaRepository*. En la figura 4.3 se puede ver la organización del desarrollo de este apartado.

[1]Botón que cambia de comportamiento al hacer click sobre él

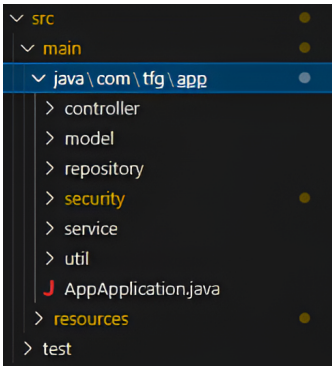


Figura 4.3: Estructura Backend

Para entender mejor las relaciones que existen entre las diferentes clases en la figura 4.4 se pueden analizar las diferentes clases que compone el modelo, donde se ve una relación coherente entre ellas de una forma natural. Como se puede observar, sin el usuario no se puede tener ni citas, ni intervenciones. Además, sin una cita no puede haber una intervención, y por ende, un documento.

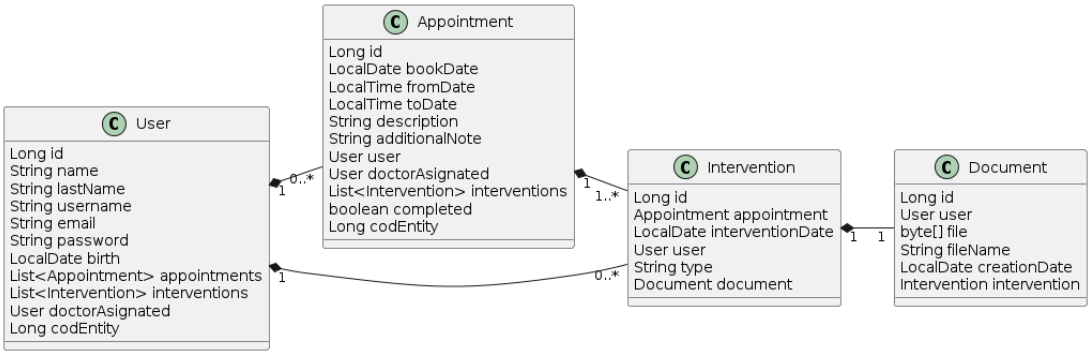


Figura 4.4: Diagrama de Clases

Para poder mostrar, añadir, modificar o eliminar los datos en la parte del frontend se han creado los siguientes controladores:

- **AppointmentRestController:** Controlador encargado de gestionar las citas y las descripciones posibles de las citas.
- **DocumentRestController:** Controlador encargado de gestionar los documentos de los pacientes.
- **EmailRestController:** Controlador encargado de los envíos de correos electrónicos a los pacientes.
- **InterventionRestController:** Controlador encargado de gestionar las intervenciones de los pacientes y recuperar el documento adherido a esa intervención.

- **UserRestController:** Controlador encargado de gestionar los usuarios.
- **UtilRestController:** Controlador encargado de generar documentos con información, y actualizar los de datos estadísticos como el número total de pacientes o el numero de citas restantes y completadas.
- **SPAContoller:** Redirecciona el contenido para que se muestre sobre la ruta, " /".

Los servicios además de promover el principio DRY (*Don't Repeat Yourself*), en el que se trata de evitar la duplicación de código, juegan el papel de transmisores de la información al *Repository*, lugar en el que se ejecuta la acción frente a la base de datos.

Representación de acciones posibles y su flujo

En relación a la forma de realizar una acción determinada como puede ser pedir una cita se puede ver en la figura 4.5 un diagrama de secuencia de esta acción, donde el usuario lo que hace es iniciar sesión, puesto que sin este requisito no podrá hacer nada, seguidamente una vez autenticado el usuario procede a pedir una cita en la plataforma, que se comunicará con el controlador de las citas, y comprobará que no existe otra en el mismo tramo horario, si está disponible se seguirá el proceso hasta guardar la solicitud mediante un POST en la base de datos.

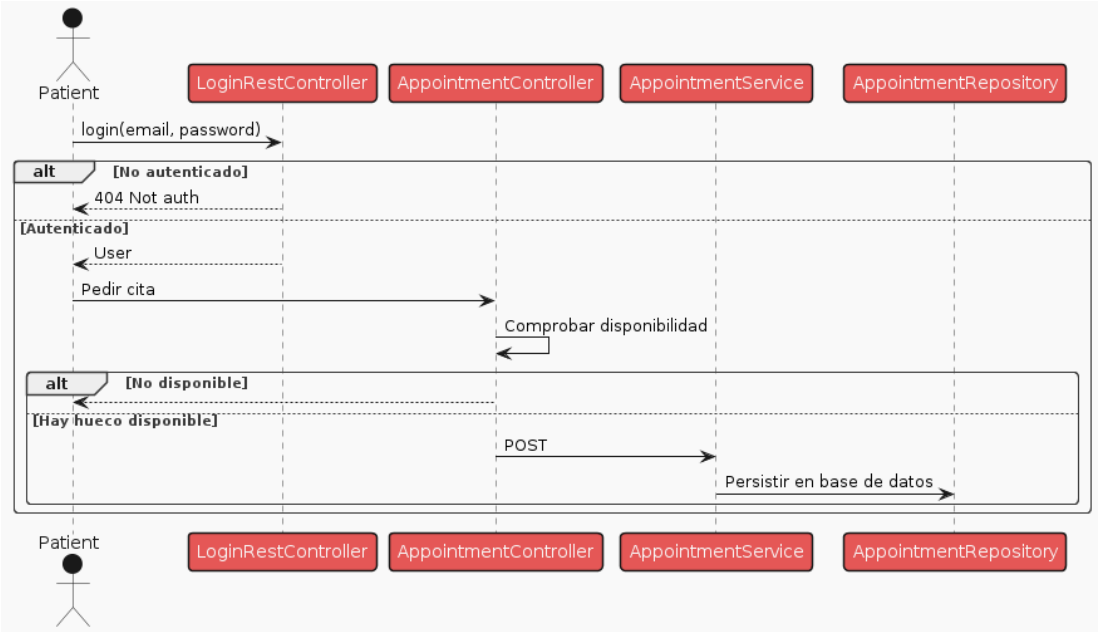


Figura 4.5: Diagrama de secuencia de pedir una cita por un paciente

En el caso en que un doctor requiera de añadir una intervención a una cita el diagrama de secuencia sería el de la figura 4.6, donde se ha planteado desde otro de punto de vista, incluyendo el perfil del frontend e indagando a más detalle de elementos y control. En este caso el doctor primero deberá visualizar las citas que están pendientes por revisar, una vez decida que cita quiere atender, tiene en su poder la decisión de generar un informe en formato PDF o no, en el caso en que no lo quiera no debería hacer otra acción más que rellenar el formulario con la intervención realizada, así como añadir un documento adjunto a la intervención si lo deseara.

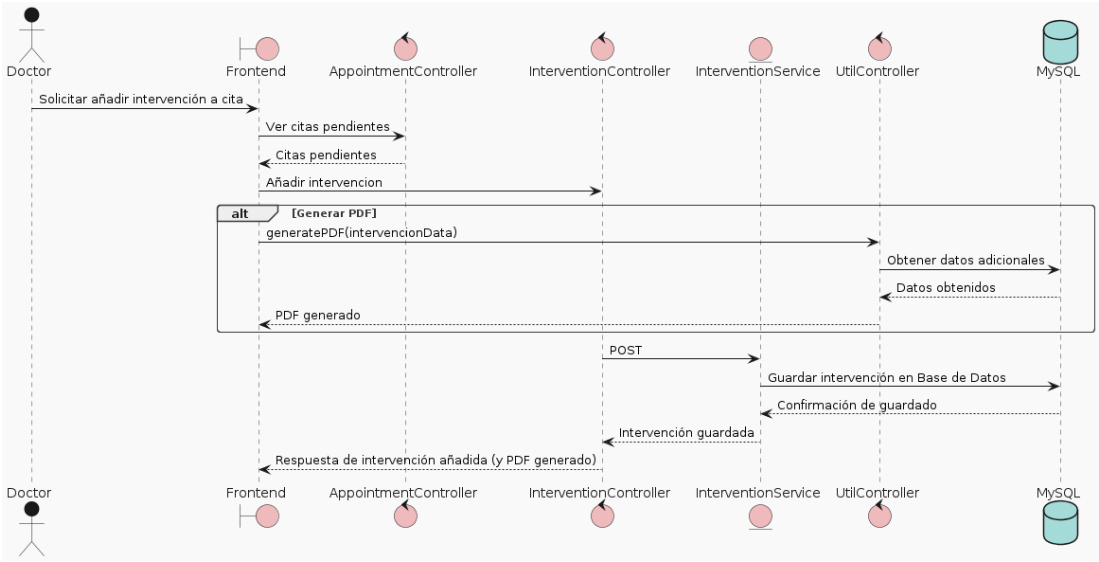


Figura 4.6: Diagrama de secuencia añadir intervención

4.2.4. Base de datos

La base de datos se ha implementado utilizando MySQL, un sistema de gestión de bases de datos relacional.

A continuación, se describen las tablas que componen esta base de datos relacional descrita en la figura 4.7.

- **appointment_table**: Almacena información sobre las citas médicas.
- **user_table**: Almacena información sobre los usuarios del sistema, que pueden ser pacientes, doctores, y administradores.
- **user_table_roles**: Almacena los roles asignados a los usuarios.
- **doc_table**: Almacena los documentos relacionados con las intervenciones.
- **intervention_table**: Almacena las intervenciones realizadas al paciente.

- **description_table**: Almacena las descripciones de los motivos de las citas y su posible intervención a realizar.
- **util_table**: Almacena datos estadísticos del uso del sistema.

