



Universidad Politécnica
de Madrid



**Escuela Técnica Superior de
Ingenieros Informáticos**

Grado en Ingeniería Informática

Trabajo Fin de Grado

**Desarrollo de un REST API y Frontend
para una Red Social**

Autor: Paje Alcántara, Santiago Antonio
Tutor(a): Lars-Åke Fredlund

Madrid, 04/2023

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado

Grado en Ingeniería Informática

Título: Desarrollo de un REST API y Frontend para una Red Social

04/2023

Autor: Paje Alcántara, Santiago Paje

Tutor:

Lars-Åke Fredlund

Lenguajes y sistemas DLSIIS

ETSI Informáticos

Universidad Politécnica de Madrid

Resumen

En la actualidad, las redes sociales se han convertido en un elemento esencial de la vida cotidiana, al permitir la interconexión entre individuos, la compartición de información y la participación en comunidades virtuales. Es innegable su impacto en la sociedad, lo cual implica la necesidad de soluciones tecnológicas eficientes y sólidas para su continuo crecimiento y evolución.

El presente trabajo se enfoca en el desarrollo de una API REST utilizando el lenguaje de programación Java y el *framework* Spring, junto con la implementación del Frontend utilizando Angular y MySQL como sistema de gestión de bases de datos. Para garantizar la calidad, escalabilidad y mantenibilidad del sistema, se han empleado las metodologías MVC (Modelo-Vista-Controlador) y el modelo cascada.

El objetivo principal de esta investigación ha sido el desarrollo de una API REST y un Frontend de alta calidad, que cumpla con los estándares de diseño y mejores prácticas, ofreciendo una experiencia de usuario fluida y una arquitectura escalable. Se ha prestado especial atención en asegurar la seguridad de la información, la eficiencia en el procesamiento de datos y una interacción intuitiva con la aplicación.

El enfoque adoptado se ha basado en el desarrollo iterativo, siguiendo la metodología del modelo cascada, lo cual ha permitido una planificación y estructuración eficiente del proyecto. Además, se han llevado a cabo pruebas exhaustivas para validar el correcto funcionamiento de las funcionalidades implementadas, asegurando de esta manera la calidad y confiabilidad del sistema.

Abstract

Presently, social media has emerged as an indispensable facet of everyday life, facilitating interpersonal connections, information sharing, and virtual community engagement. Undeniably, its impact on society necessitates the development of efficient and robust technological solutions to accommodate its ongoing growth and evolution.

The project focuses on the development of a Java-based REST API utilizing the Spring framework, accompanied by the Angular implementation of the Frontend and MySQL as the underlying database management system. To ensure maximum quality, scalability, and maintainability of the system, the methodologies of MVC (Model-View-Controller) and the waterfall model have been judiciously employed.

The principal objective of this project is to deliver a meticulously crafted REST API and Frontend that impeccably conform to industry-recognized design standards and best practices, thereby endowing users with a seamless and immersive experience while providing a foundation for a scalable architectural framework. Particular emphasis has been placed on the preservation of information security, the optimization of data processing efficiency, and the fostering of an intuitively interactive user interface.

To achieve this, an iterative development approach, firmly grounded in the principles of the waterfall model, has been dutifully embraced, enabling an intricately orchestrated project plan and structure. Complementary to this, comprehensive testing protocols have been meticulously conducted to validate the correct implementation of functional features, thus ensuring the impeccable quality and unwavering reliability of the system.

Tabla de contenidos

1	Introducción.....	1
1.1	Contexto y motivación	1
1.2	Objetivos del Trabajo.....	1
1.3	Metodología Utilizada	2
1.4	Plan del proyecto	3
2	Estado del arte	5
2.1	Redes sociales y su importancia	5
2.2	Tecnologías utilizadas en el desarrollo del API REST	6
2.3	Tecnologías utilizadas para el desarrollo del Frontend.....	8
2.4	Tecnologías utilizadas restantes	9
3	Análisis de requisitos.....	11
3.1	Especificación de Requisitos.....	11
3.1.1	Requisitos Software.....	11
3.1.2	Requisitos de Usuario	13
3.2	Casos de uso	15
4	Diseño del sistema.....	18
4.1	Diseño de la arquitectura	18
4.2	Diseño de la base de datos	19
4.2.1	Detalles del diseño	19
4.2.2	Diagrama Entidad-Relación	20
4.3	Diseño de la API REST.....	20
4.3.1	Documentación de la API y endpoints	21
4.3.2	Seguridad	28
4.3.3	Manejo de errores y excepciones	29
4.4	Diseño del Frontend	29
4.4.1	Diseño pantallas	29
4.4.2	Diseño y recursos gráficos utilizados.....	33
4.4.3	Estructura del proyecto Angular	33
4.4.4	Documentación de las rutas de la aplicación	35
5	Implementación.....	37
5.1	Desarrollo de la API REST con Java y Spring Boot.....	37
5.1.1	Configuración del proyecto.....	37
5.1.2	Desarrollo del Controlador	42
5.1.3	Desarrollo de los Servicios.....	46
5.1.3.1	UserService.....	47
5.1.3.2	EmailService	60
5.1.4	Integración con la base de datos	62
5.1.5	Excepciones	64

5.2	Desarrollo del Frontend con Angular	65
5.2.1	Configuración del proyecto.....	65
5.2.2	Implementación de las vistas y componentes	65
5.2.2.1	Módulo “auth.module”	65
5.2.2.2	Módulo “home.module”	78
5.2.2.3	Módulo “shared.module”	92
5.3	Pruebas de uso.....	96
6	Conclusiones	110
6.1	Logro de objetivos.....	110
6.2	Aprendizajes y dificultades encontradas	111
6.3	Evaluación del código.....	112
7	Análisis de Impacto	113
7.1	Impacto Personal.....	113
7.2	Impacto Empresarial	113
7.3	Impacto Social.....	114
7.4	Impacto Económico	114
7.5	Impacto Medioambiental	114
7.6	Impacto Cultural	114
7.7	Referencia a los Objetivos de Desarrollo Sostenible (ODS)	114
7.8	Consideración del Impacto en las Decisiones Tomadas.....	115
8	Bibliografía	116
9	Índice de ilustraciones.....	118

1 Introducción

1.1 Contexto y motivación

Como es bien sabido, las redes sociales se han convertido en una parte fundamental de la sociedad humana en los últimos años. Actualmente, millones de personas utilizan las redes sociales para comunicarse con otras personas, para informarse o para consumir contenido de entretenimiento.

Teniendo en cuenta la gran importancia que tienen las redes sociales en la sociedad actual, resulta realmente interesante y beneficioso tener conocimiento sobre su desarrollo (implementación, tecnologías, técnicas usadas, etc.).

Además, el desarrollo de este trabajo supone la adquisición de ciertos conocimientos sobre desarrollo de aplicaciones web, los cuales son tanto realmente interesantes como ampliamente demandados.

Tanto el desarrollo de redes sociales como el desarrollo de aplicaciones web son las principales motivaciones que han hecho que el desarrollo de este trabajo me haya resultado francamente fascinante.

1.2 Objetivos del Trabajo

Los objetivos planteados en este Trabajo son los siguientes:

Desarrollar una Interfaz de Programación de Aplicaciones (API) REST completa y altamente funcional, diseñada específicamente para gestionar y controlar todas las diversas funcionalidades y características esenciales asociadas a una plataforma de red social moderna y dinámica.

Crear y desarrollar un Frontend sofisticado y estéticamente agradable, que brinde a los usuarios de la red social una experiencia interactiva, intuitiva y cautivadora. El enfoque se centra en la implementación de una interfaz de usuario que combine una navegación fluida, un diseño visualmente atractivo y una usabilidad excepcional, con el objetivo de proporcionar a los usuarios una plataforma atractiva y amigable para acceder, explorar y aprovechar al máximo todas las funcionalidades y características disponibles en la red social.

Lograr una integración eficiente y sólida entre la API REST y el Frontend, asegurando una comunicación continua y segura entre ambas partes. Se prioriza la implementación de un mecanismo de interacción fluida y sin interrupciones, que permita a los usuarios aprovechar todas las funcionalidades de la red social de manera ágil y sin contratiempos. Para lograr esto, se aplicarán prácticas de desarrollo modernas y sólidas, haciendo uso de tecnologías avanzadas que garanticen la transmisión segura de datos y la sincronización adecuada entre el Backend y el Frontend. Asimismo, se establecerán estándares de seguridad rigurosos para proteger la integridad y la privacidad de los usuarios, asegurando una experiencia segura y confiable en el uso de la red social.

Durante el proceso de desarrollo, se hará hincapié en la aplicación rigurosa de buenas prácticas de ingeniería de software con el objetivo de garantizar un código limpio, eficiente y fácilmente mantenible. Se emplearán patrones de diseño reconocidos y ampliamente adoptados, como el patrón MVC (Modelo-Vista-Controlador) o el patrón de Inyección de Dependencias, para lograr una arquitectura escalable y modular. Además, se realizarán pruebas unitarias exhaustivas para asegurar la calidad y la robustez del software desarrollado, detectando y corrigiendo posibles fallos o errores en cada una de las partes del sistema.

Asimismo, se otorgará una especial atención a la documentación adecuada del código y de los procesos involucrados en el desarrollo. Se generarán documentos claros y detallados que describan la estructura, el funcionamiento y las diferentes funcionalidades del sistema, lo que facilitará su mantenimiento futuro y permitirá a otros desarrolladores comprender y colaborar en el proyecto de manera efectiva. Además, se fomentará el uso de comentarios explicativos en el código fuente para mejorar su legibilidad y comprensión.

En resumen, se hará un enfoque consciente en la aplicación de buenas prácticas de ingeniería de software, desde el diseño hasta la documentación, con el fin de asegurar la calidad, la escalabilidad y la mantenibilidad del sistema desarrollado.

1.3 Metodología Utilizada

La metodología empleada en el desarrollo de este proyecto es el modelo en cascada, una metodología tradicional que sigue un enfoque secuencial y lineal. Este modelo se divide en tres fases principales, cada una con su propio conjunto de actividades y objetivos específicos.

La primera fase es la de análisis y planificación, donde se recopilan y documentan exhaustivamente los requisitos del proyecto, se definen los objetivos y se establecen las metas a alcanzar. En esta etapa, se lleva a cabo un análisis detallado de las necesidades de los usuarios finales y se definen los entregables esperados. Además, se realiza una planificación cuidadosa de los recursos, el cronograma y los costos asociados al proyecto.

La segunda fase es la de diseño, donde se crea una estructura y una arquitectura sólida para el sistema. Se definen los componentes, módulos y subsistemas necesarios, así como las interfaces entre ellos. Además, se desarrolla un diseño detallado de la base de datos y se establecen los estándares y las pautas de codificación. Esta fase también implica la creación de diagramas de flujo, diagramas de clase y otros artefactos que ayudan a comprender y visualizar la estructura del sistema.

La tercera y última fase es la de implementación y pruebas, donde se lleva a cabo la codificación del software y se realizan pruebas para verificar su correcto funcionamiento. Durante esta etapa, se siguen los estándares de codificación establecidos y se realizan pruebas unitarias para asegurar la calidad del software desarrollado. Además, se llevan a cabo pruebas funcionales y de rendimiento para garantizar que el sistema cumpla con los requisitos establecidos.

En resumen, en este proyecto se ha utilizado el modelo en cascada, dividido en tres fases distintas, para garantizar un enfoque secuencial y estructurado en el desarrollo del software. Además, se ha implementado el *git* controlador de versiones, proporcionando un marco de trabajo claro y ordenado para la gestión de los cambios en el código fuente. Estas metodologías combinadas brindan un enfoque sólido y eficiente para el desarrollo del proyecto.

1.4 Plan del proyecto

A continuación, se presenta el plan del proyecto para el desarrollo del trabajo:

Elección de las tecnologías: Investigar y evaluar diferentes tecnologías y herramientas que se adapten a los requisitos del trabajo. Seleccionar las tecnologías más adecuadas para el desarrollo del proyecto, como lenguajes de programación, *frameworks*, bases de datos, entre otros.

Formación previa: Identificar las brechas de conocimiento existentes y determinar qué habilidades y competencias deben ser adquiridas. Buscar recursos de aprendizaje, como cursos en línea, tutoriales o documentación oficial, para adquirir los conocimientos necesarios.

Análisis y planificación: Definir los requisitos del proyecto en detalle, incluyendo funcionalidades, objetivos y restricciones. Realizar un análisis de viabilidad y factibilidad para determinar la viabilidad técnica y temporal del proyecto. Establecer un cronograma de trabajo con plazos claros para cada etapa del trabajo.

Diseño de la arquitectura: Definir la arquitectura general del proyecto, incluyendo la estructura de carpetas, la interacción entre los componentes y la organización de la base de datos. Diseñar los diferentes módulos y componentes del proyecto, teniendo en cuenta la escalabilidad, la reutilización del código y las buenas prácticas de diseño.

Implementación del sitio web: Desarrollar el sitio web siguiendo el diseño y la arquitectura definidos anteriormente. Escribir el código, crear las funcionalidades, integrar las tecnologías y desarrollar las interfaces de usuario. Realizar pruebas periódicas para verificar el correcto funcionamiento del sitio web a medida que se avanza en la implementación y poder mejorarlo.

Pruebas y mejora: Realizar pruebas exhaustivas para identificar posibles errores, fallos de funcionamiento o problemas de rendimiento.

Documentación de la memoria: Documentar todo el proceso de desarrollo, incluyendo los pasos seguidos, las decisiones tomadas y los resultados obtenidos. Elaborar una memoria técnica que describa de manera detallada el proyecto, su objetivo, la metodología utilizada, las tecnologías empleadas y los resultados alcanzados. Revisar y editar la documentación para asegurar que sea clara, concisa y coherente.

Diagrama de Gantt

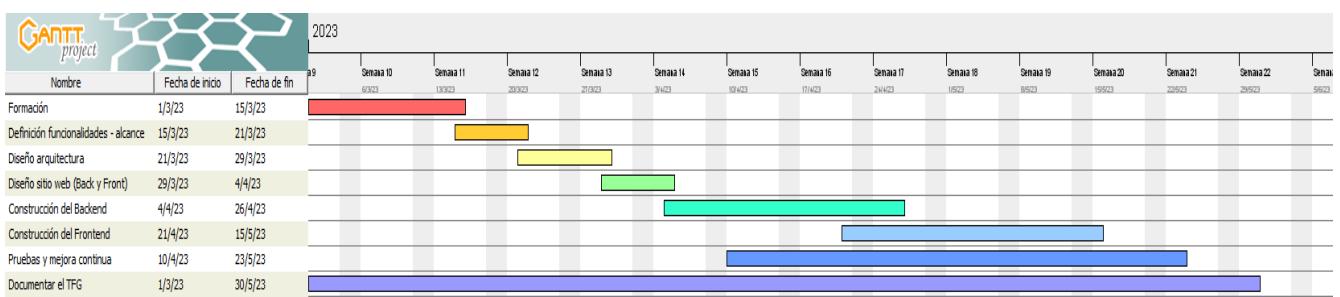


Ilustración 1. Diagrama de Gantt

2 Estado del arte

2.1 Redes sociales y su importancia

Las redes sociales se han convertido en un fenómeno global, con cifras asombrosas que reflejan su popularidad y la creciente dependencia de las personas en estas plataformas. Por ejemplo, según estudios recientes [1], se estima que más de 4.76 mil millones de personas utilizan las redes sociales a nivel mundial, lo que representa casi el 60% de la población global. Esta cifra es un testimonio del alcance masivo y la penetración profunda que tienen las redes sociales en la sociedad moderna.

Además del impresionante número de usuarios, es importante destacar el impacto multifacético de las redes sociales en diversas áreas. En el ámbito de la comunicación, las redes sociales han transformado la forma en que las personas se conectan, interactúan y comparten información. La capacidad de comunicarse instantáneamente con personas de todo el mundo ha eliminado barreras geográficas y ha fomentado la creación de comunidades globales.

En el ámbito del marketing, las redes sociales se han convertido en una herramienta fundamental para las empresas y los profesionales del marketing. Permiten la segmentación precisa del público objetivo y brindan la oportunidad de promocionar productos y servicios de manera efectiva. Además, las redes sociales ofrecen una plataforma para el análisis de datos y la recopilación de información sobre las preferencias y comportamientos de los usuarios, lo que facilita la personalización y la adaptación de las estrategias de marketing.

En el ámbito del entretenimiento, las redes sociales han dado lugar a una nueva forma de consumo de contenido. Los usuarios pueden disfrutar de una amplia variedad de contenido multimedia, desde videos virales, hasta transmisiones en vivo, todo ello al alcance de sus manos. Las redes sociales también han permitido que los creadores de contenido independientes encuentren una audiencia global y establezcan nuevas formas de monetización a través de patrocinios y publicidad.

Sin embargo, el crecimiento y la evolución de las redes sociales también plantean desafíos y oportunidades. Temas como la privacidad, la seguridad de los datos y la proliferación de noticias falsas son cuestiones que deben abordarse de manera responsable. Asimismo, el desarrollo de nuevos modelos de negocio, como el comercio electrónico en redes sociales y la monetización de *influencers*, brindan oportunidades para la innovación y la creación de empleo.