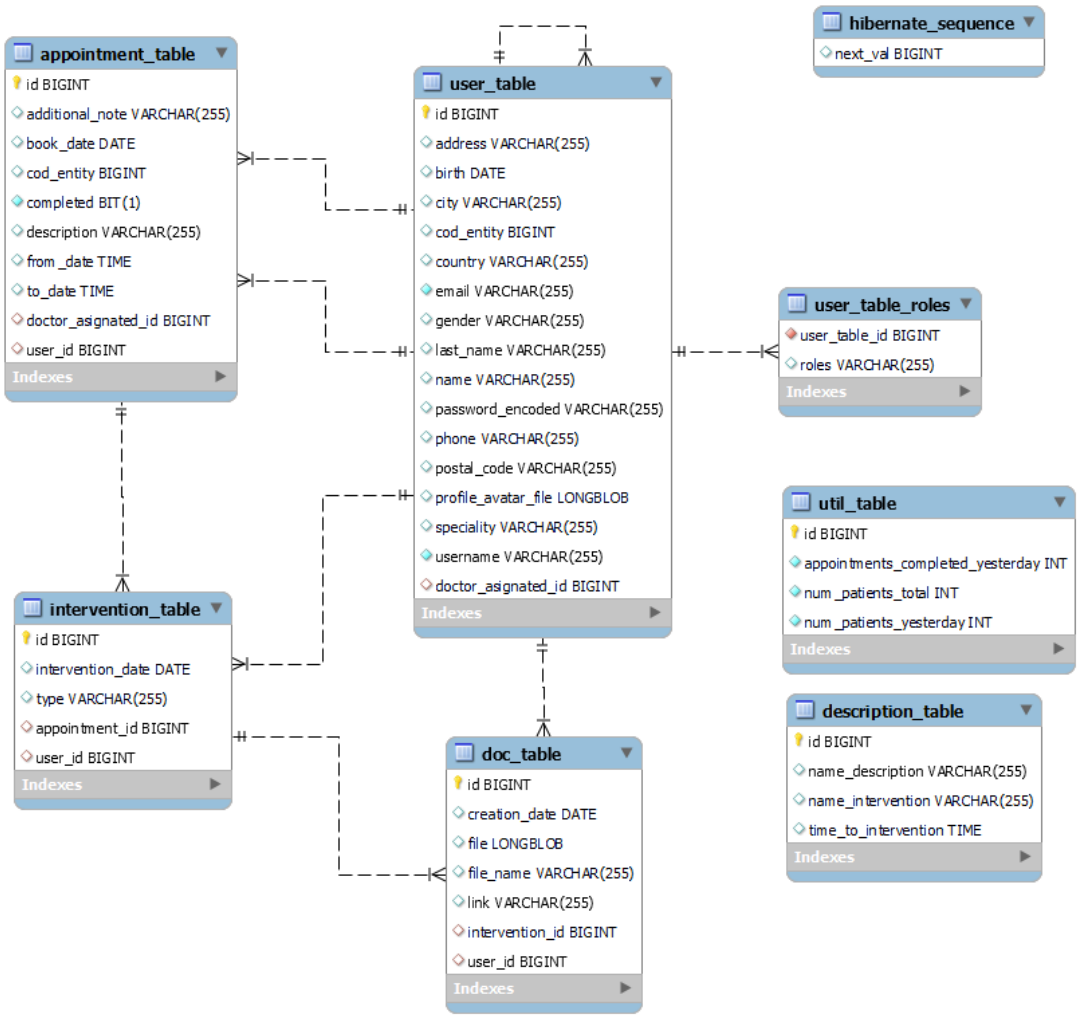


Figura 4.7: Esquema relacional Base de datos

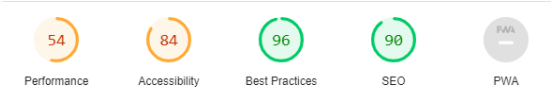
4.2.5. Análisis web y repositorio

Para realizar un análisis automatizado de la pantallas de la aplicación web se ha hecho uso de la extensión del navegador Google Chrome llamada *LightHouse* [31]. Esta herramienta analiza métricas como: *Performance, Accessibility, Best Practices y SEO*[32].

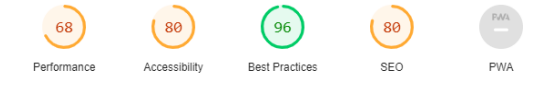
- **Performance:** Mide el rendimiento total de la página web, analizando el tiempo que tarda en visualizar el primer contenido, o el tiempo que tarda en poder interactuar con el usuario.
- **Accessibility:** Mide el uso de atributos y etiquetas de HTML, buenos ratios de contraste entre el fondo y el texto, atributos...
- **Best Practices:** Mide factores que mejoran la seguridad de la página, el uso de Javascript, certificado SSL...
- **SEO:** Mide la capacidad de interpretación de la web por los motores de búsqueda.

Como se puede ver en la figura 4.8 los grados de confianza son amplios, sobre todo, con gran holgura en el apartado de SEO, accesibilidad y buena prácticas. No obstante, se ha detectado un problema en el apartado *Performance*. En trabajos futuros, que se menciona en el capitulo 5 se comentarán acciones futuras respecto a la mejora de este proceso.

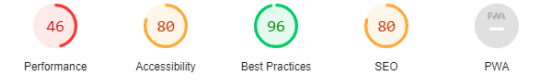
Se puede concluir de este análisis las siguientes medias: *Performance* (**58**), *Accessibility* (**85**), *Best Practices* (**96**) y *SEO* (**82**). Por tanto, hay que hacer un especial hincapié en el rendimiento total de la página web para trabajos futuros.



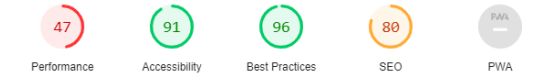
(a) Pantalla Inicio Sesión



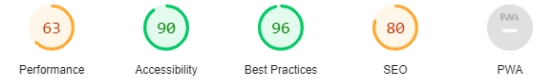
(b) Pantalla Dashboard Administrador Total



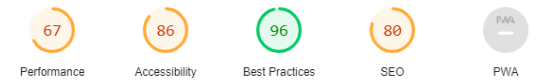
(c) Pantalla creación de cita



(d) Pantalla perfil usuario



(e) Pantalla Dashboard Paciente



(f) Pantalla Dashboard Doctor

Figura 4.8: Análisis LightHouse

Por otro lado, se ha realizado un análisis del repositorio de GitHub con la herramienta GitGuardian [33]. GitGuardian es una herramienta que trata de cubrir y detectar las posibles brechas de seguridad que suceden en los repositorios, como son el uso de credenciales en archivos de configuración. En la figura 4.9 se pueden ver los fallos. Una vez detectados esos fallos de seguridad, se implementó como respuesta a ello el uso de variables de entorno que se pasarán como parámetros al ejecutar el despliegue de la aplicación mediante docker.

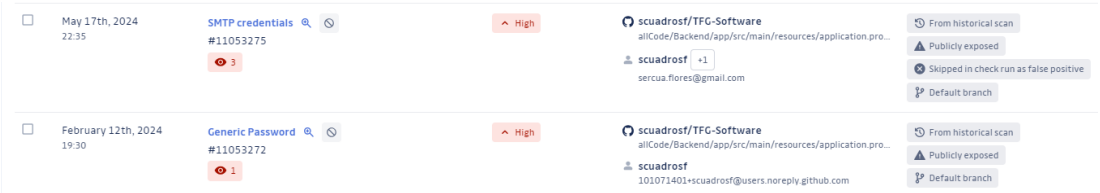


Figura 4.9: Brechas de seguridad encontradas en credenciales

4.3. Diseño e Implementación

4.3.1. Frontend

Implementación

Cabe destacar la implementación de un buscador en el listado de pacientes para encontrarlos mediante nombre, apellido o DNI. Como se puede ver en el código 4.1, consiste en un algoritmo en el se observa si cambia el valor del buscador, de esta forma, cuando se se empiece a introducir un nombre, apellido o DNI, el buscador realizará la solicitud al servidor para buscar ese campo en la base de datos y por tanto, mostrando ese paciente. En el caso en que no haya coincidencias, se le muestra que no existe un paciente con esos datos y se introduce esa condición en la variable booleana *noResults*, que indica si hay o no resultados, donde haciendo uso de esta variable se ha realizado una implementación condicional en el archivo HTML. Se puede ver en el código 4.2.

```
1  observerChangeSearch() {
2      this.control.valueChanges
3        .pipe(
4          debounceTime(500),
5          switchMap(query => {
6            this.loading = true;
7            if (query.trim().length === 0) {
8              this.noResults = false;
9              if (this.showingAllUsers) {
10               return this.patientService.getUsersByCodEntity(this
                     .codEntity).pipe(
11                 map(list => list.filter(patient =>
```

```
12                    patient.roles.length === 1 && patient.roles.
                          includes('USER'))
13                )
14              );
15            } else {
16              return this.patientService.getUserListByDoctor(this
                    .doctorAsignated).pipe(
17                map(list => list.filter(patient =>
18                  patient.roles.length === 1 && patient.roles.
                          includes('USER'))
19                )
20              );
21            }
22          } else {
23            this.noResults = false;
24            if (this.showingAllUsers) {
25              return this.patientService.
                    getUsersByNameOrLastNameOrUsername(query).pipe(
26                map(list => list.filter(patient =>
27                  patient.roles.length === 1 && patient.roles.
                          includes('USER') && patient.codEntity ==
                          this.codEntity
28                ))
29              );
30            } else {
31              return this.patientService.
                    getUsersByNameOrLastNameOrUsername(query).pipe(
32                map(list => list.filter(patient =>
33                  patient.roles.length === 1 && patient.roles.
                          includes('USER') && patient.codEntity ==
                          this.codEntity && patient.doctorAsignated.id
                           == this.doctorAsignated
34                ))
35              );
36            }
37          }
38        })
39      )
40      .subscribe(result => {
41        if (result.length === 0) {
42          this.noResults = true;
43        }
44        this.patientsList = result;
45        this.loading = false;
46      });
47  }
```

Código 4.1: Búsqueda de pacientes

```
1  <div *ngIf="noResults">
2      <p>No hay resultados coincidentes</p>
3  </div>
```

Código 4.2: Sentencia condicional HTML

Por otro lado, en relación a las interacciones que el usuario puede tener con la aplicación, se ha implementado unos sistemas de confirmación de acciones utilizando la tecnología de SweetAlert. De esta forma, el usuario si ejecuta por error una acción tiene la doble verificación de que realmente es lo que está queriendo hacer como se puede ver en la figura 4.10.

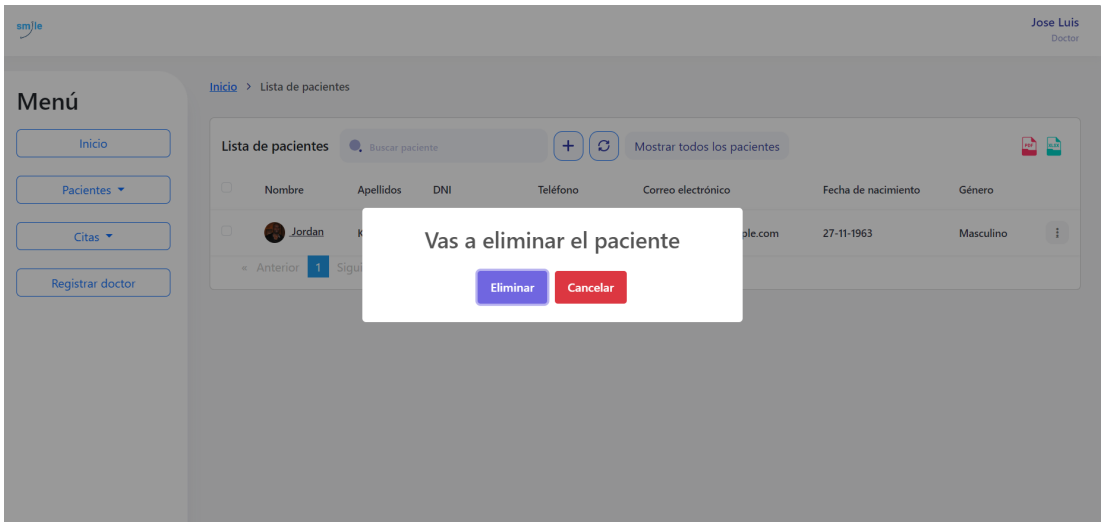


Figura 4.10: Confirmación de acción

4.3.2. Backend

Gestión de perfiles para ejecución en diferentes entornos

En la realización del backend se ha optado por desarrollar dos perfiles, uno llamado *test* y otro *production* para poder realizar acciones determinadas o, por ejemplo, configurar diferentes bases de datos dependiendo del perfil que se activara, de modo que si la aplicación se despliega con el perfil *test* mediante el comando `SPRING_PROFILES_ACTIVE=test`, se inicializará una base de datos concreta soportada por H2, y si se despliega sin determinar activamente ese parámetro del perfil se ejecutará en modo *production*, ejecutandose una base de datos MySQL. En la figura 4.11 podemos ver que se configura un perfil general de las *properties* llamado *application.properties*, y luego dos para configuraciones específicas llamados *application-production.properties* para el modo *production*, y *application-test.properties* para el modo *test*.

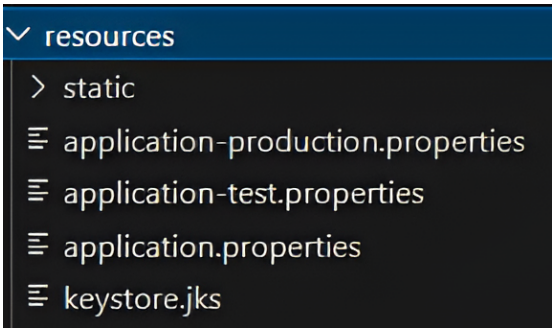


Figura 4.11: Configuraciones diferentes dependiendo del perfil indicado

Implementación

En cuanto a la gestión de los controladores, todos se tratan de *RestController*, donde mediante solicitudes HTTP se realizan las operaciones CRUD con los parámetros: *@GetMapping, @PostMapping, @PutMapping* y *@DeleteMapping*.

Además, se ha hecho uso del patrón de diseño DTO (*Data Transfer Object*). Un DTO es un tipo de objeto que sirve únicamente para transportar los datos entre capas. De esta manera, haciendo uso de un DTO que contiene únicamente las propiedades que se quieren almacenar, se facilita el transporte de la información hacia la base de datos. Cabe destacar que los DTO, no tienen procesos ni código complejo, únicamente *getters* y *setters*.

Por otro lado, se ha utilizado un DTO en el caso en el que se quiere realizar un informe médico, entre otros. De esta forma, es necesario agrupar la información contenida en la entidad *User*, *Appointment*, e *Intervention*, en un mismo objeto, filtrando por los datos que se requieren. De esta forma, gracias al uso de un DTO podemos unir este contenido. En la figura 4.12 podemos ver un ejemplo de uso de este sistema.

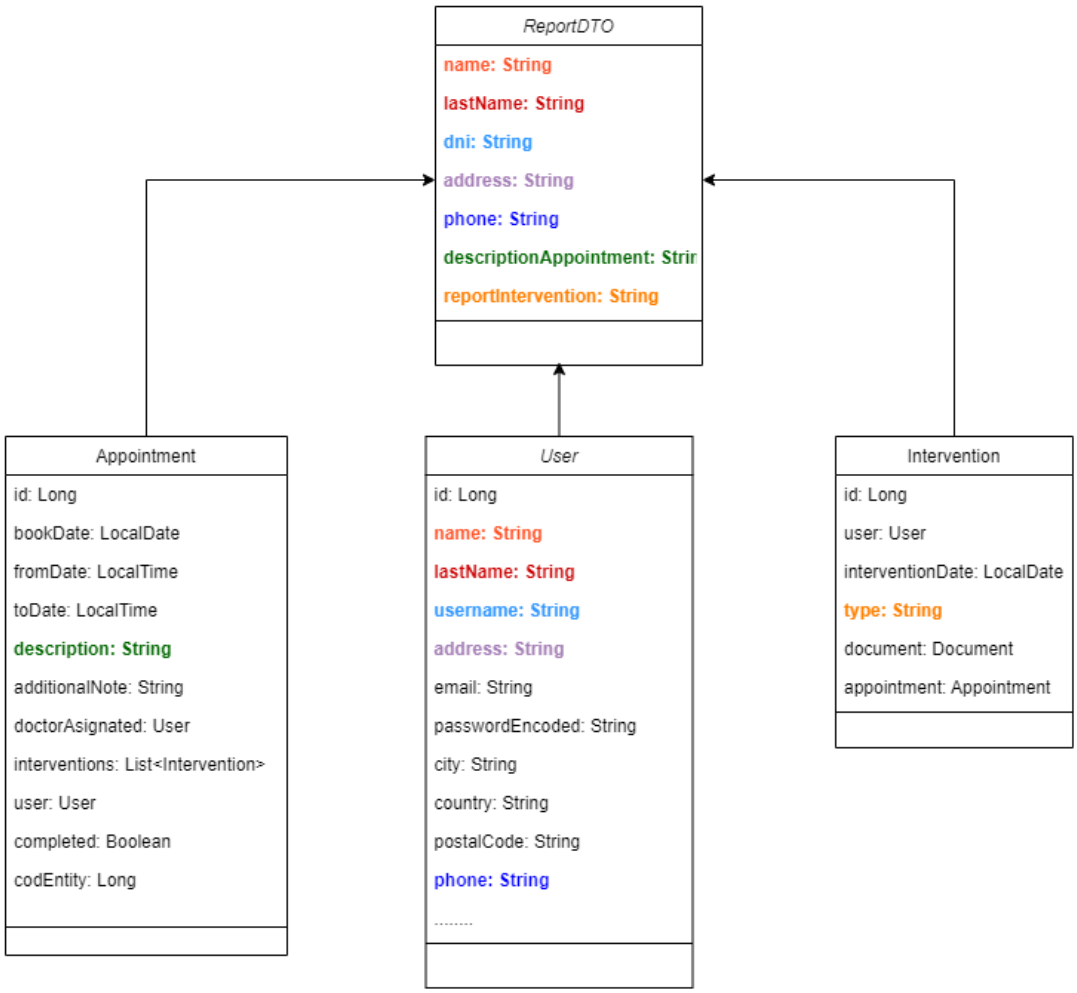


Figura 4.12: Diagrama ReportDTO

Se ha de destacar la implementación que se ha realizado con el servicio de envío de correos electrónicos a los pacientes. En esta estructura representada en la figura 4.13, se puede ver como se tiene una clase padre (Interfaz) llamada *EmailService* y dos clases hijas llamadas *EmailServiceProduction* y *EmailServiceLocal*. De esta forma, con esta configuración, se consigue que si se están ejecutando los test, es decir, se ha activado el perfil *test* se realizarán unas acciones determinadas y con el perfil de *production* otras.

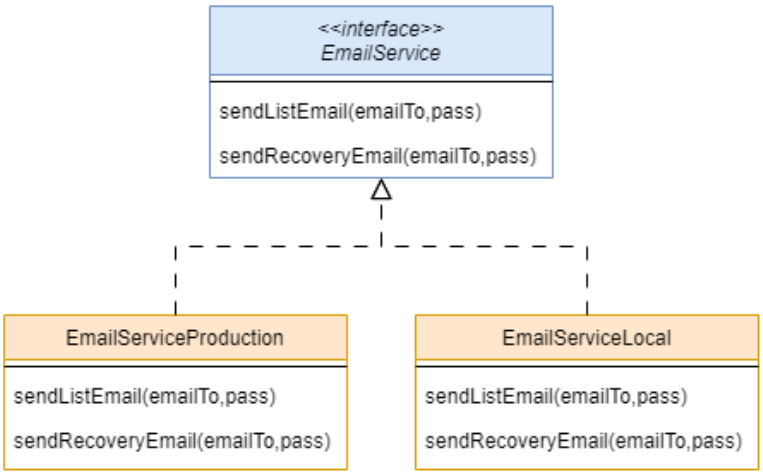


Figura 4.13: Diagrama con la implementación de Email Service

Además, se implementó un principio SOLID llamado *Dependency Inversion Principle*. Estos principios fueron introducidos por Robert C. Martin (Uncle Bob) en su escrito en el año 2000 *Design Principles and Design Patterns* [34], donde la intención de estos principios es que el diseño de nuestro código de software sea más entendible, más manejable, fácil de extender y agregar nuevas funcionalidades.[35]. En este caso se ha aplicado para la gestión del envío de correos electrónicos a los pacientes que se mencionó en la figura anterior (4.13).

Este principio menciona que los módulos de alto nivel no deben depender de los módulos de bajo nivel. Ambos deben depender de abstracciones. Y que las abstracciones no deben depender de los detalles, sino que los detalles deben depender de las abstracciones. Por ello en el diagrama descrito, la interfaz *EmailService* actúa como una abstracción que define los métodos a realizar. Y las clases *EmailServiceProduction* y *EmailServiceLocal* implementan esta interfaz, proporcionando las implementaciones específicas de esos métodos.

Con este ajuste, se consigue un desacomplamiento, pues las clases de alto nivel que dependen de *EmailService* no necesitan conocer los detalles de las implementaciones concretas. Solo interactúan con él. Además, se consigue una buena flexibilidad, puesto que si se quiere cambiar implementación de *EmailService* no se verían afectadas las clases que dependen de ella. Por último, se facilita la realización de pruebas.