En resumen, el análisis del estado actual de las redes sociales revela su indiscutible relevancia en la sociedad actual. A través de estadísticas impactantes y ejemplos concretos, se ha demostrado su alcance masivo y su impacto en áreas como la comunicación, el marketing, el entretenimiento y la interacción social. A medida que las redes sociales continúan evolucionando, es crucial abordar los desafíos y aprovechar las oportunidades que surgen, asegurando un uso responsable y beneficioso para los usuarios y la sociedad en general.

2.2 Tecnologías utilizadas en el desarrollo del API REST

En esta sección, se detallarán las tecnologías fundamentales utilizadas en el desarrollo del API REST. Estas tecnologías desempeñan un papel clave en la implementación de servicios web eficientes y escalables. A continuación, se presentan las principales tecnologías utilizadas en este trabajo:

Protocolo HTTP: El protocolo de transferencia de hipertexto (HTTP) [2] es el estándar utilizado para la comunicación entre el cliente y el servidor en la web. Se destacan los principales métodos de HTTP, como GET, POST, PUT y DELETE, los cuales permiten realizar operaciones de lectura, escritura, actualización y eliminación de datos en el servidor. Estos métodos son fundamentales en el diseño y la implementación de un API REST.

JSON: JSON (JavaScript Object Notation) [3] es un formato ligero y legible para el intercambio de datos. Es ampliamente utilizado en el desarrollo de APIs REST debido a su simplicidad y su capacidad para representar datos estructurados. JSON permite enviar y recibir datos de manera eficiente y es compatible con la mayoría de los lenguajes de programación y *frameworks* web.



Ilustración 2. JSON logo

Spring: Spring [4] es un *framework* de desarrollo de aplicaciones Java ampliamente utilizado en el desarrollo de APIs REST. Proporciona un conjunto completo de características y herramientas que facilitan la creación de servicios web robustos y escalables. Spring ofrece soporte para la implementación de la arquitectura RESTful y simplifica tareas como la gestión de la configuración, la inyección de dependencias y el enrutamiento de las solicitudes HTTP.



Ilustración 3. Spring logo

Java: Java [5] es un lenguaje de programación versátil y ampliamente utilizado en el desarrollo de aplicaciones. En el contexto de las APIs REST, Java proporciona un entorno sólido para la implementación de servicios web y la manipulación de datos. Su gran variedad y calidad de bibliotecas y *frameworks* facilitan el desarrollo eficiente y mantenible de APIs REST.

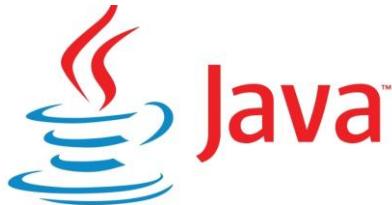


Ilustración 4. Java logo

Postman: Postman [6] es una herramienta popular utilizada para probar y documentar APIs. Permite enviar solicitudes HTTP a la API REST y recibir las respuestas correspondientes, lo que facilita la verificación del funcionamiento correcto de la API. Postman ofrece una interfaz intuitiva y funcionalidades avanzadas, como la creación de colecciones de solicitudes y la automatización de pruebas, lo que agiliza el proceso de desarrollo y depuración de la API.



Ilustración 5. Postman logo

2.3 Tecnologías utilizadas para el desarrollo del Frontend

En el desarrollo del Frontend de este proyecto, se han utilizado diversas tecnologías que permiten crear una interfaz intuitiva y atractiva para los usuarios. A continuación, se describen las principales tecnologías utilizadas:

Angular: Angular [7] es un *framework* de desarrollo de aplicaciones web de código abierto desarrollado por Google. Es ampliamente utilizado en la creación de aplicaciones de una sola página (Single Page Applications - SPAs) y ofrece un conjunto completo de herramientas y características para construir interfaces interactivas. Angular utiliza el lenguaje de programación TypeScript y proporciona una arquitectura basada en componentes que facilita la reutilización y la modularidad del código.



Ilustración 6. Angular logo

HTML: HTML (HyperText Markup Language) [8] es el lenguaje de marcado estándar utilizado para estructurar y presentar el contenido en la web. En el desarrollo del Frontend, HTML se utiliza para definir la estructura y el esqueleto de las páginas web. Se emplean elementos HTML para crear elementos como encabezados, párrafos, tablas, formularios y enlaces, entre otros.



Ilustración 7. HTML logo

CSS: CSS (Cascading Style Sheets) [9] es un lenguaje utilizado para describir la presentación y el estilo de los documentos HTML. Se utiliza para definir el diseño, los colores, las fuentes, los efectos visuales y otros aspectos de la apariencia de la interfaz de usuario. CSS permite crear diseños atractivos y responsivos, asegurando una presentación coherente en diferentes dispositivos y tamaños de pantalla.



Ilustración 8. CSS logo

TypeScript: TypeScript [10] es un lenguaje de programación que se basa en JavaScript, pero añade características adicionales, como el tipado estático y la orientación a objetos. En el desarrollo del Frontend, TypeScript se utiliza junto con Angular para escribir el código de la lógica de la aplicación y definir los componentes, servicios y modelos.



Ilustración 9. TypeScript logo

Angular CLI: Angular CLI (Command Line Interface) [11] es una herramienta de línea de comandos que facilita la creación, el desarrollo y la compilación de proyectos de Angular. Proporciona comandos predefinidos para generar componentes, servicios, módulos y otras entidades de Angular, lo que acelera el proceso de desarrollo y automatiza tareas comunes.

DevTools del navegador: Las DevTools [12] son herramientas integradas en los navegadores web modernos que permiten inspeccionar, depurar y analizar el código y los elementos de una página web en tiempo real. Estas herramientas proporcionan una interfaz gráfica que permite ver la estructura del DOM, realizar seguimiento de las solicitudes HTTP, depurar el código JavaScript y realizar cambios en tiempo real para probar y solucionar problemas en el Frontend.

2.4 Tecnologías utilizadas restantes

MySQL: es un sistema [13] de gestión de bases de datos relacional de código abierto ampliamente utilizado. Proporciona un entorno seguro y confiable para almacenar, organizar y administrar grandes volúmenes de datos de manera eficiente. MySQL es conocido por su rendimiento rápido, escalabilidad y por ser ampliamente utilizado. MySQL utiliza el lenguaje SQL (*Structured Query Language*) para interactuar con la base de datos y ofrece numerosas características, como la replicación, la alta disponibilidad y la seguridad avanzada.



Ilustración 10. MySQL logo

GitHub: es una plataforma [14] de desarrollo colaborativo basada en la web que permite a los desarrolladores trabajar en proyectos de software de manera conjunta y controlar las versiones de su código fuente. Proporciona un sistema de control de versiones distribuido llamado Git, que permite rastrear y gestionar los cambios realizados en los archivos de un proyecto a lo largo del tiempo. Es ampliamente utilizado en la comunidad de desarrollo de software y brinda un entorno colaborativo y transparente para el desarrollo de proyectos.



Ilustración 11. GitHub logo



Ilustración 12. Git logo

3 Análisis de requisitos

El análisis de requisitos es una fase esencial en el desarrollo de cualquier proyecto. En esta sección, se realizará un análisis exhaustivo de los requisitos, con el objetivo de comprender a fondo las necesidades y expectativas del sistema o aplicación a desarrollar. El análisis de requisitos [15] nos permite establecer una base sólida para el diseño y la implementación, garantizando que el producto final cumpla con las necesidades de los usuarios y los objetivos del proyecto. A través de esta sección, exploraremos en detalle los requisitos, definiendo claramente las características y funcionalidades que se esperan del sistema.

3.1 Especificación de Requisitos

3.1.1 Requisitos Software

Los requisitos software [16] son declaraciones documentadas de las necesidades y restricciones que debe cumplir un sistema, software o producto para satisfacer los objetivos. Estos requisitos describen las funcionalidades del software que se va a desarrollar. Los requisitos software del trabajo son los siguientes:

SR-1: Registro de Usuario

SR-1.1: El sistema debe permitir a los usuarios registrarse proporcionando un nombre de usuario único, su nombre completo, su correo electrónico y una contraseña para su cuenta.

SR-1.2: El sistema debe validar que el nombre de usuario sea único en el sistema.

SR-1.3: El sistema debe validar el formato y la validez del correo electrónico proporcionado por el usuario.

SR-1.4: El sistema debe validar el tamaño mínimo de la contraseña proporcionada por el usuario.

SR-1.5: El sistema debe almacenar de forma segura la información del usuario en la base de datos.

SR-2: Verificación de Cuenta de Usuario

SR-2.1: El sistema debe enviar un correo electrónico de verificación al usuario después del registro para autenticar su dirección de correo electrónico y verificar su cuenta.

SR-2.2: El sistema debe generar un enlace único en el correo electrónico de verificación para que el usuario pueda confirmar su cuenta.

SR-3: Inicio de Sesión de Usuario

SR-3.1: El sistema debe permitir a los usuarios iniciar sesión utilizando su nombre de usuario y contraseña.

SR-3.2: El sistema debe autenticar las credenciales del usuario y proporcionar acceso seguro a la cuenta del usuario.

SR-4: Publicación de Fotografía

SR-4.1: El sistema debe permitir a los usuarios publicar fotografías en el sistema.

SR-4.2: El sistema debe permitir a los usuarios agregar un encabezado opcional a la publicación de la fotografía.

SR-5: Visualización de Publicaciones en la Página Principal

SR-5.1: El sistema debe mostrar en la página principal las publicaciones de los usuarios seguidos por el usuario que ha iniciado sesión.

SR-6: Gestión de Seguimiento de Usuarios

SR-6.1: El sistema debe permitir a los usuarios seguir a otro usuario.

SR-6.2: El sistema debe permitir a los usuarios dejar de seguir a otro usuario.

SR-6.4: El sistema debe mantener un registro de los seguidores y seguidos de cada usuario.

SR-7: Modificación de Información de Usuario

SR-7.1: El sistema debe permitir a los usuarios modificar su nombre de usuario, contraseña, nombre completo.

SR-7.2: El sistema debe validar las modificaciones realizadas por el usuario antes de guardar los cambios.

SR-8: Dejar de Seguir a un Usuario

SR-8.1: El sistema debe permitir a los usuarios dejar de seguir a otro usuario al cual siguen.

SR-9: Cambio de Contraseña

SR-9.1: El sistema debe permitir a los usuarios cambiar su contraseña en caso de que se les haya olvidado.

SR-9.2: El sistema debe enviar un correo electrónico al usuario con un enlace de acceso al cambio de contraseña.

SR-9.3: El sistema debe validar la autenticidad del enlace de cambio de contraseña antes de permitir al usuario modificar su contraseña.

SR-10: Funcionalidad de "Me Gusta"

SR-10.1: El sistema debe permitir a los usuarios dar "Me gusta" a una publicación.

SR-10.2: El sistema debe mantener un registro de los usuarios que han dado "Me gusta" a una publicación.

SR-11: Búsqueda de Usuarios

SR-11.1: El sistema debe permitir a los usuarios buscar a otros usuarios por su nombre de usuario.

SR-12: Visualización de Perfil de Usuario

SR-12.1: El sistema debe permitir a los usuarios visualizar su propio perfil, donde se muestra el nombre de usuario, los seguidores, los seguidos, el número de publicaciones y sus publicaciones.

SR-12.2: El sistema debe permitir a los usuarios visualizar el perfil de cualquier usuario.

SR-13: Cierre de Sesión

SR-13.1: El sistema debe proporcionar a los usuarios la opción de cerrar sesión.

3.1.2 Requisitos de Usuario

Los requisitos de usuario [16] son declaraciones de las necesidades, expectativas y restricciones de los usuarios en relación con un sistema o software determinado. Estos requisitos se centran en los objetivos, funcionalidades y características que los usuarios desean que el sistema o software cumpla para satisfacer sus necesidades. Los requisitos de usuario del trabajo son los siguientes:

UR-1: Registro de Usuario

UR-1.1: El sistema permitirá a un usuario registrarse proporcionando un nombre de usuario único, su nombre completo, su correo electrónico y una contraseña para su cuenta.

UR-2: Verificación de Cuenta de Usuario

UR-2.1: Despues de registrarse, el usuario recibirá un correo electrónico de verificación para autenticar su dirección de correo electrónico y verificar su cuenta.

UR-3: Inicio de Sesión de Usuario

UR-3.1: Un usuario podrá iniciar sesión en el sistema utilizando su nombre de usuario y contraseña.

UR-4: Publicación de Fotografía

UR-4.1: Un usuario podrá publicar una fotografía en el sistema, con la opción de agregar un encabezado opcional a la publicación.

UR-5: Página Principal de Publicaciones

UR-5.1: El sistema mostrará en la página principal las publicaciones de los usuarios seguidos por el usuario que ha iniciado sesión.

UR-6: Seguimiento de Usuarios

UR-6.1: Un usuario podrá dejar de seguir a otro usuario.

UR-6.2: Un usuario podrá seguir a otro usuario.

UR-7: Modificación de Información de Usuario

UR-7.1: Un usuario podrá modificar su nombre de usuario, contraseña, nombre completo.

UR-8: Dejar de Seguir a un Usuario

UR-8.1: Un usuario podrá dejar de seguir a otro usuario al cual sigue.

UR-9: Restablecimiento de Contraseña

UR-9.1: Un usuario podrá restablecer su contraseña en caso de haberla olvidado, proporcionando su correo electrónico y siguiendo el enlace enviado a su dirección de correo electrónico.

UR-10: Interacción con Publicaciones

UR-10.1: Un usuario podrá dar "Me Gusta" a una publicación.

UR-11: Búsqueda de Usuarios

UR-11.1: Un usuario podrá buscar a otro usuario por su nombre de usuario.

UR-12: Perfil de Usuario

UR-12.1: Un usuario podrá visualizar su propio perfil, que mostrará su nombre de usuario, el número de seguidores, el número de seguidos y el número de publicaciones realizadas.

UR-12.2: Un usuario podrá visualizar el perfil de cualquier usuario.

UR-13: Cierre de Sesión

UR-13.1: Un usuario podrá cerrar sesión en el sistema.

3.2 Casos de uso

Los casos de uso [16] son una técnica utilizada en la ingeniería de software para capturar y describir las interacciones entre los usuarios y el sistema o software en desarrollo. Representan las acciones o funcionalidades que un usuario puede realizar en el sistema y proporcionan una visión clara de cómo se utiliza el sistema desde la perspectiva del usuario. Los casos de uso se representan mediante descripciones detalladas que especifican los pasos, entradas y salidas involucrados en cada interacción. Los casos de uso del trabajo son los siguientes:

Registro de Usuario

Descripción: Permite a un usuario registrarse en el sistema.

Actores: Usuario

Flujo principal:

El usuario proporciona sus datos de registro.

El sistema valida los datos ingresados.

El sistema crea una cuenta para el usuario.

El sistema envía un correo de verificación al usuario.

El usuario verifica su cuenta a través del enlace de verificación.

Inicio de Sesión

Descripción: Permite a un usuario iniciar sesión en el sistema.

Actores: Usuario

Flujo principal:

El usuario ingresa su nombre de usuario y contraseña.

El sistema valida las credenciales del usuario.

El sistema proporciona acceso a la cuenta del usuario.

Publicación de Fotografía

Descripción: Permite a un usuario publicar una fotografía en el sistema.

Actores: Usuario

Flujo principal:

El usuario selecciona una fotografía para publicar.

El usuario puede agregar un encabezado opcional a la publicación.

El sistema guarda la publicación en el sistema.

Seguir a un Usuario

Descripción: Permite a un usuario seguir a otro usuario en el sistema.

Actores: Usuario

Flujo principal:

El usuario busca al usuario que desea seguir.

El usuario sigue al otro usuario.

Dejar de Seguir a un Usuario

Descripción: Permite a un usuario dejar de seguir a otro usuario en el sistema.

Actores: Usuario

Flujo principal:

El usuario busca al usuario que desea dejar de seguir.

El usuario deja de seguir al otro usuario.

Modificación de Información de Usuario

Descripción: Permite a un usuario modificar su información personal en el sistema.

Actores: Usuario

Flujo principal:

El usuario accede a el perfil de su cuenta.

El usuario modifica su nombre de usuario, contraseña, nombre completo o estado de cuenta.

El sistema valida los cambios y actualiza la información del usuario.

Dejar de Seguir a un Usuario

Descripción: Permite a un usuario dejar de seguir a otro usuario en el sistema.

Actores: Usuario

Flujo principal:

El usuario busca al usuario que desea dejar de seguir.

El sistema elimina la relación de seguimiento entre los usuarios.

Cambio de Contraseña

Descripción: Permite a un usuario cambiar su contraseña en caso de olvido.

Actores: Usuario

Flujo principal:

El usuario solicita cambiar su contraseña debido al olvido.

El usuario ingresa su correo electrónico.

El sistema envía un correo electrónico con un enlace de cambio de contraseña al correo electrónico registrado del usuario.