Por otro lado, un controlador a destacar es la validación de que no haya otra cita en el mismo tramo horario. Como se puede ver en el código 4.3, es importante gestionar las citas de los pacientes para que no ocurra el problema de que dos pacientes tengan cita a la misma hora, o que se superpongan en horas.

```
@PostMapping("/check-availability")
public ResponseEntity<Boolean> checkAppointmentAvailability(
    @RequestBody AvailabilityCheckRequest request) {
    List<Appointment> appointments = appointmentService
            .findAllAppointmentsByDoctorAsignatedId(request.
               getDoctorId());
    boolean isDoctorAvailable = true;
    for (Appointment appointment : appointments) {
        if (appointment.getBookDate().isEqual(request.getBookDate
           ()) &&
                !(appointment.getFromDate().isAfter(request.
                   getToDate()) ||
                        appointment.getToDate().isBefore(request.
                           getFromDate()))) {
            return ResponseEntity.ok(false);
        }
    }

    return ResponseEntity.ok(isDoctorAvailable);
}
```

Código 4.3: Comprobación de disponibilidad horaria

4.3.3. Base de datos

La base de datos se compone de varias tablas, cada una con un propósito específico y con relaciones definidas. A continuación se describen las tablas y sus interrelaciones, haciendo referencia al esquema relacional proporcionado en la figura 4.7.

- **appointment_table**: Citas médicas: Uso de la tabla en la figura 4.14.
  - **Campos principales:**
    - **id**: Identificador único.
    - **book_date, from_date, to_date**: Fecha, hora de inicio y hora de fin de una cita.
    - **description**: Motivo de la cita.
    - **cod_entity**: Código identificador de la entidad/clínica.
    - **completed**: Booleano con la condición de si la cita ha sido realizada (true) o no (false).
    - **doctor_asignated_id**: Identificador del doctor asignado a la cita.
    - **user_id**: Identificador del paciente.
  - **Relaciones:**
    - Relación **muchos a uno** con *user_table* a través de *doctor_asignated_id* y *user_id*.

- Relación **uno a muchos** con *intervention_table* a través de *appointment_id* en la tabla de intervention.

id	additional_note	book_date	cod_entity	completed	description	from_date	to_date	doctor_asignated_id	user_id
29	NULL	2024-03-05	200	1	Mantenimiento y Prevención	11:45:00	12:05:00	11	22
30	NULL	2024-04-19	200	1	Problemas Específicos y Emergencias	11:45:00	12:05:00	11	22
24	NULL	2023-07-25	200	0	Problemas Comunes y Tratamientos de Rutina	13:00:00	13:20:00	11	22

Figura 4.14: Tabla Appointment con algunos datos

- **user_table**: Usuarios del sistema (Pacientes, doctores y administradores). Uso de la tabla en la figura 4.15.
  - **Campos principales:**
    - **id**: Identificador único.
    - **address, city, country, postal_code**: Datos para recoger el domicilio del usuario.
    - **birth_date, gender, phone, name, last_name, username (DNI), profile_avatar_file**: Datos sobre el usuario.
    - **cod_entity**: Código identificador de la entidad/clínica.
    - **email, password_encoded**: Correo electrónico y contraseña para autenticarse.
    - **doctor_asignated_id**: Identificador del doctor asignado al paciente.
    - **speciality**: Especialidad a la que se dedica el doctor.
  - **Relaciones:**
    - Relación **uno a muchos** con *appointment_table* a través de *user_id* en la tabla de appointment.
    - Relación **uno a muchos** con *intervention_table* a través de *user_id* en la tabla de intervention.
    - Relación **uno a muchos** con *doc_table* a través de *user_id* en la tabla de doc.
    - Relación **uno a muchos** con *user_table_roles* a través de *user_id* en la tabla de roles.
    - Relación **uno a muchos** reflexiva. (Un doctor está en muchos pacientes).

id	address	birth	city	cod_entity	country	email	gender	last_name	name	password_encoded	phone
1	NULL	NULL	NULL	NULL	NULL	admin@smilelink.es	NULL	Administrador	Administrador	[redacted]	NULL
2	NULL	2024-05-29	NULL	100	NULL	admin100@smilelink.es	Masculino	Ane	Fernando	[redacted]	711548969
4	NULL	2024-05-29	NULL	200	NULL	admin200@smilelink.es	Masculino	Rodriguez	Jose Luis	[redacted]	785264122
5	NULL	1993-05-12	NULL	100	NULL	Anton.Lehtinen@smilelink.es	Masculino	Lehtinen	Anton	[redacted]	06-001-894
6	NULL	1961-07-09	NULL	200	NULL	Rasmus.Ranta@smilelink.es	Masculino	Ranta	Rasmus	[redacted]	03-824-469
7	NULL	1995-01-20	NULL	100	NULL	William.Fleury@smilelink.es	Masculino	Fleury	William	[redacted]	01-02-63-21-35
8	NULL	1998-12-19	NULL	200	NULL	Alida.Schüttler@smilelink.es	Femenino	Schüttler	Alida	[redacted]	0723-0593832
9	NULL	1959-12-24	NULL	100	NULL	Jacob.Taylor@smilelink.es	Masculino	Taylor	Jacob	[redacted]	(601)-506-1172
10	NULL	1946-06-27	NULL	200	NULL	María Elena.Gonzales@smilelink.es	Femenino	Gonzales	María Elena	[redacted]	(648) 082 7162
11	NULL	1993-05-29	NULL	200	NULL	jaime.flores@smilelink.es	Masculino	Flores	Jaime	[redacted]	444444444
12	7917 Rua Castro Alves	1962-06-15	Gua…	100	Brazil	julio.viana@example.com	Masculino	Viana	Júlio	[redacted]	(68) 1129-0713
13	9442 Lisozahisna	1976-03-29	Kali…	200	Ukraine	nagnibida.kis@example.com	Masculino	Kis	Nagnibida	[redacted]	(096) 514-6361

Figura 4.15: Tabla User con algunos datos

- **user_table_roles**: Roles de los usuarios. Uso de la tabla en la figura 4.16.
  - **Campos principales:**
    - **user_table_id**: Identificador único.
    - **roles**: Rol que ejerce. ADMIN, DOCTOR, USER
  - **Relaciones:** Relación **muchos a uno** con *user_table*

user_table_id	roles
1	ADMIN
2	DOCTOR
2	ADMIN
4	DOCTOR
4	ADMIN
5	DOCTOR
6	DOCTOR
7	DOCTOR
8	DOCTOR
9	DOCTOR
10	DOCTOR
11	DOCTOR
12	USER
13	USER
14	USER

Figura 4.16: Tabla de Roles con algunos datos

- **doc_table**: Documentos de los pacientes. Uso de la tabla en la figura 4.17.
  - **Campos principales:**
    - **id**: Identificador único.
    - **intervention_date**: Fecha de realización de la intervención.
    - **type**: Contenido que el doctor quiere dejar reflejado en la intervención.
    - **appointment_id**: Identificador de la cita a la que se hace intervención.
    - **user_id**: Identificador del usuario al que se hace la intervención.
  - **Relaciones:**
    - Relación **mucho a uno** con *appointment_table* a través de *appointment_id*.
    - Relación **mucho a uno** con *user_table* a través de *user_id*.

id	creation_date	file	file_name	link	intervention_id	user_id
32	2024-05-29	NULL	pdf-ejemplo.pdf	NULL	31	22
53	2024-05-29	BLOB	IFCD015PO.pdf	NULL	52	22

Figura 4.17: Tabla Doc con algunos datos

- **intervention_table**: Intervenciones de los pacientes. Uso de la tabla en la figura 4.18.
  - **Campos principales:**
    - **id**: Identificador único.
    - **creation_date**: Fecha de creación del documento.
    - **file, file_name**: Fichero blob y nombre del fichero.
    - **link**: Futuro campo para introducir archivos S3 de AWS.
    - **intervention_id**: Identificador de la intervención a donde ha sido adjuntado el documento.
    - **user_id**: Identificador del usuario que posee ese documento.
  - **Relaciones:**
    - Relación **muchos a uno** con *appointment_table* a través de *appointment_id*.
    - Relación **muchos a uno** con *user_table* a través de *user_id*.
    - Relación **uno a muchos** con *doc_table* a traves de *intervention_id* de la tabla doc.

id	intervention_date	type	appointment_id	user_id
31	2024-05-29	Mantenimiento y Prevención	29	22
33	2024-05-29	Problemas Específicos y Emerge...	30	22
52	2024-05-29	Empaste realizado	26	22

Figura 4.18: Tabla Intervention con algunos datos

- **description_table**: Descripciones de las citas. Uso de la tabla en la figura 4.19.
  - **Campos principales:**
    - **id**: Identificador único.
    - **name_description**: Descripción de la cita.
    - **name_intervention**: Descripción de la posible intervención a realizar.
    - **time_to_intervention**: Tiempo dedicado a realizar esa intervención. Usado para guardar ese tramo de tiempo al pedir cita.

id	name_description	name_intervention	time_to_intervention
34	Mantenimiento y Prevención	Chequeos y limpiezas regulares	00:30:00
35	Mantenimiento y Prevención	Consulta sobre hábitos orales	00:15:00
36	Mantenimiento y Prevención	Prevención de enfermedades dentales y educac...	00:15:00
37	Problemas Comunes y Tratamientos de Rutina	Obturación simple	00:20:00
38	Problemas Comunes y Tratamientos de Rutina	Gran reconstrucción	01:00:00
39	Problemas Comunes y Tratamientos de Rutina	Dolor de encías	01:30:00
40	Problemas Comunes y Tratamientos de Rutina	Problemas de encías	00:45:00

Figura 4.19: Tabla Description con algunos datos

- **util_table**: Datos estadísticos

  - **Campos principales:**
    - **id**: Identificador único.
    - **appointments_completed_yesterday**: Cantidad de citas realizadas el día anterior.
    - **num_patients_total**: Número total de pacientes en el sistema.
    - **num_patients_yesterday**: Número total de pacientes en el sistema el día anterior.

Repositorio GitHub

Como se puede ver, se ha proporcionado una figura (4.20) en la que se muestra la evolución de los *commit* realizados al repositorio de GitHub. En esta gráfica se muestra que se ha mantenido una evolución constante en el tiempo desde el comienzo del proyecto, no obstante, hay días en los que se ha trabajado más, y otros en los que o no se ha trabajado o no entraba la capacidad de realizar un *commit* considerable.

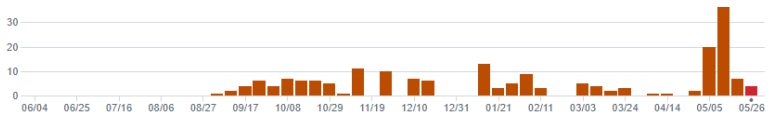


Figura 4.20: Evolución de commits en GitHub

4.4. Pruebas

En esta sección se describen las pruebas realizadas en el desarrollo de la aplicación.