El usuario accede al enlace de cambio de contraseña y establece una nueva contraseña.

4 Diseño del sistema

En esta sección, nos adentraremos en el diseño del sistema, abarcando tanto la arquitectura general como los detalles específicos de la base de datos, la API REST y el Frontend.

4.1 Diseño de la arquitectura

El sistema se basa en una arquitectura MVC (Modelo-Vista-Controlador) [17] de tres capas, que consta de la capa de presentación (Vista), la capa de lógica de negocio (Controlador) y la capa de almacenamiento de datos (Modelo). Esta arquitectura permite una separación clara de responsabilidades y facilita la escalabilidad y el mantenimiento del sistema.

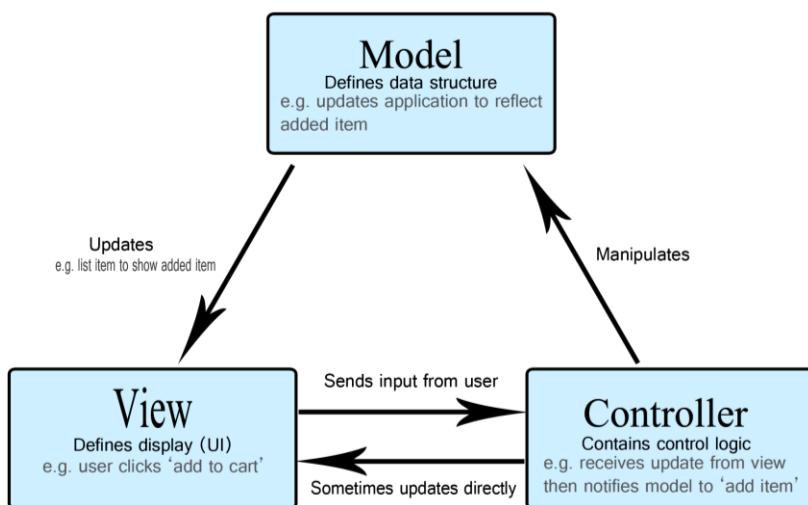


Ilustración 13. Diagrama MVC

En la capa de presentación (Vista), se define la interfaz de usuario y se maneja la interacción con el usuario. Aquí se diseñan y desarrollan las diferentes pantallas y componentes visuales que forman parte del sistema.

En la capa de lógica de negocio (Controlador), se implementan las reglas y la lógica que gobiernan el funcionamiento del sistema. Aquí se procesan las solicitudes del usuario, se realizan las validaciones necesarias y se coordinan las acciones entre la capa de presentación y la capa de almacenamiento de datos.

En la capa de almacenamiento de datos (Modelo), se maneja el acceso y la gestión de los datos del sistema. Aquí se definen las estructuras de datos, se realiza la comunicación con la base de datos y se ejecutan las operaciones de lectura y escritura.

4.2 Diseño de la base de datos

4.2.1 Detalles del diseño

En el diseño de la base de datos, se han definido las siguientes tablas: Usuario, Publicación, *Like*, *Follow*, Hobby y *user_hobby*. Estas tablas representan las entidades principales de la red social y sus relaciones.

La tabla "**Usuario**" almacena la información de los usuarios, mientras que la tabla "Publicación" guarda los detalles de las publicaciones realizadas por los usuarios. La relación entre Usuario y Publicación es de uno a muchos (1:n), ya que un usuario puede tener varias publicaciones.

La tabla "**Like**" registra los "Me Gusta" dados a las publicaciones, estableciendo una relación uno a muchos (1: n) con la tabla Usuario. Esto significa que un usuario puede dar "Me Gusta" a varias publicaciones.

La tabla "**Follow**" permite que los usuarios sigan a otros usuarios. También hay una relación muchos a muchos (m: n) entre Usuario y Usuario en este caso, lo que significa que un usuario puede seguir a varios usuarios y también puede ser seguido por varios usuarios.

La tabla "**Hobby**" almacena los diferentes hobbies o intereses disponibles. La relación entre Usuario y Hobby es de muchos a muchos (m:n). Para representar esta relación, se utiliza una tabla de relación llamada "*user_hobby*" que establece la conexión entre Usuario y Hobby.

Además del diseño de las tablas y las relaciones, se ha considerado la creación de índices en la base de datos para mejorar el rendimiento y la eficiencia en las consultas. Los índices [18] son estructuras que permiten acelerar la búsqueda y recuperación de datos, agilizando las operaciones de lectura y optimizando la velocidad de respuesta del sistema.

En el diseño de la base de datos de la red social, se han identificado los campos que se utilizan con mayor frecuencia en las consultas, como los identificadores de usuario, los identificadores de publicaciones o los campos utilizados en la búsqueda de usuarios. Estos campos se han considerado candidatos para la creación de índices.

En cuanto a la base de datos, se ha utilizado MySQL como sistema de gestión de bases de datos para almacenar y manipular los datos de la red social. MySQL es una tecnología ampliamente utilizada y reconocida por su rendimiento, confiabilidad y facilidad de uso en entornos web.

Con este diseño de base de datos y el uso de MySQL, se busca asegurar una estructura eficiente y coherente para almacenar y gestionar la información de la red social, permitiendo un acceso rápido y seguro a los datos.

4.2.2 Diagrama Entidad-Relación

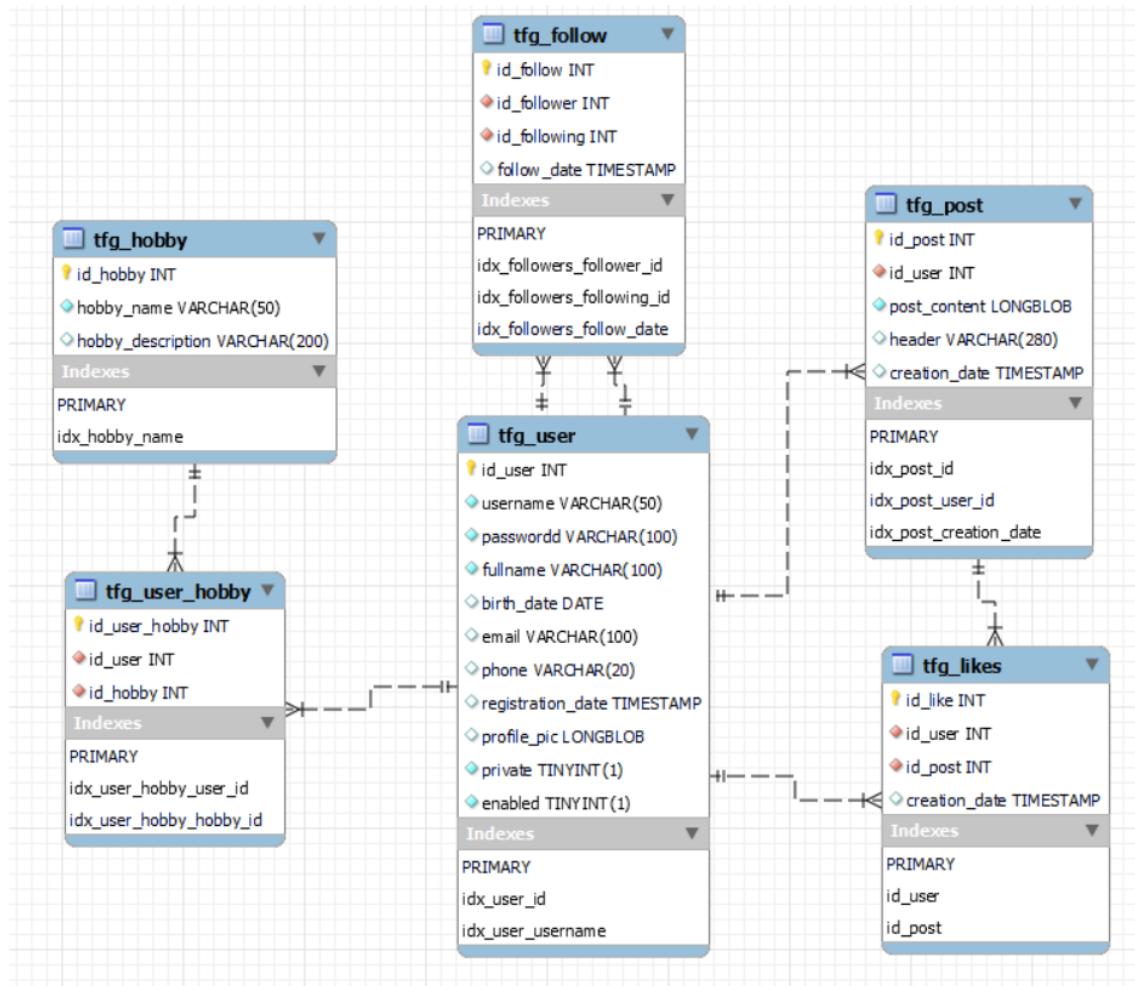


Ilustración 14. Diagrama de Entidad-Relación

4.3 Diseño de la API REST

La API REST juega un papel fundamental en la implementación de la funcionalidad de la red social, permitiendo la interacción con la base de datos y ofreciendo los servicios necesarios para el correcto funcionamiento del sistema. En este apartado, se aborda en detalle el diseño de esta API, considerando diferentes aspectos clave.

4.3.1 Documentación de la API y endpoints

Método: register

Endpoint: POST /tfg/register

Descripción: Permite registrar un nuevo usuario en el sistema.

Parámetros:

request: Objeto RequestRegisterVO que contiene la información del usuario a registrar.

Excepciones: EmailUsedException, UserUsedException, IOException

Respuestas:

Retorna un objeto ResponseRegisterVO con los detalles del registro.

201 CREATED: Registro exitoso.

409 CONFLICT

Método: verifyEmail

Endpoint: GET /tfg/verify-email

Descripción: Verifica la dirección de correo electrónico de un usuario registrado.

Parámetros:

token: Token de verificación enviado por correo electrónico.

Excepciones: UserNotFoundException, UserAlreadyEnabledException, TokenExpiredException

Respuestas:

Retorna un objeto ResponseVerifyEmail con el resultado de la verificación

200 OK: Verificación exitosa. 404 NOT FOUND 409 CONFLICT

Método: login

Endpoint: POST /tfg/login

Descripción: Permite que un usuario inicie sesión en el sistema.

Parámetros:

request: Objeto RequestLoginVO que contiene las credenciales del usuario para iniciar sesión.

Excepciones: UserNotFoundException, IncorrectPwdException

Respuestas:

Retorna el ID de usuario y el encabezado de respuesta en un ResponseEntity.

200 OK: Inicio de sesión exitoso.

404 NOT FOUND

401 UNAUTHORIZED

Método: forgotPass

Endpoint: POST /tfg/forgot-password

Descripción: Permite a un usuario solicitar restablecer su contraseña.

Parámetros:

emailRequest: Cadena de texto que contiene la dirección de correo electrónico del usuario.

Excepciones: UserNotFoundException

Respuestas:

200 OK: Solicitud de restablecimiento de contraseña exitosa.

404 NOT FOUND

Método: authToken

Endpoint: GET /tfg/authenticate-token

Descripción: Autentica un token de acceso.

Parámetros:

token: Token de acceso.

Excepciones: TokenExpiredException

Respuestas:

Retorna un objeto ResponseAuthToken con los detalles de autenticación.

200 OK: Autenticación exitosa.

404 NOT FOUND

Método: changePass

Endpoint: POST /tfg/change-password

Descripción: Permite a un usuario cambiar su contraseña.

Parámetros:

request: Objeto RequestChangePassword que contiene la información necesaria para el cambio de contraseña.

Excepciones: UserNotFoundException

Respuestas:

200 OK: Cambio de contraseña exitoso.

404 NOT FOUND

Método: createPost

Endpoint: POST /tfg/create-post

Descripción: Crea una nueva publicación en la red social.

Parámetros:

image: Archivo de imagen de la publicación.

header: Encabezado de la publicación.

Excepciones: ImagePostException, UserNotFoundException, EmptyFileException

Respuestas:

Retorna un objeto ResponseCreatePostVO con los detalles de la publicación creada.

200 OK: Creación de publicación exitosa.

204 NO CONTENT

404 NOT FOUND

500 INTERNAL SERVER ERROR

Método: getFollowingPosts

Endpoint: GET /tfg/feed

Descripción: Obtiene las publicaciones de los usuarios seguidos por el usuario actual.

Excepciones: UserNotFoundException, PostNotFoundException

Respuestas:

Retorna una lista de objetos PostDTO con los detalles de las publicaciones.

200 OK: Obtención de publicaciones exitosa.

404 NOT FOUND

Método: searchUser

Endpoint: GET /tfg/search-user

Descripción: Busca usuarios en la red social por nombre de usuario.

Parámetros:

username: Nombre de usuario a buscar.

Excepciones: UserNotFoundException

Respuestas:

Retorna un objeto ResponseSearchUserVO con los detalles del usuario encontrado

200 OK: Búsqueda de usuario exitosa.

404 NOT FOUND

Método: followRequest

Endpoint: GET /tfg/ follow-request

Descripción: Sigue usuarios en la red social por nombre de usuario.

Parámetros:

request: Nombre de usuario a buscar.

Excepciones: UserNotFoundException

Respuestas:

Retorna un objeto ResponseSearchUserVO con los detalles del usuario encontrado

200 OK: Búsqueda de usuario exitosa.

404 NOT FOUND

Método: unfollow

Endpoint: GET /tfg/unfollow

Descripción: Deja de seguir usuarios en la red social por nombre de usuario.

Parámetros:

request: Nombre de usuario a buscar.

Excepciones: UserNotFoundException

Respuestas:

Retorna un objeto ResponseSearchUserVO con los detalles del usuario encontrado

200 OK: Búsqueda de usuario exitosa.

404 NOT FOUND

Método: showProfile

Endpoint: GET /tfg/show-profile

Descripción: Obtiene los detalles del perfil del usuario actual.

Excepciones: UserNotFoundException

Respuestas:

Retorna un objeto ResponseShowProfileVO con los detalles del perfil.

200 OK: Obtención de perfil exitosa.

404 NOT FOUND

Método: editUser

Endpoint: POST /tfg/edit-user

Descripción: Permite editar la información de un usuario.

Parámetros:

request: Objeto RequestEditUserVO que contiene la nueva información del usuario.

Excepciones: UserNotFoundException, UserUsedException

Respuestas:

200 OK: Edición de usuario exitosa.

404 NOT FOUND

409 CONFLICT

Método: like

Endpoint: POST /tfg/like

Descripción: Permite que un usuario le dé "Me gusta" a una publicación.

Parámetros:

request: Objeto PostDTO que contiene los detalles de la publicación a la que se le dará "Me gusta".

Excepciones: UserNotFoundException, PostNotFoundException

Respuestas:

200 OK: Acción de "Me gusta" realizada exitosamente.

404 NOT FOUND

Método: getLiked

Endpoint: GET /tfg/getLiked

Descripción: Obtiene las publicaciones a las que un usuario ha dado "Me gusta".

Excepciones: UserNotFoundException, PostNotFoundException

Respuestas:

Retorna una lista de objetos PostDTO con los detalles de las publicaciones.

200 OK: Obtención de publicaciones exitosa.

404 NOT FOUND

4.3.2 Seguridad

La API cuenta con un sólido sistema de seguridad implementado mediante Spring Security, el cual se encarga de verificar la autenticación y autorización de los usuarios que acceden a ella. Para lograr esto, se utiliza el mecanismo de JSON Web Tokens (JWT).

El uso de JSON Web Tokens brinda un nivel adicional de seguridad al proceso de autenticación de la API. Mediante este enfoque, se generan tokens criptográficamente seguros que contienen información sobre la identidad del usuario y los permisos asignados. Estos tokens son firmados digitalmente y se envían al cliente, quien los incluye en las solicitudes posteriores.

Al recibir una solicitud, la API valida y verifica la autenticidad de cada token JWT enviado por el cliente. Esto permite garantizar que el usuario esté autenticado correctamente y tenga los permisos necesarios para acceder a los recursos solicitados. En caso de que el token no sea válido o haya expirado, se denegará el acceso y se devolverá una respuesta correspondiente.

Gracias a estas medidas de seguridad, los usuarios pueden tener la confianza de que su información personal y las interacciones en la red social están protegidas de accesos no autorizados.

4.3.3 Manejo de errores y excepciones

En cuanto al manejo de errores y excepciones, se ha diseñado un controlador de excepciones personalizado para abordar y gestionar los errores que puedan surgir durante el funcionamiento de la API. Este controlador se encarga de capturar y manejar las excepciones, devolviendo respuestas HTTP adecuadas y mensajes descriptivos de error.

Al utilizar este *exceptionHandler* personalizado, se logra un tratamiento más controlado y consistente de las excepciones en la API. En caso de que ocurra alguna excepción, se captura de manera centralizada y se procesa para generar una respuesta adecuada, evitando así que se produzcan errores no controlados o respuestas inesperadas.