Las pruebas aseguran que la aplicación funcione correctamente y cumpla con los requisitos especificados. De esta manera, se han implementado diferentes tipos de pruebas: pruebas de integración (API) y pruebas End-to-End (E2E). Las primeras de ellas, consisten en comprobar el correcto funcionamiento de las solicitudes y su respuesta, asegurando el buen funcionamiento del proceso backend. En cambio, en las pruebas E2E, se simula el comportamiento del usuario final para verificar que el sistema completo funcione como se espera.

Por otro lado en relación a las pruebas unitarias, debido a la forma acelerada del desarrollo del proyecto y la necesidad de entregar funcionalidad en plazos, se optó por centrarse en pruebas de integración y end-to-end que nos permitieran validar el sistema completo de manera más efectiva. Aunque se reconoce la importancia de las pruebas unitarias para asegurar la calidad del código, se han priorizado las pruebas que abarcaran el flujo de trabajo completo del usuario para garantizar que la aplicación cumpla con los requisitos funcionales y de usabilidad. En futuros ciclos de desarrollo, se planteará integrar la ejecución de pruebas unitarias para mejorar la cobertura y la robustez del código.

La ejecución de las pruebas desarrolladas se pueden realizar mediante código con el comando de Maven indicado en el código 4.4 ubicándonos en la carpeta raíz del backend (donde se encuentra el fichero *pom.xml*) o referenciando al fichero *pom.xml* mediante la sentencia `--file ruta_a_raiz_backend/pom.xml`, o mediante el IDE de desarrollo.

```
1    // Ejecucion Test Integracion (API)
2    mvn test -Dtest=**APITest
3
4    // Ejecucion Test E2E
5    mvn test -Dtest=Test_E2E/**
```

Código 4.4: Comandos para ejecutar los test por consola

Las pruebas de integración cubren los siguientes casos:

- **UserAPITest**
  - Iniciar sesión y recoger el token de autenticación.
  - Registrar un paciente.
  - Acceder a todos los usuarios.
  - Comprobación de que sin autenticación se devuelve el código 403 al querer conseguir los usuarios, por ejemplo.
  - Acceder al usuario que inicia sesión.
  - Acceder a un usuario mediante su ID.
  - Eliminar un usuario.
  - Actualizar un usuario.

- **AppointmentAPITest**
  - Crear una cita.
  - Conseguir todas las citas.
  - Conseguir una cita concreta.
  - Actualizar una cita.
  - Eliminar/Cancelar una cita.
- **InterventionAPITest**
  - Añadir una intervención a una cita de un paciente.
  - Conseguir todas las intervenciones.
  - Eliminar una intervención.
  - Actualizar una intervención.

En estas pruebas ejecutadas con la ayuda de la tecnología de Rest Assured, salvo la prueba de *Iniciar sesión*, requieren autenticación. Esta autenticación se ha conseguido gracias a la cookie llamada *AuthToken* que se crea al iniciar sesión. En el código 4.6 se muestra como se recoge la cookie tras iniciar sesión y se almacena en la variable *jwtToken*, empleada posteriormente para hacer uso de la autenticación, de esta forma, al realizar una petición que requiera de credenciales se deberá especificar en el cuerpo de la petición la cookie como se puede ver en el código 4.5.

```
1     given().cookie("AuthToken", jwtToken)
```

Código 4.5: Uso de la cookie AuthToken

```
1     Cookie jwtCookie = response.getDetailedCookie("AuthToken");
2     jwtToken = jwtCookie.getValue();
```

Código 4.6: Obtención de la cookie AuthToken

En la siguiente figura 4.21 se puede ver el momento en que se pasan todos las pruebas de integración correctamente.

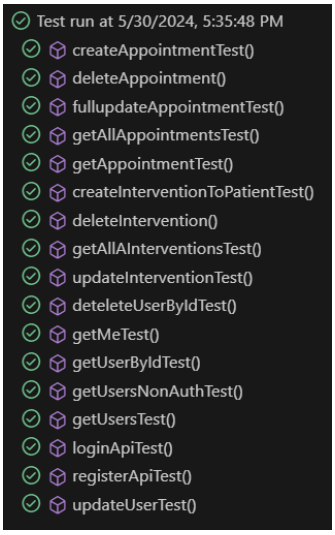


Figura 4.21: Ejecución de las pruebas de integración

La cobertura de las pruebas E2E se basa en:

- **LoginEachUserRoleTest**
  - Iniciar sesión administrador total.
  - Iniciar sesión administrador doctor.
  - Iniciar sesión doctor.
  - Iniciar sesión paciente.
- **AppointmentTest**
  - Crear una cita a un paciente.
  - Crear una cita por un paciente.
- **AddInterventionTest**
  - Añadir una intervención a una cita de un paciente.
- **AddNewEntityOfficeTest**
  - Crear una nueva entidad.
- **PatientTest**
  - Dar de alta un paciente.
  - Actualizar información de un paciente.
  - Actualizar como paciente mi información.

En la siguiente figura 4.22 se puede ver el momento en que se pasan todos las pruebas E2E correctamente.

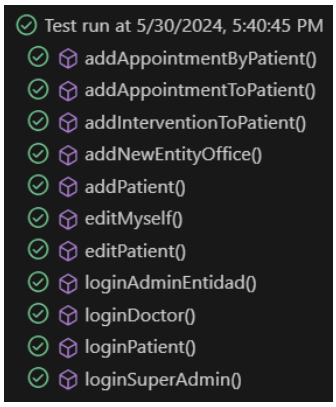


Figura 4.22: Ejecución de las pruebas E2E

En su configuración se ha decidido ejecutar con el driver de Google Chrome (aunque puede ejecutarse en navegador) en modo *headless* (sin abrir el navegador) para su futura incorporación en la integración continua (CI), además de la ejecución de las pruebas en un puerto aleatorio para no colapsar con la posible ejecución de la aplicación en producción como se observa en el siguiente comando (4.7):

```
1 @SpringBootTest(classes = AppApplication.class,
2     webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
```

Código 4.7: Ejecución de las pruebas en puerto aleatorio

Como se ha comentado en el párrafo anterior, se ha incorporado la realización de la Integración Continua (CI). La Integración Continua es una práctica de desarrollo de software en la que cada integración es verificada mediante una construcción automatizada y pruebas automatizadas para detectar errores lo más pronto posible y no hacer que perduren estos fallos durante el tiempo ni en la rama de desarrollo, ni mucho menos, pase a producción un código que realmente no funciona correctamente, o modifica el funcionamiento de otros componentes.

Este proceso se realiza mediante GitHub Actions. Esta herramienta nos ayuda a ejecutar las pruebas mediante un pipeline específico. El flujo de trabajo es el siguiente: El desarrollador realiza un *commit* al repositorio. En este momento se activa la ejecución del flujo desarrollado en el pipeline y en diferentes pasos ejecuta las pruebas de integración y end-to-end. Si todas las pruebas han ido correctamente, el final del flujo de trabajo será exitoso. Si alguna prueba falla, será erróneo. En la siguiente sección 4.5 se explicará más en detalle.

4.5. Distribución y despliegue

En este apartado se describen los procesos para la distribución y despliegue de la aplicación web desarrollada en este proyecto.

Herramientas y tecnologías utilizadas

La distribución y despliegue de la aplicación web desarrollada en este proyecto han sido gestionados mediante una serie de herramientas y tecnologías que aseguran una entrega continua, confiable y eficiente. De esta manera, se ha hecho uso de Docker para empaquetar la aplicación completa. Docker consigue encapsular todas las dependencias en un contenedor, mediante la construcción del frontend con Node.js y npm, y el backend con Maven. De esta forma, conseguimos con los comandos 4.8, la construcción y publicación de la imagen Docker:

```
1 docker build -t scuadrosf/smilelink:latest -f Docker/Dockerfile .
2
3 docker push scuadrosf/smilelink:latest
```

Código 4.8: Construcción y subida de imagen Docker

Esta imagen Docker nos servirá posteriormente para ejecutarla en la instancia de la nube que se comentará más adelante.

Además, se ha desarrollado un pipeline[2] en GitHub Actions en el que se gestiona la Integración Continua, la entrega continua y el Despliegue Continuo.

Este pipeline se desarrolla en dos jobs: *build-and-test* y *deploy-on-master*. Lo que se quiere conseguir con este diseño del pipeline es que se puedan ejecutar múltiples pruebas automatizadas al subir nuevo código en desarrollo y poder llegar a producción con código estable.

1. ***build-and-test***

   - ¿Cuándo se ejecuta?

   Este job se ejecuta independientemente de en qué rama se accione la acción. En este caso en desarrollo y master.

   - ¿Que ocurre en esta ejecución?

   En este job se realizarán las pruebas desarrolladas en Java en diferentes pasos como se puede ver en la figura 4.23a, primero se hará un clonado del repositorio completo, después se procederá a realizar los test de integración,

[2] `https://github.com/scuadrosf/TFG-Software/blob/504f88bd028de7339711bd90fce702ee373923c5/.github/workflows/maven.yml`

una vez hayan concluido los test, la máquina construirá una imagen docker para el siguiente paso que es la ejecución de los test E2E bajo el perfil de test. Lo podemos ver en el paso descrito con referencia 4.9 en el que se le pasa la variable de entorno `SPRING_PROFILES_ACTIVE=test` y se han usado secretos de GitHub para una mayor seguridad.

Para no pasar al siguiente paso y que ocurran errores puesto que la aplicación no se ha desplegado aún, se creó un script en el que identificaba cuándo la aplicación estaba levantada (Código 4.10) y se podía proceder al siguiente paso.

El siguiente procedimiento es realizar las pruebas E2E, donde finalmente, si todo ha ido correctamente se concluirá el proceso de *build-and-test*.