Además, se han creado excepciones personalizadas para manejar los errores específicos que pueden ocurrir en la API. Estas excepciones están diseñadas para representar situaciones particulares, como errores de usuario no encontrado, contraseñas incorrectas, solicitudes inválidas, entre otros. Al utilizar excepciones personalizadas, se mejora la claridad y la granularidad en la identificación y el manejo de los errores, lo que facilita la depuración y el mantenimiento del sistema.

En resumen, mediante el uso de un *exceptionHandler* personalizado y excepciones propias, se garantiza un manejo eficiente de los errores y excepciones en la API.

4.4 Diseño del Frontend

En este apartado, nos adentramos en el diseño del Frontend de la aplicación, que es la interfaz de usuario con la que los usuarios interactuarán directamente. El diseño del Frontend es fundamental para ofrecer una experiencia atractiva, intuitiva y funcional a los usuarios.

4.4.1 Diseño pantallas

En el proceso de diseño de las pantallas, se ha seguido una metodología que incluye la creación de prototipos de baja fidelidad. Estos prototipos permiten obtener una visión general y preliminar de las interfaces antes de proceder al desarrollo completo de las pantallas.

Los prototipos de baja fidelidad son representaciones esquemáticas y simplificadas de las pantallas, donde se prioriza la estructura y disposición de los elementos visuales sobre los detalles visuales y de diseño. Estos prototipos suelen realizarse utilizando herramientas sencillas, como lápiz y papel o software especializado en prototipado.

Los prototipos principales serían los siguientes:

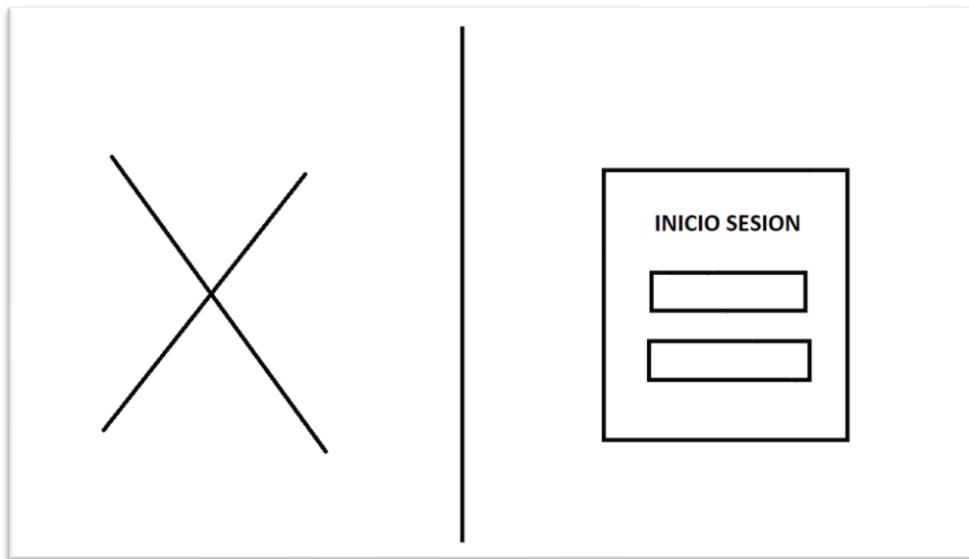


Ilustración 15. Prototipo Baja Fidelidad 1

Prototipo de la pantalla de inicio de sesión de iniciación a la aplicación. La X representa una imagen o un diseño para aportar un apoyo visual atractivo a la página de inicio de sesión principal.

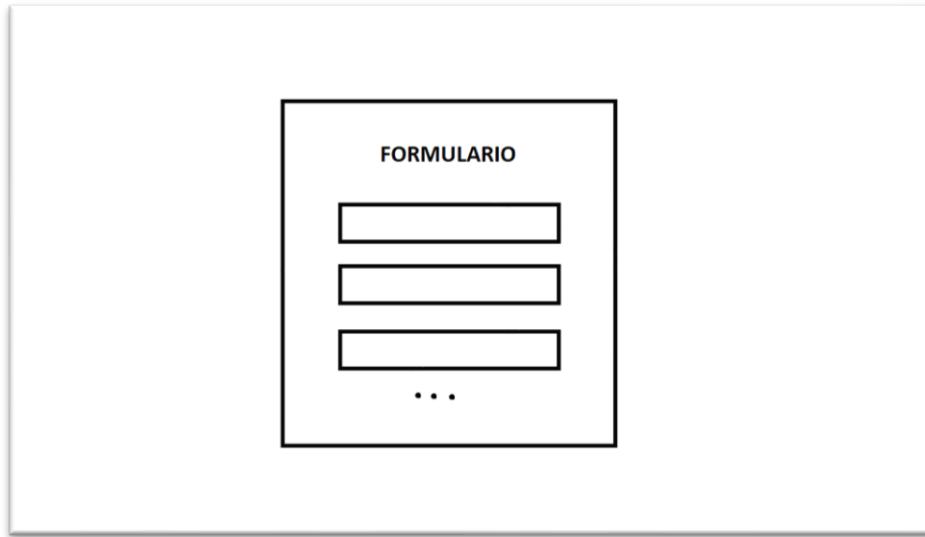


Ilustración 16. Prototipo Baja Fidelidad 2

Prototipo diseñado para cualquier formulario, ya sea de inicio de sesión, de registro o de cambio de contraseña.



Ilustración 17. Prototipo Baja Fidelidad 3

Prototipo de la barra de navegación lateral con el nombre de la aplicación en la parte central superior.

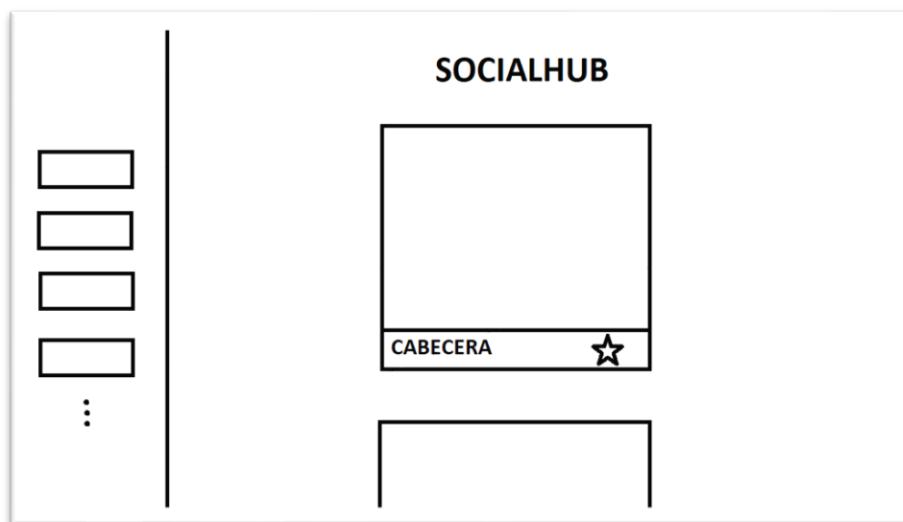


Ilustración 18. Prototipo Baja Fidelidad 4

Prototipo la pantalla del “feed” de publicaciones.

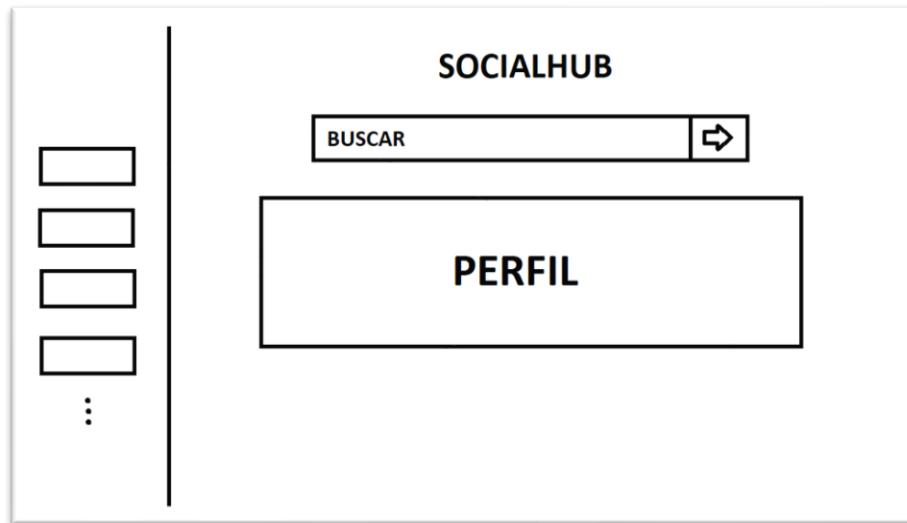


Ilustración 19. Prototipo Baja Fidelidad 5

Prototipo de la pantalla de “buscar”, donde se muestra la barra de búsqueda y el perfil del usuario en caso de haber sido encontrado.

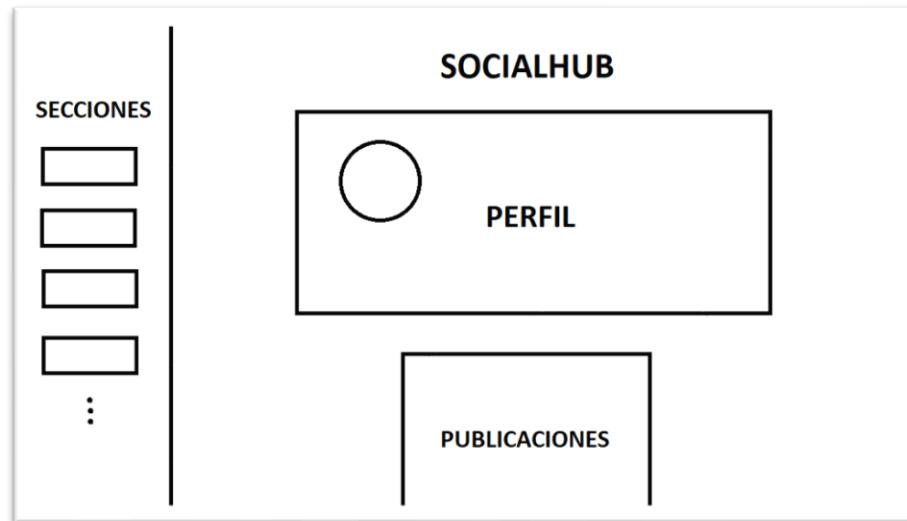


Ilustración 20. Prototipo Baja Fidelidad 6

Prototipo de cualquier pantalla que muestre un perfil.

4.4.2 Diseño y recursos gráficos utilizados

Se ha diseñado un logo para el trabajo, brindándole a este un toque más profesional.

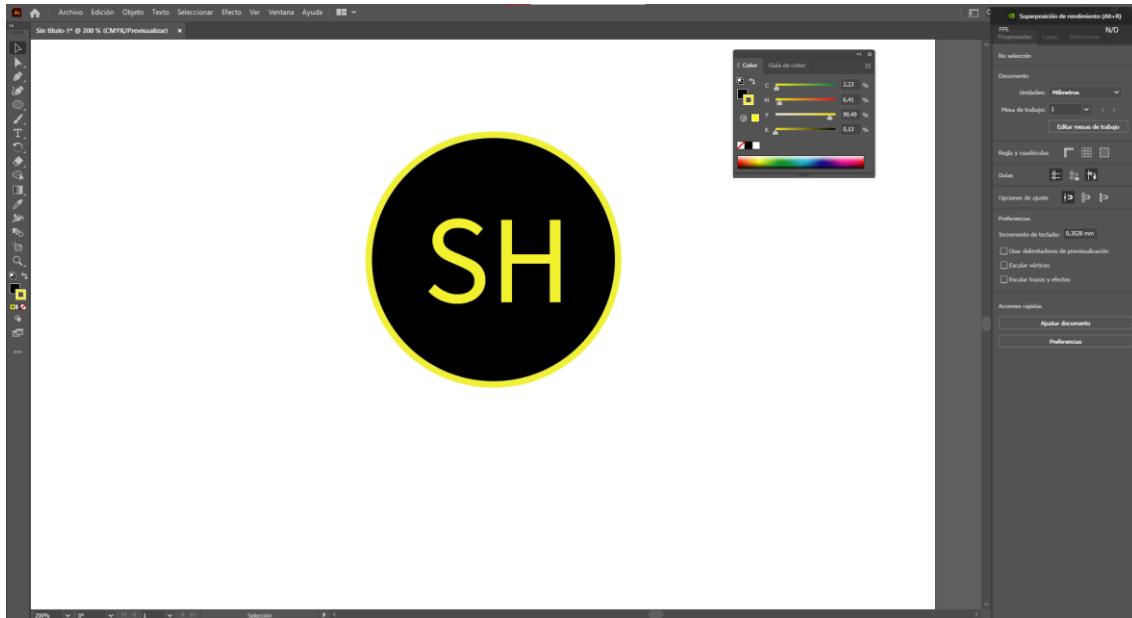


Ilustración 21. Proceso de diseño del logo



Ilustración 22. SocialHub logo

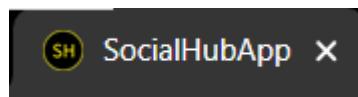


Ilustración 23. SocialHub logo en ventana

Hablando de los recursos gráficos, se han utilizado varios emoticonos [19] en la aplicación para ciertas secciones. Además, se ha utilizado una imagen de acceso abierto y libre [20] para la página de inicio de sesión principal.

4.4.3 Estructura del proyecto Angular

En el apartado de la estructura del proyecto Angular, se aborda la organización y estructuración de la aplicación utilizando el *framework* Angular. El objetivo principal es establecer una estructura clara y escalable

que facilite el desarrollo, mantenimiento y reutilización de los componentes y funcionalidades.

Módulo Principal (app.module):

El módulo principal es el punto de entrada de la aplicación Angular. Aquí se definen las dependencias y configuraciones generales del proyecto.

Módulo de Autenticación y Autorización (auth.module):

Este módulo se encarga de gestionar la autenticación y autorización de los usuarios en la aplicación. Está compuesto por diferentes componentes que abordan distintos aspectos del proceso de autenticación, como el registro de usuarios, el inicio de sesión (que incluye tanto el formulario básico como el formulario principal), la verificación del email y la recuperación de contraseña (con su respectivo subcomponente para restablecer la contraseña).

Módulo de Contenido de la Aplicación (home.module):

Este módulo se centra en el contenido principal de la aplicación una vez que el usuario ha iniciado sesión. Contiene componentes relacionados con el feed de publicaciones, la visualización de publicaciones filtradas por hobbies (planificado para una futura fase), la lista de publicaciones que el usuario ha marcado como Me gusta, el perfil personal y la búsqueda de usuarios.

Módulo de Componentes Compartidos (shared.module):

En este módulo se encuentran componentes y métodos que son compartidos por varios componentes de la aplicación. Un ejemplo de esto es el componente de la barra de navegación lateral, que es utilizado por todos los componentes del módulo home.

Carpeta de Servicios:

Los Servicios desempeñan un papel fundamental en la comunicación con la API REST. En esta carpeta se encuentran los servicios correspondientes al módulo de autenticación, así como el servicio utilizado en el módulo de home.

4.4.4 Documentación de las rutas de la aplicación

El enrutamiento de la aplicación se encarga de definir las rutas y redirecciónamientos dentro de la misma. A continuación, se muestra la documentación del enrutamiento basado en el código proporcionado:

Rutas del módulo Home:

- Ruta principal: **/home**
 - Guardia de activación: **guardsGuard**
 - Carga perezosa (lazy loading) del módulo: **HomeModule**

Dentro del módulo Home, se definen las siguientes rutas hijas:

- Ruta: **/feed**
 - Componente asociado: **FeedComponent**
- Ruta: **/search**
 - Componente asociado: **SearchComponent**
- Ruta: **/hobbies**
 - Componente asociado: **HobbiesComponent**
- Ruta: **/liked**
 - Componente asociado: **LikedComponent**
- Ruta: **/profile**
 - Componente asociado: **ProfileComponent**
- Ruta: **/profile/{usuario}**
 - Componente asociado: **UserprofileComponent**
- Ruta por defecto: ****** (cualquier otra ruta no especificada)
 - Redirecciónamiento a: **/feed**

Rutas del módulo de autenticación (Auth):

- Ruta principal: "" (vacío)
- Carga perezosa (lazy loading) del módulo: **AuthModule**

Dentro del módulo de autenticación, se definen las siguientes rutas hijas:

- Ruta: **/loginhome**
 - Componente asociado: **LoginhomeComponent**
- Ruta: **/register**
 - Componente asociado: **RegisterComponent**
- Ruta: **/login**
 - Componente asociado: **LoginComponent**
- Ruta: **/verify-email**
 - Componente asociado: **EmailverificationComponent**
- Ruta: **/forgot-pwd**
 - Componente asociado: **ForgotPwdComponent**
- Ruta: **/reset-pwd**
 - Componente asociado: **ResetpasswordComponent**
- Ruta por defecto: ** (cualquier otra ruta no especificada)
 - Redireccionamiento a: **/loginhome**

Estas configuraciones de enrutamiento permiten que las diferentes rutas definidas en los módulos de Home y Auth se carguen de manera dinámica y se muestren correctamente en la interfaz de usuario. La utilización de guardias de activación, como **guardsGuard**, puede ayudar a controlar el acceso a ciertas rutas según la lógica de autenticación y autorización definida en la aplicación.