2. ***deploy-on-master***

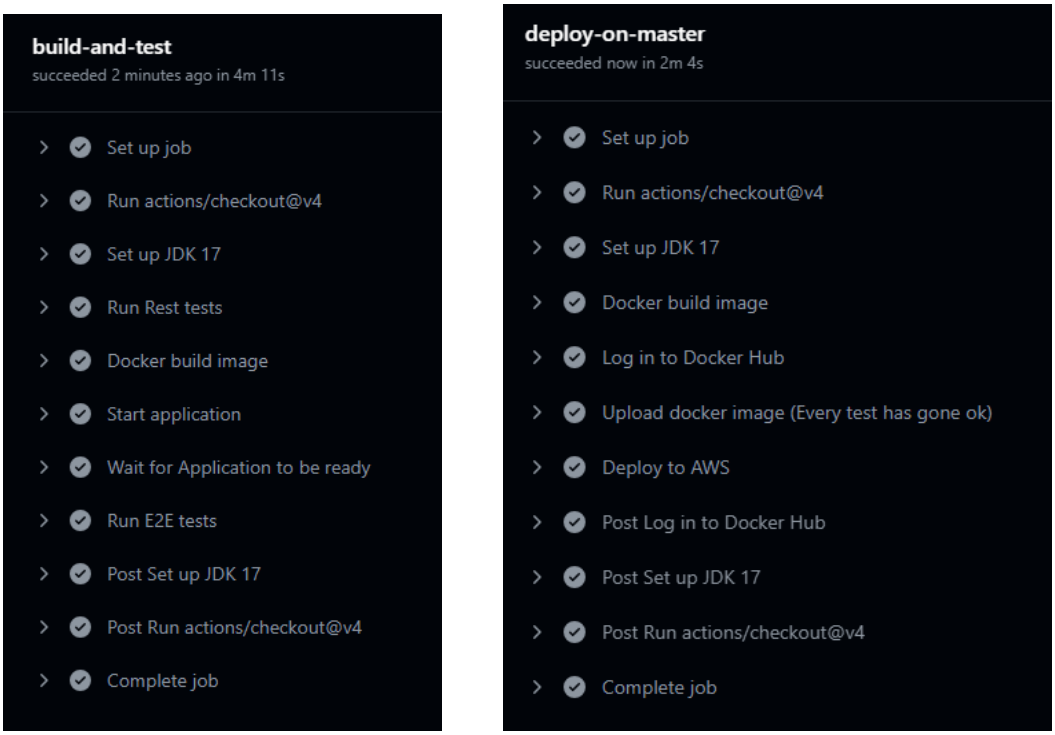
   - ¿Cuándo se ejecuta?

   Este segundo job únicamente se ejecuta si la acción se está realizando sobre la rama master, es decir, producción.

   - ¿Que ocurre en esta ejecución?

   Este job (Figura 4.23b) publicará la nueva imagen construida en el repositorio de Docker Hub y posteriormente se conectará con la instancia desplegada en AWS y reiniciará la operativa incluyendo los nuevos cambios, en el caso en que esté encendida la máquina, y sino, procederá a su encendido. De esta manera se ejecuta la Entrega Continua y el Despliegue Continuo (CD) y el fin del flujo de trabajo.

Como se puede observar, en un proceso de subida a producción primero se ejecutarían las pruebas en desarrollo, y una vez pasadas todas las pruebas, se procede a hacer el proceso de subida a master mediante un merge, de esta forma, se volverán a ejecutar las pruebas ya realizadas en desarrollo para una doble verificación, y se ejecutará el job específico de master (*deploy-on-master*).



(a) Pasos del job build-and-test (b) Pasos del job deploy-on-master

Figura 4.23: Pasos de los Job del workflow

En la figura 4.24 se puede ver el workflow de la ejecución de los job de automatización.

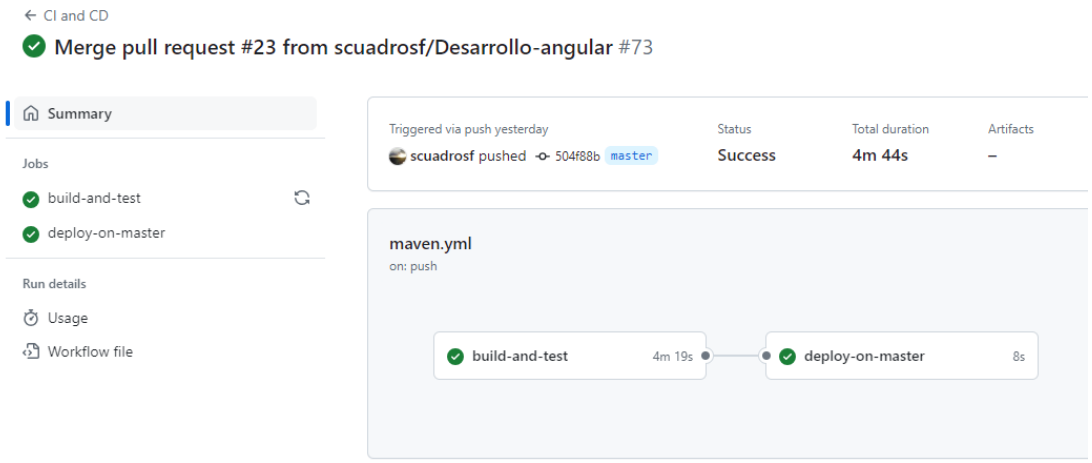


Figura 4.24: Workflow ejecución de CI/CD

Se ha dejado en el pie[3] de la página una URL para acceder directamente al pipeline desarrollado para la automatización de los procesos.

```
- name: Start application
  run: docker run -d -p 443:443 -e SPRING_PROFILES_ACTIVE=test --
     name ${{ secrets.DOCKERHUB_IMAGE }} ${{ secrets.
     DOCKERHUB_USERNAME }}/${{ secrets.DOCKERHUB_IMAGE }}:latest
```

Código 4.9: Ejecución de imagen docker bajo el perfil test

```
- name: Wait for Application to be ready
  run: |
       until $(curl --output /dev/null --head -k --fail https://
          localhost:443); do
         printf '.'
         sleep 5
       done
```

Código 4.10: Espera hasta que la aplicación esté desplegada

Para el despliegue en la nube se ha realizado mediante el soporte de Amazon Web Services (AWS). Para ello, se ha creado una instancia EC2 (Elastic Compute Cloud), una máquina virtual con las siguientes características:

- Sistema operativo Ubuntu Server 24.04.
- Tipo de servicio virtual (tipo de instancia) t2.micro, que contiene 1 vCPU (CPU virtual) y 1 GiB de memoria.
- Funciona mediante una VPC (Virtual Private Cloud) interna y ajustada a una subred.
- 8 GiB de almacenamiento
- Asignación de IPv4 públicas dinámicas
- 100 GB de ancho de banda a Internet.

Además, para dar soporte a la base de datos, se ha creado una RDS (Relational Database Service) con las siguiente configuración:

- Tipo de motor MySQL Community 8.0
- Tipo de instancia db.t3.micro con 2 vCPU, 1 GiB RAM y 2085 Mbps de red
- Almacenamiento de 20 GiB en SSD de uso general (gp2)

[3] https://github.com/scuadrosf/TFG-Software/blob/504f88bd028de7339711bd90fce702ee373923c5/.github/workflows/maven.yml

Dentro de la instancia EC2, se ha configurado un servicio llamado *scuadrosf-smilelink.service* dentro de la carpeta `/etc/systemd/system` para que cuando se encienda o reinicie la instancia, ejecute un script específico, y cuando se apague, elimine todas las imágenes zombie que se han quedado almacenadas para evitar colapsos de almacenamiento. El servicio creado se muestra en el código 4.11.

```
[Unit]
Description=Run script with docker image
After=network.target docker.service

[Service]
Type=oneshot
ExecStart=/usr/local/bin/run-app.sh
RemainAfterExit=true
Restart=on-failure
RestartSec=5s

[Install]
WantedBy=multi-user.target

[Unit]
Description=Remove all Docker images on shutdown
DefaultDependencies=no
Before=shutdown.target

[Service]
Type=oneshot
ExecStart=/usr/local/bin/remove-docker-images.sh
RemainAfterExit=true

[Install]
WantedBy=shutdown.target
```

Código 4.11: Servicio

El script que se ejecuta al iniciarse o reiniciarse la instancia tiene un proceso complejo (4.12). En primer lugar, se comprueba que esté instalado Docker para su posterior uso al lanzar la aplicación como se ha comentado al principio de la sección. Seguidamente, se obtuvo el problema de que AWS cobra por reservarte una dirección IP puesto que son limitadas, de esta forma para paliar el problema de asignar una IP fija mediante una Elastic IP, se abordó la iniciativa de obtener la dirección IP de la máquina virtual (que va cambiando en cada ejecución) y mediante la API de IONOS [36], que es el host del dominio de SmileLink, se hace una actualización de los parámetros del registro A y www, que son los que se exponen a Internet. Por otro lado, para automatizar más el proceso, se incluyó la sentencia para iniciar la instancia RDS usando AWS CLI. Una vez que el proceso de encendido de la base de datos se complete, se crean variables de entorno con las credenciales y contenido crítico para el público, que se pasarán al lanzamiento de la imagen de docker para paliar el problema de fallo de seguridad obtenido mediante GitGuardian.

```
#!/bin/bash

# Instalar Docker si no esta instalado
if ! which docker > /dev/null; then
    sudo apt-get update -y
    sudo apt-get install docker.io -y
    sudo systemctl start docker
    sudo systemctl enable docker
fi

INSTANCE_IP=$(curl -s http://checkip.amazonaws.com/)

data=$(printf '{"disabled": false, "content": "%s", "ttl": 3600,
   "prio": 0}' "$INSTANCE_IP")

# Modificar Registro A (@) IONOS
curl -X 'PUT' \
  'https://api.hosting.ionos.com/dns/v1/zones/ab0aaff4-c068-11ee-
     a410-0a5864441317/records/c909a64d-2588-7639-26a9-
     d4e0b5e94f43' \
  -H "X-API-Key: [PUBLIC KEY].[SECRET KEY]" \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d "$data"

sleep 2

# Modificar Registro A (www) IONOS
curl -X 'PUT' \
  'https://api.hosting.ionos.com/dns/v1/zones/ab0aaff4-c068-11ee-
     a410-0a5864441317/records/216dea74-67f4-1b08-4f26-
     f97214ee9afd' \
  -H "X-API-Key: [PUBLIC KEY].[SECRET KEY]" \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d "$data"

# Iniciar base de datos para su completa ejecucion
aws rds start-db-instance --db-instance-identifier smilelinkdb2

# Funcion para verificar el estado de la instancia de RDS
check_rds_status() {
    instance_status=$(aws rds describe-db-instances --db-instance
       -identifier smilelinkdb2 --query 'DBInstances[0].
       DBInstanceStatus' --output text)
    echo "Estado de la instancia de RDS: $instance_status"
    if [ "$instance_status" = "available" ]; then
        echo "La instancia de RDS esta iniciada."
        return 0
    else
        echo "La instancia de RDS aun no esta iniciada. Esperando
           ..."
        return 1
```

```
46      fi
47  }
48  # Esperar hasta que la instancia de RDS este iniciada
49  while ! check_rds_status; do
50      sleep 30  # Esperar 30 segundos antes de volver a verificar
51  done
52  sleep 5
53
54  export SPRING_DATASOURCE_URL=[URL]
55  export SPRING_DATASOURCE_USERNAME=[USER]
56  export SPRING_DATASOURCE_PASSWORD=[PASSWORD]
57  export SPRING_MAIL_USERNAME=[EMAIL]
58  export SPRING_MAIL_PASSWORD=[PASSWORD]
59
60  sudo docker stop smilelink || true
61  sudo docker rm smilelink
62  sudo docker pull scuadrosf/smilelink:latest
63  sudo docker run -p 443:443 \
64      -e SPRING_MAIL_USERNAME=$SPRING_MAIL_USERNAME \
65      -e SPRING_MAIL_PASSWORD="$SPRING_MAIL_PASSWORD" \
66      -e SPRING_DATASOURCE_URL=$SPRING_DATASOURCE_URL \
67      -e SPRING_DATASOURCE_USERNAME=$SPRING_DATASOURCE_USERNAME \
68      -e SPRING_DATASOURCE_PASSWORD=$SPRING_DATASOURCE_PASSWORD \
69      --name smilelink scuadrosf/smilelink:latest
```

Código 4.12: Script publicación de la aplicacion

5

Conclusiones y trabajos futuros

Reflexión sobre el Trabajo Realizado

El desarrollo de esta aplicación web ha sido un proyecto desafiante y enriquecedor que ha permitido abordar una amplia gama de tecnologías y metodologías. Desde el inicio, se plantearon objetivos claros y específicos con fechas, que guiarían cada fase del proyecto, haciendo su desarrollo más llevadero.

Objetivos Cumplidos

Se ha creado una interfaz intuitiva y atractiva utilizando Angular, CSS y Bootstrap, lo que permite una navegación sencilla y una experiencia de usuario fluida tanto en dispositivos móviles como en escritorio, como se ha podido observar gracias al análisis proporcionado por LightHouse.

Las funcionalidades de Crear, Leer, Actualizar y Borrar (CRUD) para citas, intervenciones, pacientes, entidades y documentos han sido implementadas correctamente, facilitando la gestión completa de los datos médicos. Seguridad:

La seguridad de la aplicación se ha reforzado mediante la implementación de autenticación basada en JWT (JSON Web Token), manejando que solo los usuarios autorizados puedan acceder a ciertas funcionalidades.

Una de las tareas más costosas en su desarrollo ha sido la ejecución de pruebas de integración y End-to-End, utilizando Selenium y RestAssured garantizando la funcionalidad del sistema, y la implementación de un pipeline de Integración

Continua (CI) y Despliegue Continuo (CD) utilizando GitHub Actions, Docker y AWS, automatizando el proceso de construcción, prueba y despliegue de la aplicación, puesto que eran tecnologías que no conocía, añadiendo dificultad en su desarrollo. No obstante, la posibilidad de realizar este Trabajo de Fin de Grado ha sido la oportunidad culminante para aprender sobre ellas.

La implementación de bases de datos relacionales con MySQL para producción y H2 para desarrollo y pruebas ha permitido una gestión eficiente y escalable de los datos.

Aspectos Pendientes y Trabajos Futuros

A pesar de los logros alcanzados, existen varios aspectos que podrían mejorarse o ampliarse en futuros trabajos:

Mejora en la escalabilidad y rendimiento:

Optimizar el rendimiento de la aplicación mediante la implementación de técnicas avanzadas de caching (almacenamiento en caché) y balanceo de carga mediante la implementación de Load Balancers en AWS (ELB) para conseguir mejorar el valor de *Performance* obtenido tras el análisis de LightHouse.

Ampliar la infraestructura de la nube para soportar un mayor número de usuarios simultáneos.

Funcionalidades adicionales:

1. Implementar recordatorios automáticos por correo electrónico o SMS para las citas.
2. Desarrollar un sistema de informes y estadísticas más avanzado para los administradores de la clínica.
3. Integración con entidades sanitarias públicas y privadas.
4. Implementar autenticación multifactor (MFA) para aumentar la seguridad de la aplicación.
5. Integración con otros campos sanitarios como por ejemplo oftalmología.
6. Implementación de un sistema de almacenamiento de documentos en AWS S3 en producción.
7. Implementación de un sistema NoSQL como MongoDB para almacenar datos semi-estructurados como son los tipos de citas y su duración mediante documentos de MongoDB.

8. Mejora de las bases de datos pudiendo dividir la tabla de usuarios, en pacientes y administradores (doctores).

9. Implementar un sistema de comunicación directa con el doctor asignado.

Como se puede ver no hay un final escrito en la elaboración de este proyecto, siempre se puede aumentar la complejidad y por tanto la perfección de la aplicación.

Conclusiones Personales

La realización de este Trabajo de Fin de Grado ha sido una experiencia profundamente gratificante puesto que tenía este proyecto en mente desde hace tiempo y gracias a esta oportunidad he podido llevarlo a cabo.. A lo largo del desarrollo del proyecto, he tenido la oportunidad de profundizar mis conocimientos en una variedad de tecnologías y metodologías modernas, desde el desarrollo frontend con Angular hasta la gestión de bases de datos con MySQL y la implementación de pipelines de CI/CD con GitHub Actions y Docker.

Este proyecto no solo me ha permitido aplicar conocimientos teóricos adquiridos durante la carrera, sino que también me ha enseñado a enfrentar y resolver problemas prácticos del mundo real. La experiencia de trabajar con un entorno de desarrollo completo, gestionar la seguridad de datos sensibles y asegurar la calidad del código mediante pruebas automatizadas ha sido una carga de gran valor.

Además, he aprendido la importancia de la colaboración y la comunicación efectiva entre servicios, especialmente al trabajar con tecnologías en la nube y herramientas de control de versiones como Git y GitHub. Estas habilidades serán esenciales para mi futura carrera profesional en el desarrollo de software.

En resumen, este proyecto ha sido un viaje de aprendizaje y crecimiento personal y profesional. Estoy satisfecho con los resultados alcanzados y emocionado con empezar las implementaciones de mejora.

Bibliografía

[1] M. de Sanidad, "Informe anual del sistema nacional de salud," pp. 147 – 151, 2021, accedido 20-05-2024. [Online]. Available: https://www.sanidad.gob.es/estadEstudios/estadisticas/sisInfSanSNS/tablasEstadisticas/InfAnualSNS2020_21/INFORME_ANUAL_2020_21.pdf

[2] A. española de fabricantes de pasta papel y cartón, "Consumo de papel españa," 2023, accedido 24-05-2024. [Online]. Available: https://www.aspapel.es/content/la-industria-de-la-celulosa-y-el-papel-incrementa-su-facturacion-pero-experimenta-un-retroce#:~:text=En%20la%20comparativa%20europea%2C%20con,141%2C4%20Kg%20per%20cpita.

[3] Statista, "Consumo de papel y cartón en españa de 2010 a 2022," Statista, Tech. Rep., 2023, accedido 19-05-2024. [Online]. Available: https://es.statista.com/estadisticas/544531/consumo-de-papel-y-carton-en-espana/

[4] V. Raffio, "El uso de herramientas de salud digital en las consultas de atención primaria aumenta en europa," 2019, accedido 21-05-2024. [Online]. Available: https://www.uoc.edu/es/news/2019/151-tic-atencion-primaria

[5] Microsoft, "Typescript is javascript with syntax for types." 2012-2024, accedido 24-05-2024. [Online]. Available: https://www.typescriptlang.org/

[6] Oracle, "¿qué es java?." accedido 24-05-2024. [Online]. Available: https://azure.microsoft.com/es-es/resources/cloud-computing-dictionary/what-is-java-programming-language

[7] B.Gustavo, "Bash script: qué es, cómo escribir uno y ejemplos," 2023, accedido 24-05-2024. [Online]. Available: https://www.hostinger.es/tutoriales/bash-script-linux#Que_es_Bash

[8] Google, "Angular," 2010-2024, accedido 24-05-2024. [Online]. Available: https://angular.io

[9] M. A. Alvarez, "Qué es una spa," 2016, accedido 24-05-2024. [Online]. Available: https://desarrolloweb.com/articulos/que-es-una-spa.html

[10] SweetAlert2, accedido 24-05-2024. [Online]. Available: https://sweetalert2.github.io

[11] Bootstrap, accedido 24-05-2024. [Online]. Available: https://getbootstrap.com

[12] Y. Muradas, "Qué es spring framework y por qué usarlo," 2018, accedido 24-05-2024. [Online]. Available: https://openwebinars.net/blog/conoce-que-es-spring-framework-y-por-que-usarlo/

[13] C. Álvarez Caules, "H2 database java y acceso remoto," 2023, accedido 24-05-2024. [Online]. Available: https://www.arquitecturajava.com/h2-database-java-y-acceso-remoto/

[14] Apryse, "itext," accedido 25-05-2024. [Online]. Available: https://itextpdf.com

[15] Apache, "Apache poi - the java api for microsoft documents," accedido 25-05-2024. [Online]. Available: https://poi.apache.org

[16] OpenPDF, "Lowagie by openpdf," accedido 25-05-2024. [Online]. Available: https://javadoc.io/doc/com.github.librepdf/openpdf/1.3.6/com/lowagie/text/pdf/package-summary.html

[17] V. Yatsyuk, "Pdfviewer," accedido 25-05-2024. [Online]. Available: https://www.npmjs.com/package/ng2-pdf-viewer

[18] "junit5," accedido 25-05-2024. [Online]. Available: https://junit.org/junit5/

[19] J. Haleby, "Rest assured," accedido 25-05-2024. [Online]. Available: https://rest-assured.io

[20] "Selenium," accedido 25-05-2024. [Online]. Available: https://www.selenium.dev

[21] "Docker," accedido 25-05-2024. [Online]. Available: https://www.docker.com

[22] "Github actions," accedido 26-05-2024. [Online]. Available: https://docs.github.com/es/actions/learn-github-actions/understanding-github-actions

[23] Apache, "Apache maven," accedido 25-05-2024. [Online]. Available: https://maven.apache.org

[24] Oracle, "Mysql workbench," accedido 25-05-2024. [Online]. Available: https://www.mysql.com/products/workbench/

[25] "Postman," accedido 25-05-2024. [Online]. Available: https://www.postman.com

[26] Atlassian, "Jira," accedido 25-05-2024. [Online]. Available: https://www.atlassian.com/es/software/jira

[27] Amazon, "Amazon web services, aws," accedido 25-05-2024. [Online]. Available: https://aws.amazon.com/es/

[28] "Ubuntu," accedido 25-05-2024. [Online]. Available: https://ubuntu.com/download

[29] "Ionos," accedido 25-05-2024. [Online]. Available: https://www.ionos.es

[30] E. Bello, "Stakeholders: quiénes son, por qué son importantes y cómo gestionarlos," 2021, accedido 26-05-2024. [Online]. Available: https://www.iebschool.com/blog/stakeholders-quienes-son-digital-business/#:~:text=En%20el%20mundo%20de%20los,los%20gobiernos%20y%20las%20comunidades

[31] Google, "Lighthouse," accedido 27-05-2024. [Online]. Available: https://chromewebstore.google.com/detail/lighthouse/blipmdconlkpinefehnmjammfjpmpbjk?hl=es

[32] Arsys, "Analizar una web con lighthouse," 2021, accedido 27-05-2024. [Online]. Available: https://www.arsys.es/blog/lighthouse#:~:text=Mide%20el%20rendimiento%20de%20la,de%20texto%20o%20im%C3%A1genes%2C%20etc.