5 Implementación

En este apartado, se aborda la implementación del sistema, centrándose en el desarrollo de la API REST utilizando Java con Spring Boot y la implementación del Frontend con Angular.

5.1 Desarrollo de la API REST con Java y Spring Boot

5.1.1 Configuración del proyecto

La configuración del proyecto es un paso fundamental en la implementación de una API REST utilizando Java y Spring.

En la configuración inicial del proyecto, se ha optado por utilizar Maven como herramienta para gestionar las dependencias del proyecto. Maven simplifica el proceso de importar bibliotecas y asegura que todas las dependencias necesarias se encuentren correctamente configuradas.



```
1 <dependency>
2     <groupId>org.springframework.boot</groupId>
3     <artifactId>spring-boot-starter-data-jpa</artifactId>
4 </dependency>
5 <dependency>
6     <groupId>org.springframework.boot</groupId>
7     <artifactId>spring-boot-starter-mail</artifactId>
8 </dependency>
9 <dependency>
10    <groupId>org.springframework.boot</groupId>
11    <artifactId>spring-boot-starter-security</artifactId>
12 </dependency>
13 <dependency>
14    <groupId>org.springframework.boot</groupId>
15    <artifactId>spring-boot-starter-web</artifactId>
16 </dependency>
17 <dependency>
18    <groupId>com.mysql</groupId>
19    <artifactId>mysql-connector-j</artifactId>
20    <scope>runtime</scope>
21 </dependency>
22 <dependency>
23    <groupId>org.projectlombok</groupId>
24    <artifactId>lombok</artifactId>
25    <optional>true</optional>
26 </dependency>
27 <dependency>
28    <groupId>io.jsonwebtoken</groupId>
29    <artifactId>jjwt-api</artifactId>
30    <version>0.11.5</version>
31 </dependency>
```

Ilustración 24. Código pom.xml

Estas son las dependencias principales del pom.xml que han sido utilizadas.

spring-boot-starter-data-jpa: Esta dependencia proporciona las funcionalidades de Spring Data JPA, que simplifica el acceso y la manipulación de la base de datos mediante el uso de la API de persistencia de Java (JPA).

spring-boot-starter-mail: Esta dependencia agrega las capacidades de envío de correos electrónicos a la aplicación utilizando las funcionalidades proporcionadas por Spring Boot.

spring-boot-starter-security: Esta dependencia integra Spring Security en la aplicación, permitiendo la implementación de la autenticación y la autorización para proteger los recursos y gestionar el acceso de los usuarios.

spring-boot-starter-web: Esta dependencia proporciona un conjunto de bibliotecas para construir aplicaciones web con Spring MVC (Modelo-Vista-Controlador), permitiendo la creación de controladores, gestión de solicitudes y respuestas HTTP, entre otras funcionalidades web.

mysql-connector-j: Esta dependencia es el controlador JDBC para MySQL, que permite la conexión y la comunicación con una base de datos MySQL.

lombok: Esta dependencia de desarrollo opcional agrega características de generación automática de código a través de anotaciones, como la generación de getters y setters, constructores, métodos equals y hashCode, entre otros, para reducir la cantidad de código repetitivo y mejorar la legibilidad.

jjwt-api: Esta dependencia proporciona las API para trabajar con JSON Web Tokens (JWT), un estándar para la creación de tokens de acceso seguros y autenticados. Permite la generación, verificación y manipulación de JWT en la aplicación.

Además, se han realizado ajustes en el archivo "*application.properties*" para establecer la conexión adecuada con la base de datos. Esta configuración incluye la especificación del controlador de la base de datos, la URL de conexión, el nombre de usuario y la contraseña necesarios para acceder a la base de datos. Estos cambios aseguran que la API REST pueda interactuar de manera correcta y segura con la base de datos subyacente.

```
1 # Base de Datos
2 spring.datasource.url=jdbc:mysql://localhost:3306/socialhub_db
3 spring.datasource.username=root
4 spring.datasource.password=TrabajoFinGradoSantiago2023_
5 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

Ilustración 25. *application.properties*

Estos son los ajustes necesarios para la correcta conexión con la base de datos.

La elección de Maven como herramienta de gestión de dependencias y la configuración adecuada del archivo "*application.properties*" son pasos

esenciales en la configuración inicial del proyecto. Estas decisiones permiten establecer las bases sólidas para el correcto funcionamiento y la conectividad exitosa de la API REST con la base de datos.

Además, la API REST cuenta con un paquete de configuración que incluye tres archivos de configuración: *CorsConfig*, *GeneralConfig* y *SecurityConfig*.

El archivo **CorsConfig** se encarga de manejar el acceso a la API desde diferentes orígenes, permitiendo todas las solicitudes desde el origen del Frontend.



```
● ○ ●
1  @Configuration
2  public class CorsConfig {
3
4      @Bean
5      public WebMvcConfigurer corsConfigurer(){
6          return new WebMvcConfigurer() {
7              @Override
8              public void addCorsMappings(CorsRegistry registry) {
9
10                  registry.addMapping("/tfg/**")
11                      .allowedOrigins("http://localhost:4200")
12                      .allowedMethods("*")
13                      .exposedHeaders("*");
14              }
15          };
16      }
17 }
```

Ilustración 26. CorsConfig

El archivo **GeneralConfig** se utiliza para agregar y configurar el *Model Mapper*, una herramienta que facilita el mapeo de objetos entre diferentes modelos de datos.

Por otro lado, el archivo **SecurityConfig**, junto con el paquete de seguridad del proyecto, se encarga de implementar la seguridad, autenticación y autorización de los usuarios. Estos archivos son fundamentales para garantizar la protección de los recursos y la gestión de los roles y permisos de los usuarios.

```

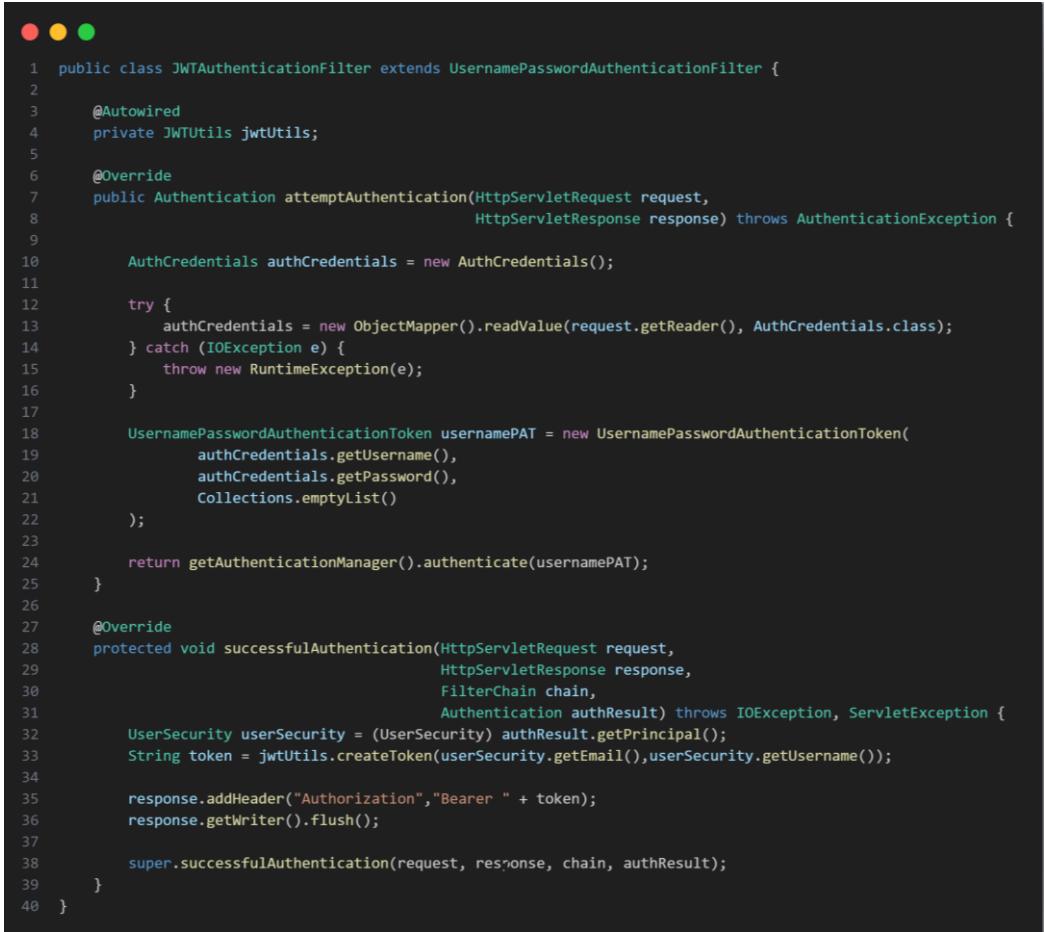
1  @Configuration
2  @AllArgsConstructor
3  public class SecurityConfig {
4
5      @Autowired
6      private UserDetailsService userDetailsService;
7      @Autowired
8      private JWTAuthorizationFilter jwtAuthorizationFilter;
9
10     @Bean
11     public SecurityFilterChain filterChain(HttpSecurity httpSecurity, AuthenticationManager authManager) throws Exception{
12         JWTAuthenticationFilter jwtAuthenticationFilter = new JWTAuthenticationFilter();
13         jwtAuthenticationFilter.setAuthenticationManager(authManager);
14         jwtAuthenticationFilter.setFilterProcessesUrl("/login");
15         return httpSecurity
16             .cors()
17             .and()
18             .csrf().disable()
19             .authorizeHttpRequests()
20             .requestMatchers("/tfg/register", "/tfg/verify-email", "/tfg/login", "/tfg/forgot-password",
21                             "/tfg/authenticate-token", "/tfg/change-password").permitAll()
22             .anyRequest()
23             .authenticated()
24             .and()
25             .sessionManagement()
26             .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
27             .and()
28             .addFilter(jwtAuthenticationFilter)
29             .addFilterBefore(jwtAuthorizationFilter, UsernamePasswordAuthenticationFilter.class)
30             .build();
31     }
32
33     @Bean
34     public AuthenticationManager authManager(HttpSecurity httpSecurity) throws Exception{
35         return httpSecurity.getSharedObject(AuthenticationManagerBuilder.class)
36             .userDetailsService(userDetailsService)
37             .passwordEncoder(bCryptPasswordEncoder())
38             .and()
39             .build();
40     }
41
42     @Bean
43     public BCryptPasswordEncoder bCryptPasswordEncoder(){
44         return new BCryptPasswordEncoder();
45     }
46 }
47 }
```

Ilustración 27. *Security Config*

El método *filterChain* configura las reglas de seguridad. Se deshabilita la protección CSRF y se configuran las rutas que se permiten sin autenticación. Todas las demás rutas requieren autenticación. Se establece una política de gestión de sesiones sin estado (STATELESS). Además, se agregan filtros de autenticación y autorización.

El método *authManager* configura el *AuthenticationManager*, que se encarga de autenticar a los usuarios utilizando el *UserDetailsService* y el *BCryptPasswordEncoder* para la codificación de contraseñas.

Por último, el método *bCryptPasswordEncoder* crea y devuelve un *BCryptPasswordEncoder*, que se utiliza para codificar las contraseñas de los usuarios.



```

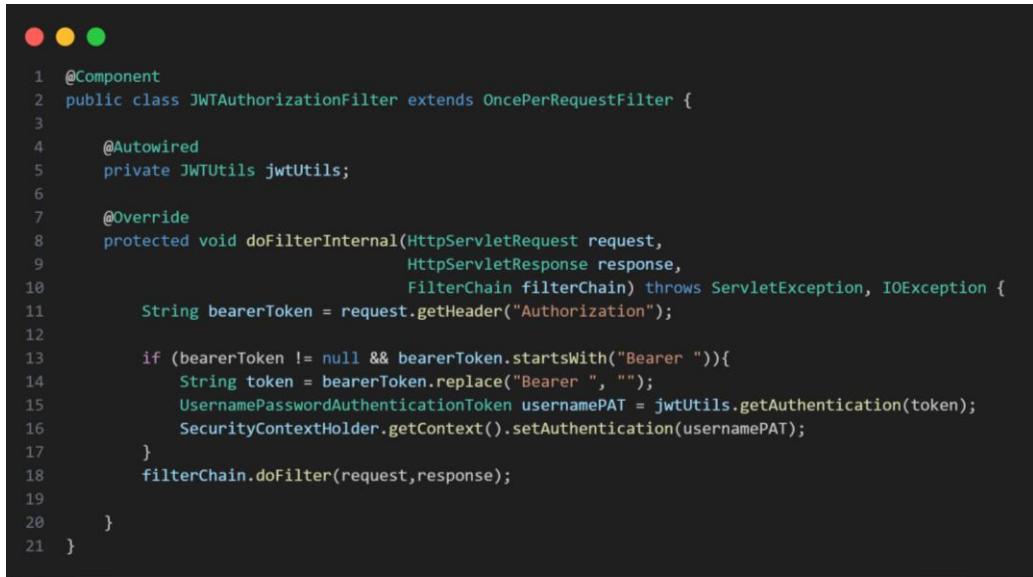
1 public class JWTAuthenticationFilter extends UsernamePasswordAuthenticationFilter {
2
3     @Autowired
4     private JWTUtils jwtUtils;
5
6     @Override
7     public Authentication attemptAuthentication(HttpServletRequest request,
8                                                 HttpServletResponse response) throws AuthenticationException {
9
10    AuthCredentials authCredentials = new AuthCredentials();
11
12    try {
13        authCredentials = new ObjectMapper().readValue(request.getReader(), AuthCredentials.class);
14    } catch (IOException e) {
15        throw new RuntimeException(e);
16    }
17
18    UsernamePasswordAuthenticationToken usernamePAT = new UsernamePasswordAuthenticationToken(
19        authCredentials.getUsername(),
20        authCredentials.getPassword(),
21        Collections.emptyList()
22    );
23
24    return getAuthenticationManager().authenticate(usernamePAT);
25}
26
27     @Override
28     protected void successfulAuthentication(HttpServletRequest request,
29                                         HttpServletResponse response,
30                                         FilterChain chain,
31                                         Authentication authResult) throws IOException, ServletException {
32     UserSecurity userSecurity = (UserSecurity) authResult.getPrincipal();
33     String token = jwtUtils.createToken(userSecurity.getEmail(), userSecurity.getUsername());
34
35     response.addHeader("Authorization", "Bearer " + token);
36     response.getWriter().flush();
37
38     super.successfulAuthentication(request, response, chain, authResult);
39   }
40 }

```

Ilustración 28. JWTAuthenticationFilter

En el método *attemptAuthentication*, se intenta autenticar al usuario a partir de las credenciales proporcionadas en la solicitud. Se obtienen las credenciales del cuerpo de la solicitud y se crea un *UsernamePasswordAuthenticationToken* con el nombre de usuario y la contraseña. Luego, se devuelve el resultado de la autenticación utilizando el *AuthenticationManager* de Spring Security.

En el método *successfulAuthentication*, se ejecuta después de que la autenticación haya sido exitosa. Se obtiene el objeto *UserSecurity* que representa al usuario autenticado y se genera un token JWT utilizando un *JWTUtils*. A continuación, se agrega el token al encabezado de la respuesta y se envía al cliente. Además, se llama al método *super.successfulAuthentication()* para continuar con la cadena de filtros de autenticación.



```

1  @Component
2  public class JWTAuthorizationFilter extends OncePerRequestFilter {
3
4      @Autowired
5      private JWTUtils jwtUtils;
6
7      @Override
8      protected void doFilterInternal(HttpServletRequest request,
9                                      HttpServletResponse response,
10                                     FilterChain filterChain) throws ServletException, IOException {
11          String bearerToken = request.getHeader("Authorization");
12
13          if (bearerToken != null && bearerToken.startsWith("Bearer ")){
14              String token = bearerToken.replace("Bearer ", "");
15              UsernamePasswordAuthenticationToken usernamePAT = jwtUtils.getAuthentication(token);
16              SecurityContextHolder.getContext().setAuthentication(usernamePAT);
17          }
18          filterChain.doFilter(request,response);
19      }
20  }
21 }

```

Ilustración 29. *JWTAuthorizationFilter*

En el método *doFilterInternal*, se verifica si la solicitud contiene un encabezado "Authorization" que comienza con "Bearer". Si se cumple esta condición, se extrae el token JWT de la solicitud y se utiliza el *JWTUtils* para obtener un *UsernamePasswordAuthenticationToken* que representa la autenticación del usuario asociada con ese token. A continuación, se establece la autenticación en el contexto de seguridad utilizando *SecurityContextHolder*.

Finalmente, se llama al método *doFilter* de la cadena de filtros para permitir que la solicitud continúe su procesamiento normal.

Además, existen 3 objetos, los cuales contienen datos como el nombre de usuario y la contraseña, e implementan otros interfaces auxiliares de autenticación. Los archivos son los siguientes: *AuthCredentials*, *UserSecurity* y *UserSecurityServiceImpl*.



Ilustración 30. Paquete Seguridad API

5.1.2 Desarrollo del Controlador

En la etapa de desarrollo del controlador de la API REST, se implementan las clases encargadas de manejar las solicitudes HTTP y realizar las operaciones necesarias en la base de datos. Para ello, se utilizan anotaciones

proporcionadas por Spring Boot, como `@RestController` y `@RequestMapping`, que permiten definir los *endpoints* y las acciones asociadas a cada uno.

Además, se hace uso de un *Logger* para poder mostrar logs de la traza de ejecución, con el objetivo de clarificar a qué métodos accede una solicitud y poder agilizar la resolución de errores.

La clase "UserController" es el controlador REST que maneja las solicitudes de la aplicación. Está mapeada a la ruta base "/tfg" en el servidor con `@RequestMapping`.