[33] GitGuardian, accedido 27-05-2024. [Online]. Available: https://www.gitguardian.com/

[34] R. C. Martin, "Design principles and design patterns," 2000, accedido 26-05-2024. [Online]. Available: https://web.archive.org/web/20150906155800/http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf

[35] R. E. Zelaya, "Principio de inversión de dependencias (dependency inversion)," 2020, accedido 26-03-2024. [Online]. Available: https://medium.com/@ricardo.zelaya/principio-de-inversin-de-dependencias-dependency-inversion-2c48cd6dc0f8

[36] IONOS, "Api ionos documentation," accedido 20-03-2024. [Online]. Available: https://developer.hosting.ionos.es/docs/getstarted

[37] A. N, "How to find your aws access key id and secret access key," 2016, accedido 20-03-2024. [Online]. Available: https://www.msp360.com/resources/blog/how-to-find-your-aws-access-key-id-and-secret-access-key/#:~:text=1%20Go%20to%20Amazon%20Web,and%20Secret%20Access%20Key)%20option.

Apéndices

A

Lanzamiento en local

Para lanzar la aplicación SmileLink en local, sigue los siguientes pasos detallados:

A.1. Prerrequisitos

Asegúrate de tener instalados los siguientes programas en tu sistema:

1. **Docker**: Puede instalar Docker siguiendo las instrucciones de su sitio web oficial.[1]
2. **Node.js y npm**: Para el desarrollo del frontend se necesita Node.js y npm (viene incluído al instalar Node.js) que se puede instar desde su sitio web.[2]

A.2. Lanzamiento

Se recomienda una terminal basada en Linux. Si tu sistema operativo es Linux, utiliza su terminal. En caso en que utilices otro sistema operativo, se recomienda utilizar la terminal de Git Bash.

A.2.1. Clonar el repositorio

Primero, clona el repositorio de SmileLink desde GitHub:

[1] `https://docs.docker.com/get-docker/`
[2] `https://nodejs.org/en`

```
git clone https://github.com/scuadrosf/TFG-Software.git
cd TFG-Software
```

A.2.2. Construir el Contenedor Docker

Dentro del repositorio clonado, navega a la carpeta que contiene el archivo 'Dockerfile' y construye la imagen Docker:

```
cd allCode
docker build -t smilelink-image:latest -f Docker/Dockerfile .
```

A.2.3. Configurar Variables de Entorno

Configura las variables de entorno necesarias para la aplicación. Crea un archivo '*.env*' en la raíz del proyecto con el siguiente contenido, reemplazando los valores por los adecuados para tu configuración:

```
SPRING_MAIL_USERNAME=tu_usuario_de_correo
SPRING_MAIL_PASSWORD=tu_contrasena_de_correo
SPRING_DATASOURCE_URL=tu_url_de_base_de_datos
SPRING_DATASOURCE_USERNAME=tu_usuario_de_base_de_datos
SPRING_DATASOURCE_PASSWORD=tu_contrasena_de_base_de_datos
```

A.2.4. Ejecutar el Contenedor Docker

Ejecuta el contenedor Docker con las variables de entorno configuradas:

```
docker run -p 443:443 \
  -e SPRING_MAIL_USERNAME=$SPRING_MAIL_USERNAME \
  -e SPRING_MAIL_PASSWORD=$SPRING_MAIL_PASSWORD \
  -e SPRING_DATASOURCE_URL=$SPRING_DATASOURCE_URL \
  -e SPRING_DATASOURCE_USERNAME=$SPRING_DATASOURCE_USERNAME \
  -e SPRING_DATASOURCE_PASSWORD=$SPRING_DATASOURCE_PASSWORD \
  --name smilelink-container smilelink-image:latest
```

A.2.5. Verificar que la Aplicación Está Corriendo

Abre tu navegador web y navega a la dirección `https://localhost:443` para verificar que la aplicación está corriendo correctamente.

A.2.6. Parado y eliminación de contenedor

Si necesitas detener el contenedor Docker en cualquier momento, puedes usar el siguiente comando:

```
docker stop smilelink-container
```

Y para eliminarlo:

```
docker rm smilelink-container
```

Para eliminar las imágenes Docker creadas, puedes usar:

```
docker rmi smilelink-image
```

A.2.7. Notas adicionales

Lanzar la Aplicación por separado

Si deseas ejecutar la aplicación frontend y backend por separado, sigue estos pasos adicionales tras el clonado del repositorio:

Navega a la carpeta del frontend, instala las dependencias y ejecuta la aplicación Angular (debería estar disponible en `http://localhost:4200/`):

```
cd allCode/Frontend
npm install
npm start
```

Navega a la carpeta del backend, instala las dependencias y ejecuta la aplicación de Spring Boot (debería estar disponible en `https://localhost:443/`):

```
cd allCode/Backend/app
mvn clean install
mvn spring-boot:run
```

Ejecución con un perfil determinado

```
mvn spring-boot:run -Dspring-boot.run.profiles=test
```

Ejecución de pruebas

```
mvn test
```

B

Lanzamiento en AWS

Para lanzar la aplicación SmileLink en AWS, sigue los siguientes pasos detallados, mediante AWS CLI (Command Line Interface):

B.1. Prerrequisitos

1. **Cuenta de AWS**
2. **AWS CLI**: Debes tener instalada la CLI de AWS. Si no la tienes, puedes seguir las instrucciones de instalación en el sitio oficial de AWS.[1]
3. **Credenciales de AWS**: Configura tus credenciales de AWS ejecutando `aws configure` en tu terminal y proporcionando tu Access Key ID y Secret Access Key.[2]
4. **Par de claves**: Necesitas un par de claves (key pair) para conectar a tu instancia. Si no tienes uno, crea uno en la consola de AWS o con el comando:

```
1 aws ec2 create-key-pair --key-name MyKeyPair --query 'KeyMaterial
    ' --output text > MyKeyPair.pem
2 chmod 400 MyKeyPair.pem
```

[1] `https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html`

[2] `https://acortar.link/1gTa5q`[37]

B.2. Instancia EC2

B.2.1. Creación instancia EC2

Para lanzar una instancia EC2, necesitas conocer el ID de la AMI (Amazon Machine Image) que deseas usar, el tipo de instancia, el ID del grupo de seguridad y el ID de la subred (si aplica).

Obtener el ID de la AMI

Puedes obtener las AMIs disponibles con el comando:

```
aws ec2 describe-images --owners amazon --filters "Name=name,
    Values=amzn2-ami-hvm-2.0.*-x86_64-gp2" --query 'Images[*].[
    ImageId,Name]' --output text
```

Crear un grupo de seguridad (si no tienes uno):

```
aws ec2 create-security-group --group-name MySecurityGroup --
    description "My security group"
```

A continuación, agrega una regla al grupo de seguridad para permitir el acceso SSH (puerto 22):

```
aws ec2 authorize-security-group-ingress --group-name
    MySecurityGroup --protocol tcp --port 22 --cidr 0.0.0.0/0
```

B.2.2. Lanzar instancia EC2

```
aws ec2 run-instances --image-id [ID IMAGEN] --count 1 --instance
    -type t2.micro --key-name MyKeyPair --security-group-ids [ID
    GRUPO SEGURIDAD] --subnet-id [ID DE LA SUBRED]
```

El ID de la subred es opcional si no estás usando una VPC específica.

Obtener la IP pública de la instancia

Después de lanzar la instancia, obtén la IP pública para conectarte a ella:

```
aws ec2 describe-instances --instance-ids i-1234567890abcdef0 --
    query "Reservations[*].Instances[*].PublicIpAddress" --output
    text
```

Conectarse a la instancia EC2

Utiliza SSH para conectarte a tu instancia EC2:

```
ssh -i "MyKeyPair.pem" ec2-user@[IP OBTENIDA EN EL PASO ANTERIOR]
```

B.3. Instancia RDS

B.3.1. Creación instancia RDS

```
aws rds create-db-instance --db-instance-identifier mydatabase --
    allocated-storage 20 --db-instance-class db.t2.micro --engine
    mysql --master-username root --master-user-password password
    --port 3306
```

B.3.2. Lanzar intancia RDS

```
aws rds start-db-instance --db-instance-identifier mydatabase
```

B.4. Lanzar aplicación web

Ya estando dentro de la máquina de EC2, hay que configurar la instancia para poder lanzar la aplicación web. Sigue estos pasos:

Instalar Docker en la instancia EC2

```
sudo apt-get update -y
sudo apt-get install docker.io -y
sudo systemcl start docker
sudo systemcl enable docker
```

Creación variables de entorno

```
export SPRING_DATASOURCE_URL=[URL]
export SPRING_DATASOURCE_USERNAME=[USER DATABASE]
export SPRING_DATASOURCE_PASSWORD=[PASSWORD DATABASE]
export SPRING_MAIL_USERNAME=[EMAIL]
export SPRING_MAIL_PASSWORD=[PASSWORD]
```

Lanzar la imagen Docker

Es necesario que la instancia de la RDS esté ya encendida.

```
docker pull scuadrosf/smilelink:latest
docker run -d -p 443:443 \
    -e SPRING_MAIL_USERNAME=$SPRING_MAIL_USERNAME \
    -e SPRING_MAIL_PASSWORD="$SPRING_MAIL_PASSWORD" \
    -e SPRING_DATASOURCE_URL=$SPRING_DATASOURCE_URL \
    -e SPRING_DATASOURCE_USERNAME=$SPRING_DATASOURCE_USERNAME \
    -e SPRING_DATASOURCE_PASSWORD=$SPRING_DATASOURCE_PASSWORD \
    --name smilelink scuadrosf/smilelink:latest
```

Siguiendo estos pasos, deberías poder desplegar tu aplicación web en AWS utilizando la CLI.

C

Repositorio GitHub

C.1. Enlace al repositorio

El proyecto ha sido alojado en la plataforma *Github*.

El enlace es el siguiente: `https://github.com/scuadrosf/TFG-Software.git`