```
● ● ●
1 @RestController
2 @RequestMapping("tfg")
3 public class UserController extends ExceptionHandlerSH {
```

Ilustración 31. Anotaciones Controlles Class

Como ejemplo, se mostrará el método de “*register*” y una explicación general que engloba al resto de métodos del Controlador, los cuales siguen una estructura similar:



```
● ● ●
1 @PostMapping("/register")
2     public ResponseEntity<ResponseRegisterVO> register(@Valid @RequestBody RequestRegisterVO request)
3         throws EmailUsedException, UserUsedException, IOException {
4     log.info("Start register method -> Parameters [{}]",request);
5     ResponseRegisterVO response = userService.register(request);
6     log.info("End register method -> Result [{}]",response);
7     return new ResponseEntity<>(response,HttpStatus.CREATED);
8 }
```

Ilustración 32. Método register API

El método está anotado con `@PostMapping` y se mapea a la ruta "/ruta-metodo", lo que significa que se invocará cuando se realice una solicitud HTTP POST a esa ruta específica.

El método toma como parámetro el objeto RequestVO, que se espera que se proporcione en el cuerpo de la solicitud. La anotación `@RequestBody` indica a Spring que los datos de la solicitud deben mapearse al objeto RequestVO. Antes de que se ejecute el método, se aplica la anotación `@Valid` al objeto RequestVO, lo que indica que se deben aplicar validaciones de datos.

Dentro del método, se realiza un registro del evento utilizando `log.info()`. Esto ayuda a rastrear y depurar la ejecución del código.

A continuación, se llama al servicio correspondiente para así poder realizar allí la lógica del método correspondiente.

El resultado del registro se guarda en un objeto ResponseVO, que contiene los detalles del registro.

Finalmente, se crea una respuesta ResponseEntity que contiene el objeto response y se devuelve con un código de estado HTTP 201 (CREATED) para indicar que el registro se ha completado con éxito.

El resto de los métodos implementados del controlador se mostrarán abajo. Tienen la misma estructura que el recién explicado, por lo que el comportamiento es análogo.

```
● ● ●
1 @RestController
2 @RequestMapping("tfg")
3 public class UserController extends ExceptionHandlerSH {
4
5     private static final Logger log = LoggerFactory.getLogger(UserController.class);
6     @Autowired
7     private UserService userService;
8
9     @PostMapping("/register")
10    public ResponseEntity<ResponseRegisterVO> register(@Valid @RequestBody RequestRegisterVO request)
11        throws EmailUsedException, UserUsedException, IOException {
12        log.info("Start register method -> Parameters [{}]",request);
13        ResponseRegisterVO response = userService.register(request);
14        log.info("End register method -> Result [{}]",response);
15        return new ResponseEntity<>(response,HttpStatus.CREATED);
16    }
17
18    @GetMapping("/verify-email")
19    public ResponseEntity<ResponseVerifyEmail> verifyEmail(@RequestParam("token") String token)
20        throws UserNotFoundException, UserAlreadyEnabledException, TokenExpiredException {
21        log.info("Start verifyEmail method");
22        ResponseVerifyEmail response = userService.verifyEmail(token);
23        log.info("End verifyEmail method");
24        return new ResponseEntity<>(response,HttpStatus.OK);
25    }
26
27    @PostMapping("/login")
28    public ResponseEntity<?> login(@Valid @RequestBody RequestLoginVO request)
29        throws UserNotFoundException, IncorrectPwdException {
30        log.info("Start login method -> Parameters [{}]",request);
31        ResponseLoginVO response = userService.login(request);
32        log.info("End login method -> Response [{}], response.getHeader()");
33        return new ResponseEntity<>(response.getId_user(),response.getHeader(),HttpStatus.OK);
34    }
35
36    @PostMapping("/forgot-password")
37    public ResponseEntity<?> forgotPass(@Valid @RequestBody String emailRequest)
38        throws UserNotFoundException{
39        log.info("Start forgotPass method -> Parameters [{}],emailRequest");
40        userService.forgotPass(emailRequest);
41        log.info("End forgotPass method");
42        return new ResponseEntity<>(HttpStatus.OK);
43    }
44
45    @GetMapping("/authenticate-token")
46    public ResponseEntity<ResponseAuthToken> authToken(@RequestParam("token") String token)
47        throws TokenExpiredException{
48        log.info("Start authToken method -> Parameters [{}],token");
49        ResponseAuthToken response = userService.authToken(token);
50        log.info("End authToken method");
51        return new ResponseEntity<>(response,HttpStatus.OK);
52    }
```

Ilustración 33. UserController 1

```

1  @PostMapping("/change-password")
2  public ResponseEntity<?> changePass(@Valid @RequestBody RequestChangePassword request)
3      throws UserNotFoundException{
4      log.info("Start changePass method -> Parameters [{}]",request);
5      userService.changePass(request);
6      log.info("End changePass method");
7      return new ResponseEntity<>(HttpStatus.OK);
8  }
9
10 @PostMapping("/create-post")
11 public ResponseEntity<ResponseCreatePostVO> createPost(@RequestPart("image") MultipartFile image,
12                                         @RequestPart("header") String header)
13     throws ImagePostException, UserNotFoundException, EmptyFileException {
14     RequestCreatePostVO request = new RequestCreatePostVO();
15     request.setHeader(header);
16     request.setPostContent(image);
17     log.info("Start createPost method -> Parameters [{}]",request);
18     ResponseCreatePostVO response = userService.createPost(getLogedUser(),request);
19     log.info("End createPost method");
20     return new ResponseEntity<>(response,HttpStatus.OK);
21 }
22
23 @GetMapping("/feed")
24 public ResponseEntity<List<PostDTO>> getFollowingPosts()
25     throws UserNotFoundException, PostNotFoundException {
26     log.info("Start getHomePosts method");
27     List<PostDTO> response = userService.getFollowingPosts(getLogedUser());
28     log.info("End getHomePosts method");
29     return new ResponseEntity<>(response,HttpStatus.OK);
30 }
31
32 @GetMapping("/search-user")
33 public ResponseEntity<ResponseSearchUserVO> searchUser(@RequestParam String username)
34     throws UserNotFoundException {
35     log.info("Start searchUser method");
36     ResponseSearchUserVO response = userService.searchUser(getLogedUser(),username);
37     log.info("End searchUser method");
38     return new ResponseEntity<>(response,HttpStatus.OK);
39 }
40
41 @PostMapping("/follow-request")
42 public ResponseEntity<String> followRequest (@Valid @RequestBody String request) throws UserNotFoundException {
43     log.info("Start FollowRequest method");
44     String response = userService.followRequest(getLogedUser(),request);
45     log.info("End followRequest method");
46     return new ResponseEntity<>(response,HttpStatus.OK);
47 }
48
49 @PostMapping("/unfollow")
50 public ResponseEntity<String> unfollow (@Valid @RequestBody String request) throws UserNotFoundException {
51     log.info("Start unfollow method");
52     String response = userService.unfollow(getLogedUser(),request);
53     log.info("End unfollow method");
54     return new ResponseEntity<>(response,HttpStatus.OK);
55 }

```

Ilustración 34. UserController 2

```
1  @GetMapping("/show-profile")
2  public ResponseEntity<ResponseShowProfileVO> showProfile(@RequestParam String username)
3      throws UserNotFoundException, PostNotFoundException {
4      log.info("Start showProfile method -> Parameters [{}]",username);
5      String usernameRequest = username;
6      if(usernameRequest.equals("")){
7          usernameRequest = getLoggedUser();
8      }
9      ResponseShowProfileVO response = userService.showProfile(usernameRequest);
10     log.info("End showProfile method");
11     return new ResponseEntity<>(response,HttpStatus.OK);
12 }
13
14 @PostMapping("/edit-user")
15 public ResponseEntity<> editUser(@Valid @RequestBody RequestEditUserVO request)
16     throws UserNotFoundException, UserUsedException {
17     log.info("Start editUser method -> Parameters [{}]",request);
18     userService.editUser(getLoggedUser(),request);
19     log.info("End editUser method");
20     return new ResponseEntity<>(HttpStatus.OK);
21 }
22
23 @PostMapping("/like")
24 public ResponseEntity<> like(@Valid @RequestBody PostDTO request)
25     throws UserNotFoundException, PostNotFoundException {
26     log.info("Start like method -> Parameters [{}]",request);
27     userService.like(getLoggedUser(),request);
28     log.info("End like method");
29     return new ResponseEntity<>(HttpStatus.OK);
30 }
31
32 @GetMapping("/getLiked")
33 public ResponseEntity<List<PostDTO>> getLiked() throws UserNotFoundException, PostNotFoundException {
34     log.info("Start getLiked method");
35     List<PostDTO> likedPosts = userService.getLiked(getLoggedUser());
36     log.info("End getLiked method");
37     return new ResponseEntity<>(likedPosts,HttpStatus.OK);
38 }
39
40 private String getLoggedUser(){
41     return SecurityContextHolder.getContext().getAuthentication().getPrincipal().toString();
42 }
43
44 }
```

Ilustración 35. UserController 3

5.1.3 Desarrollo de los Servicios

En la sección de Desarrollo de los servicios se aborda la implementación de la lógica de negocio de la aplicación en forma de servicios. Estos servicios se encargan de procesar las solicitudes recibidas desde los controladores, realizar operaciones en la base de datos y proporcionar las respuestas correspondientes.

Los servicios del API REST se encuentran definidos en una interfaz llamada `{Nombre}Service`, donde se especifican los métodos que deben ser implementados. La lógica de estos servicios se encuentra en las clases `{Nombre}ServiceImpl`, las cuales implementan los métodos definidos en la interfaz.

A continuación, se detallan los servicios disponibles en el API REST:

5.1.3.1 UserService

El servicio *UserService* se encarga de procesar las solicitudes relacionadas con los usuarios, las publicaciones y las interacciones entre ambos. Este servicio implementa los siguientes métodos:

Método register



```
1  @Override
2  public ResponseRegisterVO register(RequestRegisterVO request)
3      throws EmailUsedException, UserUsedException, IOException {
4      log.info("Start register method");
5      // Validaciones de email y usuario
6      if(userRepository.findByUsername(request.getUsername()).isPresent()) {
7          throw new UserUsedException("There is already a user with this username: " + request.getUsername());
8      }
9      if(userRepository.findByEmail(request.getEmail()).isPresent()) {
10         throw new EmailUsedException("There is already a user with this email " + request.getEmail());
11     }
12     String encodedPassword = bCrypt.encode(request.getPassword());
13     User user = modelMapper.map(request, User.class);
14     user.setPassword(encodedPassword);
15     user.setProfilePic(getDefaultProfilePic());
16     userRepository.save(user);
17     String token = jwtUtils.createToken(user.getUsername(), user.getEmail());
18     emailService.sendEmailVerification(user,token);
19
20     ResponseRegisterVO response = modelMapper.map(user, ResponseRegisterVO.class);
21     log.info("End register method");
22     return response;
23 }
```

Ilustración 36. Método Register Service

Se realizan validaciones para verificar si ya existe un usuario registrado con el mismo nombre de usuario o dirección de correo electrónico. Si se encuentra un usuario existente, se lanzan excepciones personalizadas:

"UserUsedException" si el nombre de usuario ya está en uso y
"EmailUsedException" si el correo electrónico ya está registrado.

La contraseña proporcionada se codifica usando BCrypt para mayor seguridad.

Se crea un objeto de tipo "User" a partir de los datos proporcionados en la solicitud de registro.

Se establece una imagen de perfil predeterminada para el usuario.

El objeto "User" se guarda en la base de datos a través del repositorio "userRepository".

Se genera un token JWT (JSON Web Token) para el usuario recién registrado, que se utilizará para la verificación de correo electrónico.

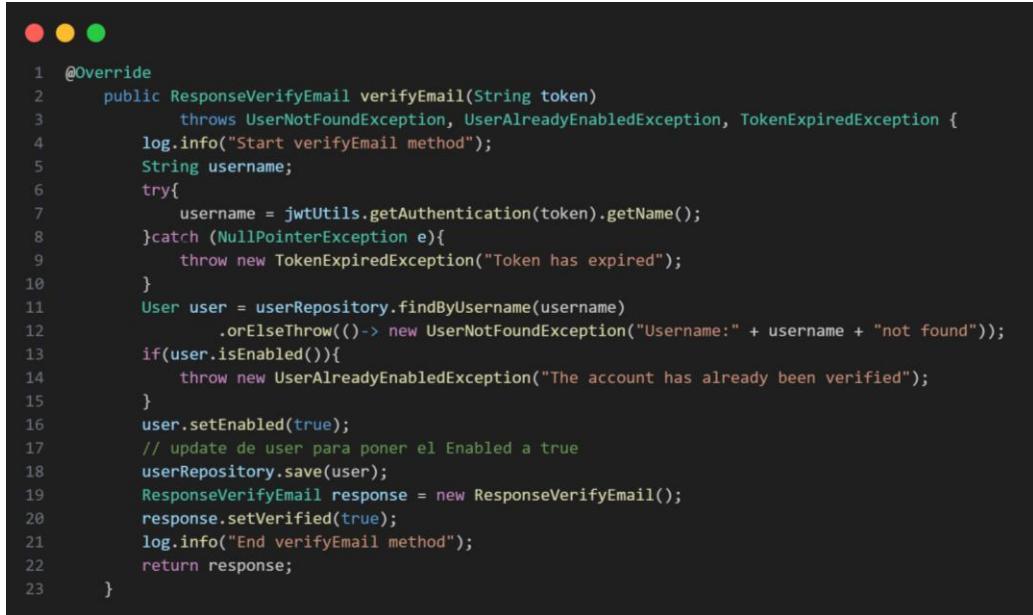
Se envía un correo electrónico de verificación al usuario utilizando un servicio de correo electrónico ("emailService").

Se mapea el objeto "User" a un objeto de respuesta "ResponseRegisterVO" que contiene los datos relevantes del usuario registrado.

Se registra información de log.

Se devuelve el objeto de respuesta "ResponseRegisterVO".

Método verifyEmail



```
1  @Override
2      public ResponseVerifyEmail verifyEmail(String token)
3          throws UserNotFoundException, UserAlreadyEnabledException, TokenExpiredException {
4      log.info("Start verifyEmail method");
5      String username;
6      try{
7          username = jwtUtils.getAuthentication(token).getName();
8      }catch (NullPointerException e){
9          throw new TokenExpiredException("Token has expired");
10     }
11     User user = userRepository.findByUsername(username)
12         .orElseThrow(()-> new UserNotFoundException("Username: " + username + " not found"));
13     if(user.isEnabled()){
14         throw new UserAlreadyEnabledException("The account has already been verified");
15     }
16     user.setEnabled(true);
17     // update de user para poner el Enabled a true
18     userRepository.save(user);
19     ResponseVerifyEmail response = new ResponseVerifyEmail();
20     response.setVerified(true);
21     log.info("End verifyEmail method");
22     return response;
23 }
```

Ilustración 37. Método verifyEmail, Service

Se obtiene el nombre de usuario asociado con el token de verificación proporcionado. El token se decodifica utilizando un objeto "jwtUtils" y se extrae el nombre de usuario.

Se busca en la base de datos un usuario con el nombre de usuario proporcionado. Si no se encuentra, se lanza una excepción personalizada "UserNotFoundException" indicando que el usuario no ha sido encontrado.

Se verifica si el usuario ya ha activado su cuenta. Si ya está habilitada, se lanza una excepción personalizada "UserAlreadyEnabledException" indicando que la cuenta ya ha sido verificada previamente.

Se establece la propiedad "enabled" del objeto "User" como true para marcar la cuenta como verificada.

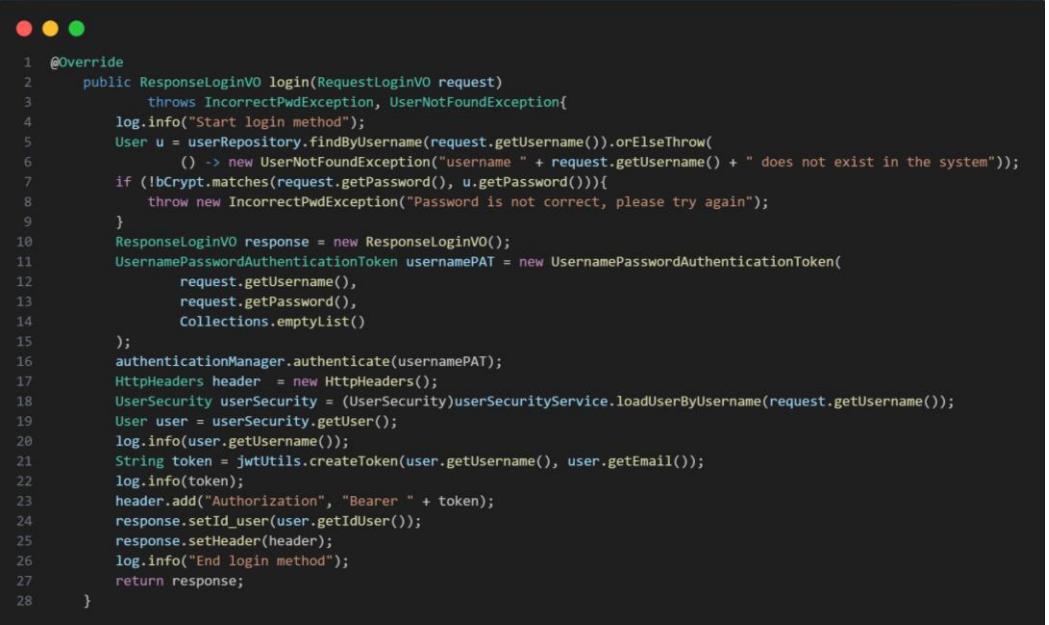
Se guarda el objeto "User" actualizado en la base de datos a través del repositorio "userRepository".

Se crea un objeto de respuesta "ResponseVerifyEmail" y se establece la propiedad "verified" como true para indicar que la verificación ha sido exitosa.

Se registra información de log.

Se devuelve el objeto de respuesta "ResponseVerifyEmail".

Método login



```

1  @Override
2  public ResponseLoginVO login(RequestLoginVO request)
3      throws IncorrectPwdException, UserNotFoundException{
4      log.info("Start login method");
5      User u = userRepository.findByUsername(request.getUsername()).orElseThrow(
6          () -> new UserNotFoundException("username " + request.getUsername() + " does not exist in the system"));
7      if (!bCrypt.matches(request.getPassword(), u.getPassword())){
8          throw new IncorrectPwdException("Password is not correct, please try again");
9      }
10     ResponseLoginVO response = new ResponseLoginVO();
11     UsernamePasswordAuthenticationToken usernamePAT = new UsernamePasswordAuthenticationToken(
12         request.getUsername(),
13         request.getPassword(),
14         Collections.emptyList()
15     );
16     authenticationManager.authenticate(usernamePAT);
17     HttpHeaders header = new HttpHeaders();
18     UserSecurity userSecurity = (UserSecurity)userSecurityService.loadUserByUsername(request.getUsername());
19     User user = userSecurity.getUser();
20     log.info(user.getUsername());
21     String token = jwtUtils.createToken(user.getUsername(), user.getEmail());
22     log.info(token);
23     header.add("Authorization", "Bearer " + token);
24     response.setUserId(user.getIdUser());
25     response.setHeader(header);
26     log.info("End login method");
27     return response;
28 }
```

Ilustración 38. Método login, Service

Se busca en la base de datos un usuario con el nombre de usuario proporcionado en la solicitud. Si no se encuentra, se lanza una excepción personalizada "UserNotFoundException" indicando que el usuario no existe en el sistema.

Se verifica si la contraseña proporcionada en la solicitud coincide con la contraseña almacenada en la base de datos para ese usuario. Si no coinciden, se lanza una excepción personalizada "IncorrectPwdException" indicando que la contraseña no es correcta.

Se crea un objeto de respuesta "ResponseLoginVO" que contendrá los datos de respuesta para el inicio de sesión.

Se crea un objeto "UsernamePasswordAuthenticationToken" con el nombre de usuario y la contraseña proporcionados en la solicitud. Este objeto se utiliza para autenticar al usuario utilizando el "authenticationManager".

Se obtiene el objeto "UserSecurity" a partir del objeto "userSecurityService" utilizando el nombre de usuario.

Se obtiene el objeto "User" del objeto "UserSecurity".

Se crea un token JWT utilizando el nombre de usuario y el correo electrónico del usuario.

Se añade el encabezado de "Authorization" con valor "Bearer: token"

Se establece el ID de usuario en el objeto de respuesta "ResponseLoginVO" y se agrega el encabezado de respuesta.

Se devuelve el objeto de respuesta "ResponseLoginVO".

Método forgotPass



```

1  @Override
2      public void forgotPass(String emailRequest) throws UserNotFoundException{
3          User user = userRepository.findByEmail(emailRequest).orElseThrow(
4              () -> new UserNotFoundException("User with email: " + emailRequest + "not found"));
5          String token = jwtUtils.createToken(user.getUsername(), user.getEmail());
6          emailService.sendPassResetEmail(user,token);
7      }

```

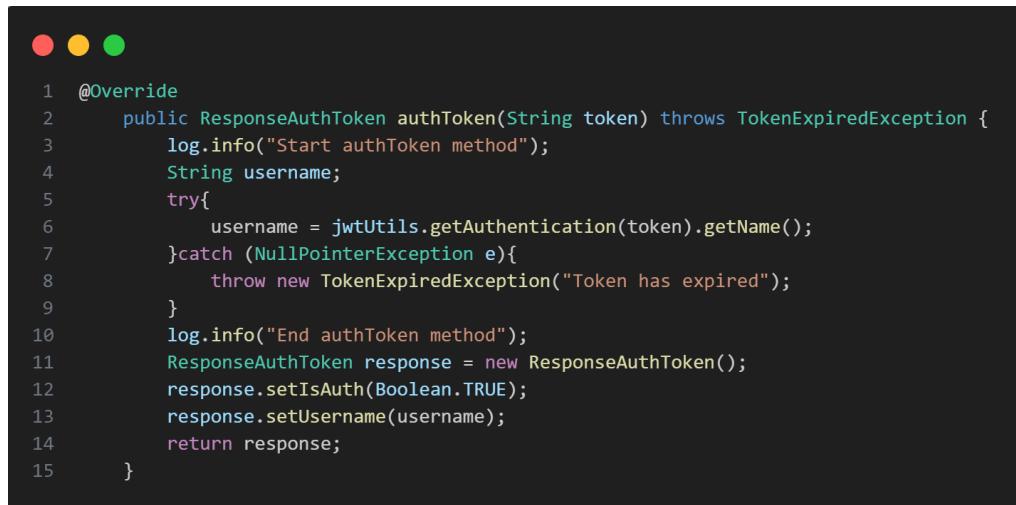
Ilustración 39. Método forgotPass, Service

Se busca en la base de datos un usuario con el correo electrónico proporcionado en la solicitud. Si no se encuentra, se lanza una excepción personalizada "UserNotFoundException" indicando que el usuario no existe en el sistema.

Se crea un token JWT utilizando el nombre de usuario y el correo electrónico del usuario.

Se utiliza el servicio de correo electrónico ("emailService") para enviar un correo electrónico al usuario con el enlace de restablecimiento de contraseña. El enlace incluirá el token JWT generado.

Método authToken



```

1  @Override
2      public ResponseAuthToken authToken(String token) throws TokenExpiredException {
3          log.info("Start authToken method");
4          String username;
5          try{
6              username = jwtUtils.getAuthentication(token).getName();
7          }catch (NullPointerException e){
8              throw new TokenExpiredException("Token has expired");
9          }
10         log.info("End authToken method");
11         ResponseAuthToken response = new ResponseAuthToken();
12         response.setIsAuth(Boolean.TRUE);
13         response.setUsername(username);
14         return response;
15     }

```

Ilustración 40. Método authToken, Service

El método recibe un token como parámetro.

Se intenta obtener el nombre de usuario asociado al token utilizando la clase "jwtUtils". Si el token ha expirado y se produce una excepción de tipo "NullPointerException", se lanza una excepción personalizada "TokenExpiredException" indicando que el token ha expirado.

Se crea un objeto de respuesta "ResponseAuthToken" y se establece la autenticación como verdadera y el nombre de usuario obtenido del token.

Se retorna el objeto de respuesta.

Método changePass

```
● ● ●
1  @Override
2      public void changePass(RequestChangePassword request)
3          throws UserNotFoundException {
4      log.info("Start changePass method");
5      User user = userRepository.findByUsername(request.getUsername())
6          .orElseThrow(() -> new UserNotFoundException("Username:" + request.getUsername() + "not found"));
7      String encodedPassword = bCrypt.encode(request.getPassword());
8      user.setPassword(encodedPassword);
9      userRepository.save(user);
10     log.info("End changePass method");
11 }
```

Ilustración 41. Método changePass, Service

El método recibe una solicitud de cambio de contraseña que contiene el nombre de usuario y la nueva contraseña.

Se busca en el repositorio de usuarios el usuario correspondiente al nombre de usuario proporcionado. Si no se encuentra el usuario, se lanza una excepción personalizada "UserNotFoundException" indicando que el usuario no ha sido encontrado.

La nueva contraseña se codifica utilizando la función de hash bcrypt.

Se actualiza la contraseña del usuario en el repositorio de usuarios.

Método createPost

```
● ● ●
1  @Override
2      public ResponseCreatePostVO createPost(String username, RequestCreatePostVO request)
3          throws UserNotFoundException, ImagePostException, EmptyFileException {
4      User user = userRepository.findByUsername(username).orElseThrow(
5          () -> new UserNotFoundException("Username:" + username + "not found"));
6      MultipartFile image = request.getPostContent();
7      if(image.isEmpty()){
8          throw new EmptyFileException("Failed to store, empty file");
9      }
10     String text = request.getHeader();
11     Post post = new Post();
12     try {
13         byte[] imageBytes = image.getBytes();
14         post.setPostContent(imageBytes);
15         post.setHeader(text);
16         post.setCreationDate(new Date(System.currentTimeMillis()));
17         post.setUser(user);
18         postRepository.save(post);
19     } catch (IOException e) {
20         throw new ImagePostException("image could not be converted to bytes");
21     }
22     return modelMapper.map(post, ResponseCreatePostVO.class);
23 }
```

Ilustración 42. Método createPost, Service

El método recibe el nombre de usuario y los detalles de la publicación en una solicitud.

Se busca en el repositorio de usuarios el usuario correspondiente al nombre de usuario proporcionado. Si no se encuentra el usuario, se lanza una excepción personalizada "UserNotFoundException" indicando que el usuario no ha sido encontrado.

Se verifica si el contenido de la publicación, que se espera que sea una imagen, está vacío. Si está vacío, se lanza una excepción personalizada "EmptyFileException" indicando que el archivo está vacío.

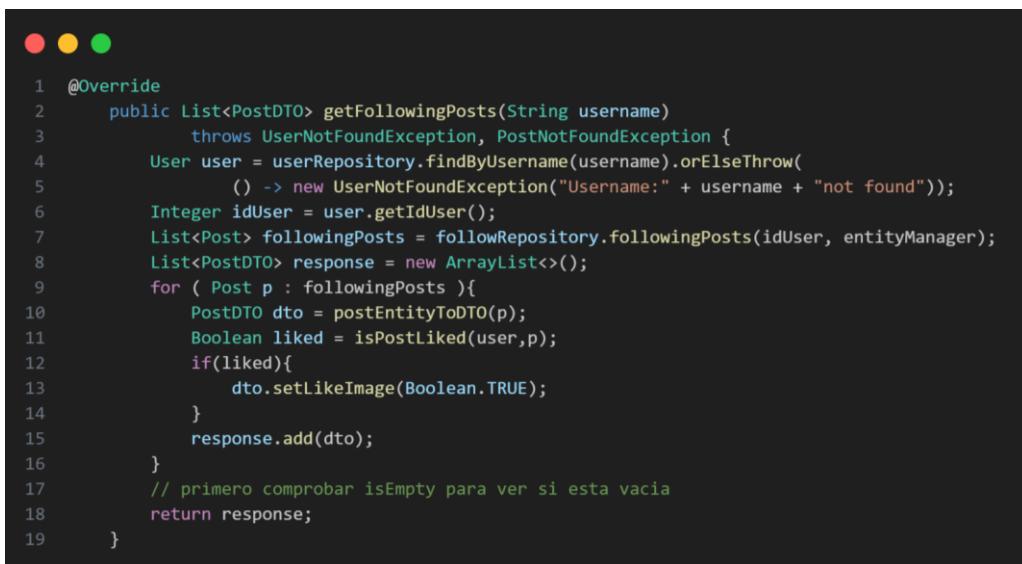
Se extrae el contenido de la publicación y se guarda en un objeto Post.

Se establecen el encabezado, la fecha de creación y el usuario asociado a la publicación.

Se guarda la publicación en el repositorio de publicaciones.

Se devuelve una respuesta que contiene los detalles de la publicación recién creada.

Método getFollowingPosts



```
1  @Override
2  public List<PostDTO> getFollowingPosts(String username)
3      throws UserNotFoundException, PostNotFoundException {
4      User user = userRepository.findByUsername(username).orElseThrow(
5          () -> new UserNotFoundException("Username:" + username + "not found"));
6      Integer idUser = user.getIdUser();
7      List<Post> followingPosts = followRepository.followingPosts(idUser, entityManager);
8      List<PostDTO> response = new ArrayList<>();
9      for ( Post p : followingPosts ){
10          PostDTO dto = postEntityToDTO(p);
11          Boolean liked = isPostLiked(user,p);
12          if(liked){
13              dto.setLikeImage(Boolean.TRUE);
14          }
15          response.add(dto);
16      }
17      // primero comprobar isEmpty para ver si esta vacia
18      return response;
19  }
```

Ilustración 43. Método getFollowingPosts, Service

El método recibe el nombre de usuario como parámetro.

Se busca en el repositorio de usuarios el usuario correspondiente al nombre de usuario proporcionado. Si no se encuentra el usuario, se lanza una excepción personalizada "UserNotFoundException" indicando que el usuario no ha sido encontrado.

Se obtiene el ID del usuario.

Se utiliza el repositorio de seguimiento (followRepository) para obtener la lista de publicaciones de los usuarios seguidos por el usuario. Esta operación se realiza a través del método "followingPosts" y se pasa el ID del usuario y el entityManager.

Se crea una lista de objetos PostDTO para almacenar las publicaciones obtenidas.

Para cada publicación obtenida, se convierte la entidad Post en un objeto PostDTO llamando al método "postEntityToDTO". Además, se verifica si el usuario actual ha dado "me gusta" a esa publicación utilizando el método "isPostLiked". Si es así, se establece una bandera "likeImage" en el objeto PostDTO.

Se agrega cada objeto PostDTO a la lista de respuesta.

Finalmente, se devuelve la lista de objetos PostDTO.

Método searchUser

```
● ● ●
1 @Override
2     public ResponseSearchUserVO searchUser(String logged, String username)
3         throws UserNotFoundException {
4     User userSearched = userRepository.find5ByUsername(username).orElseThrow(
5         () -> new UserNotFoundException("Username: " + username + " not found"));
6     User userLogged = userRepository.findByUsername(logged).orElseThrow(
7         () -> new UserNotFoundException("Username: " + logged + " not found"));
8
9     Boolean existFriendReq = friendRequestRepository.existsByUserRequesterAndUserReceiver(userLogged,userSearched);
10
11    ResponseSearchUserVO response = new ResponseSearchUserVO();
12    String profilePicBase64 = Base64.encodeBase64String(userSearched.getProfilePic());
13    Integer nFollowers = userSearched.getFollowers().size();
14    Integer nFollowing = userSearched.getFollowing().size();
15    response.setUsername(username);
16    response.setProfilePic(profilePicBase64);
17    response.setNFollowers(nFollowers);
18    response.setNFollowing(nFollowing);
19    response.setNPosts(userSearched.getPostedByUser().size());
20
21    //userSearched.getFriendRequestsMade().forEach(u -> u.getUsername().equals(username));
22    String status = "Follow";
23    if(existFriendReq){
24        status = "Pending";
25    }else{
26        Boolean existFollow = followRepository.existsByFollowerAndFollowing(userLogged,userSearched);
27        if (existFollow) {
28            status = "Followed";
29        }
30    }
31    response.setRequestStatus(status);
32    return response;
33 }
```

Ilustración 44. Método searchUser, Service

El método recibe dos parámetros: "logged" (nombre de usuario del usuario actualmente conectado) y "username" (nombre de usuario del usuario que se está buscando).

Se busca en el repositorio de usuarios el usuario correspondiente al nombre de usuario buscado. Si no se encuentra el usuario, se lanza una excepción personalizada "UserNotFoundException" indicando que el usuario no ha sido encontrado.

También se busca en el repositorio de usuarios el usuario actualmente conectado. Si no se encuentra el usuario, se lanza una excepción personalizada "UserNotFoundException" indicando que el usuario no ha sido encontrado.

Se verifica si existe una solicitud de amistad entre el usuario actualmente conectado y el usuario buscado utilizando el repositorio de solicitudes de amistad (friendRequestRepository). Se almacena el resultado en una variable booleana "existFriendReq".

Se crea un objeto ResponseSearchUserVO para almacenar la información del usuario buscado.

Se convierte la imagen de perfil del usuario buscado en una cadena Base64 y se asigna al objeto "ResponseSearchUserVO".

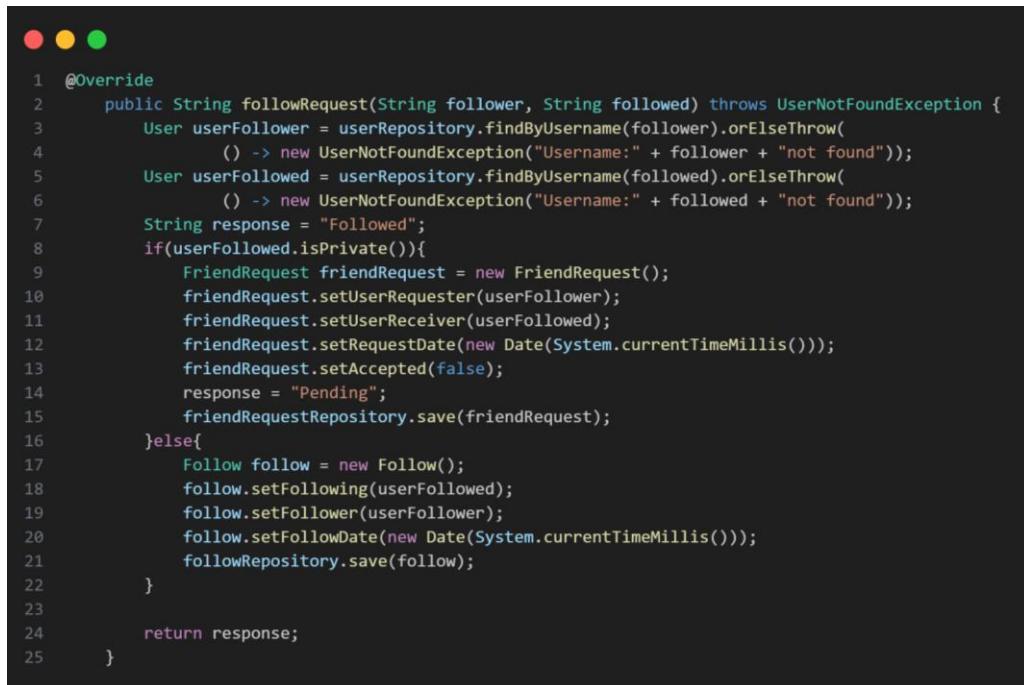
Se obtiene el número de seguidores, el número de usuarios seguidos y el número de publicaciones del usuario buscado, y se asignan al objeto "ResponseSearchUserVO".

Se establece el nombre de usuario del usuario buscado en el objeto "ResponseSearchUserVO".

Se determina el estado de la solicitud de amistad del usuario actualmente conectado con el usuario buscado. Si existe una solicitud de amistad, se establece el estado como "Pending" (pendiente). Si no existe una solicitud de amistad, se verifica si el usuario actualmente conectado sigue al usuario buscado. Si es así, se establece el estado como "Followed" (seguido). De lo contrario, se establece el estado como "Follow" (seguir).

Se devuelve el objeto "ResponseSearchUserVO" con la información del usuario buscado y el estado de la solicitud de amistad.

Método followRequest



```
1  @Override
2      public String followRequest(String follower, String followed) throws UserNotFoundException {
3          User userFollower = userRepository.findByUsername(follower).orElseThrow(
4              () -> new UserNotFoundException("Username:" + follower + "not found"));
5          User userFollowed = userRepository.findByUsername(followed).orElseThrow(
6              () -> new UserNotFoundException("Username:" + followed + "not found"));
7          String response = "Followed";
8          if(userFollowed.isPrivate()){
9              FriendRequest friendRequest = new FriendRequest();
10             friendRequest.setUserRequester(userFollower);
11             friendRequest.setUserReceiver(userFollowed);
12             friendRequest.setRequestDate(new Date(System.currentTimeMillis()));
13             friendRequest.setAccepted(false);
14             response = "Pending";
15             friendRequestRepository.save(friendRequest);
16         }else{
17             Follow follow = new Follow();
18             follow.setFollowing(userFollowed);
19             follow.setFollower(userFollower);
20             follow.setFollowDate(new Date(System.currentTimeMillis()));
21             followRepository.save(follow);
22         }
23     }
24     return response;
25 }
```

Ilustración 45. Método followRequest, Service

En este código, se implementa la funcionalidad de solicitud de seguimiento entre dos usuarios. El método "followRequest" recibe los nombres de usuario

del seguidor y del seguido como parámetros. Primero, se busca en el repositorio de usuarios al seguidor y al seguido. Si alguno de ellos no existe, se lanza una excepción de tipo “UserNotFoundException”.

A continuación, se establece la variable de respuesta como “Followed” de forma predeterminada. Si el perfil del usuario seguido es privado, se crea una nueva solicitud de amistad pendiente. Se instancia un objeto “FriendRequest” y se establecen los usuarios involucrados, la fecha de la solicitud y se marca como no aceptada. La variable de respuesta se actualiza a “Pending” para indicar que la solicitud está pendiente. Posteriormente, se guarda la solicitud en el repositorio de solicitudes de amistad.

Por otro lado, si el perfil del usuario seguido es público, se realiza directamente el seguimiento. Se crea un nuevo objeto “Follow” y se establecen el usuario seguido, el seguidor y la fecha de seguimiento. A continuación, se guarda el seguimiento en el repositorio de seguimientos.

Finalmente, se devuelve la variable de respuesta que indica si la acción fue exitosa (“Followed”) o si la solicitud está pendiente de aceptación (“Pending”).

Método unfollow



```
1  @Override
2  public String unfollow(String follower, String followed) throws UserNotFoundException {
3      User userFollower = userRepository.findByUsername(follower).orElseThrow(
4          () -> new UserNotFoundException("Username:" + follower + "not found"));
5      User userFollowed = userRepository.findByUsername(followed).orElseThrow(
6          () -> new UserNotFoundException("Username:" + followed + "not found"));
7      String response = "Follow";
8      Follow follow = followRepository.findByFollowerAndFollowing(userFollower,userFollowed);
9      if(follow!=null) {
10          followRepository.delete(follow);
11      }
12      return response;
13  }
```

Ilustración 46. Método unfollow, Service

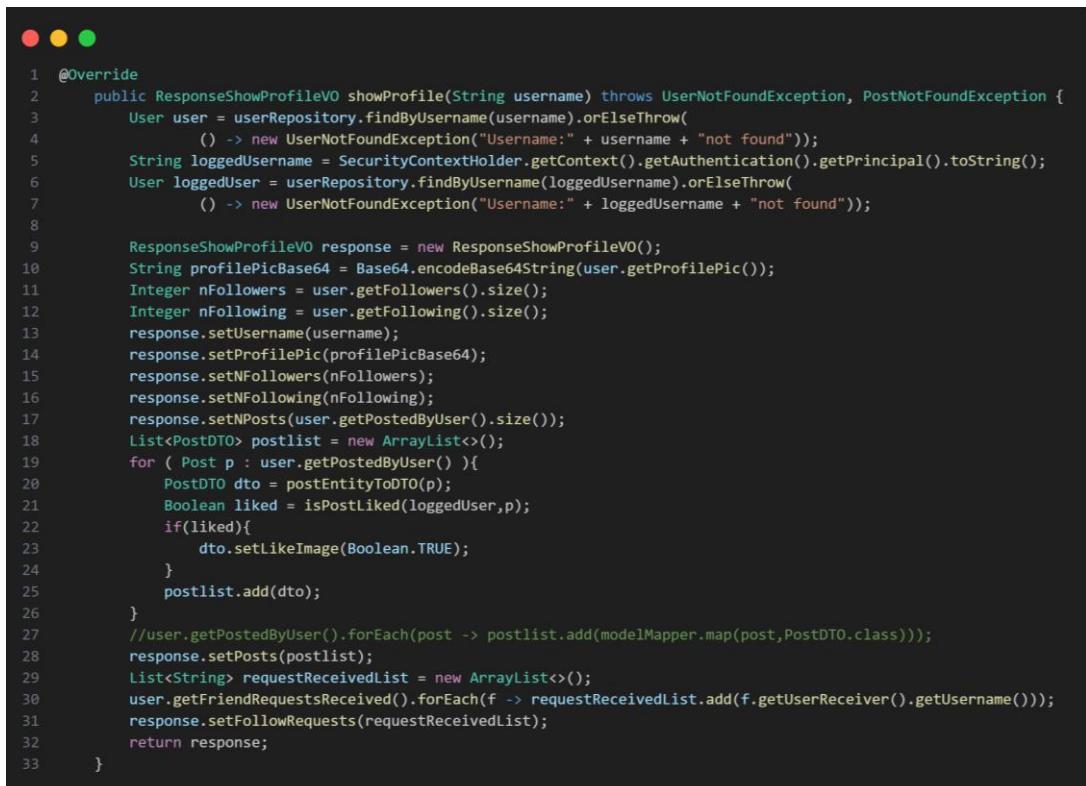
El método “unfollow” recibe los nombres de usuario del seguidor y del seguido como parámetros.

Primero, se busca en el repositorio de usuarios al seguidor y al seguido. Si alguno de ellos no existe, se lanza una excepción de tipo “UserNotFoundException”.

A continuación, se establece la variable de respuesta como “Follow” de forma predeterminada. Se busca en el repositorio de seguimientos si existe un seguimiento entre el seguidor y el seguido. Si se encuentra un seguimiento, se elimina del repositorio.

Finalmente, se devuelve la variable de respuesta que indica que se ha dejado de seguir al usuario correctamente.

Método showProfile



```
1  @Override
2  public ResponseShowProfileVO showProfile(String username) throws UserNotFoundException, PostNotFoundException {
3      User user = userRepository.findByUsername(username).orElseThrow(
4          () -> new UserNotFoundException("Username: " + username + " not found"));
5      String loggedUsername = SecurityContextHolder.getContext().getAuthentication().getPrincipal().toString();
6      User loggedUser = userRepository.findByUsername(loggedUsername).orElseThrow(
7          () -> new UserNotFoundException("Username: " + loggedUsername + " not found"));
8
9      ResponseShowProfileVO response = new ResponseShowProfileVO();
10     String profilePicBase64 = Base64.encodeBase64String(user.getProfilePic());
11     Integer nFollowers = user.getFollowers().size();
12     Integer nFollowing = user.getFollowing().size();
13     response.setUsername(username);
14     response.setProfilePic(profilePicBase64);
15     response.setNFollowers(nFollowers);
16     response.setNFollowing(nFollowing);
17     response.setNPosts(user.getPostedByUser().size());
18     List<PostDTO> postlist = new ArrayList<>();
19     for ( Post p : user.getPostedByUser() ){
20         PostDTO dto = postEntityToDTO(p);
21         Boolean liked = isPostLiked(loggedUser,p);
22         if(liked){
23             dto.setLikeImage(Boolean.TRUE);
24         }
25         postlist.add(dto);
26     }
27     //user.getPostedByUser().forEach(post -> postlist.add(modelMapper.map(post,PostDTO.class)));
28     response.setPosts(postlist);
29     List<String> requestReceivedList = new ArrayList<>();
30     user.getFriendRequestsReceived().forEach(f -> requestReceivedList.add(f.getUserReceiver().getUsername()));
31     response.setFollowRequests(requestReceivedList);
32     return response;
33 }
```

Ilustración 47. Metodo showProfile, Service

El método recibe un parámetro "username" que corresponde al nombre de usuario del usuario cuyo perfil se desea mostrar.

Se busca en el repositorio de usuarios el usuario correspondiente al nombre de usuario proporcionado. Si no se encuentra el usuario, se lanza una excepción personalizada "UserNotFoundException" indicando que el usuario no ha sido encontrado.

Se crea un objeto "ResponseShowProfileVO" para almacenar los datos del perfil del usuario.

Se convierte la imagen de perfil del usuario a una representación en base64 y se asigna al objeto de respuesta "profilePicBase64".

Se obtiene el número de seguidores y el número de usuarios a los que el usuario sigue consultando el repositorio de usuarios. Estos valores se asignan a "nFollowers" y "nFollowing" respectivamente en el objeto de respuesta.

Se asigna el nombre de usuario "username" al objeto de respuesta.

Se obtiene el número de publicaciones realizadas por el usuario consultando la lista de publicaciones asociadas al usuario. El tamaño de esta lista se asigna a "nPosts" en el objeto de respuesta.

Se crea una lista de objetos "PostDTO" para almacenar las publicaciones del usuario. Se itera sobre la lista de publicaciones asociadas al usuario y se convierte cada publicación a un objeto "PostDTO" utilizando el método "postEntityToDTO". Los objetos "PostDTO" se agregan a la lista "postlist".

Se asigna la lista de publicaciones "postlist" al objeto de respuesta.

Se crea una lista de nombres de usuario para almacenar las solicitudes de seguimiento recibidas por el usuario. Se itera sobre la lista de solicitudes de amistad recibidas por el usuario y se agrega el nombre de usuario de cada solicitante a la lista "requestReceivedList".

Se asigna la lista de solicitudes de seguimiento "requestReceivedList" al objeto de respuesta.

Se devuelve el objeto de respuesta "response", que contiene la información del perfil del usuario.

Método editUser



```
1  @Override
2  public void editUser(String username, RequestEditUserVO request) throws UserNotFoundException, UserUsedException {
3      User user = userRepository.findByUsername(username).orElseThrow(
4          () -> new UserNotFoundException("Username:" + username + " not found"));
5      if (!request.getFullscreen().equals("")) {
6          user.setFullscreen(request.getFullscreen());
7      }
8      if (!request.getUsername().equals("")) {
9          if (userRepository.findByUsername(request.getUsername()).isPresent()) {
10              throw new UserUsedException("There is already a user with this username: " + request.getUsername());
11          }
12          user.setUsername(request.getUsername());
13      }
14      userRepository.save(user);
15  }
```

Ilustración 48. Método editUser, Service

El método recibe dos parámetros: "username", que corresponde al nombre de usuario del usuario a editar, y "request", que contiene los datos de la edición en forma de objeto "RequestEditUserVO".

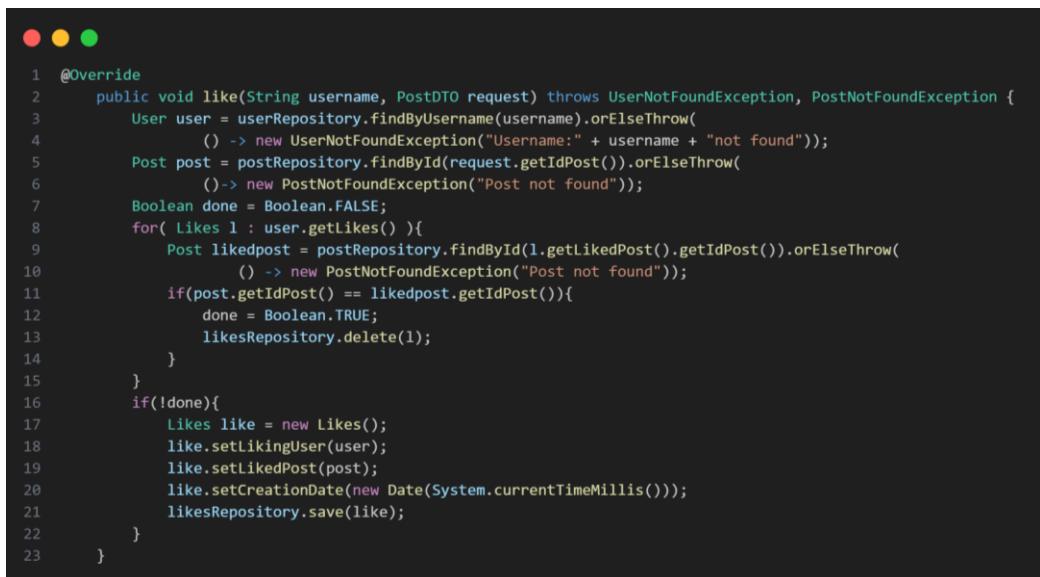
Se busca en el repositorio de usuarios el usuario correspondiente al nombre de usuario proporcionado. Si no se encuentra el usuario, se lanza una excepción personalizada "UserNotFoundException" indicando que el usuario no ha sido encontrado.

Se verifica si el campo "fullname" en el objeto "request" no está vacío. Si no está vacío, se actualiza el valor del campo "fullname" en el objeto de usuario.

Se verifica si el campo "username" en el objeto "request" no está vacío. Si no está vacío, se verifica si ya existe un usuario con el nombre de usuario proporcionado. Si existe, se lanza una excepción personalizada "UserUsedException" indicando que ya existe un usuario con ese nombre de usuario. Si no existe, se actualiza el valor del campo "username" en el objeto de usuario.

Se guarda el objeto de usuario actualizado en el repositorio de usuarios.

Método like



```

1  @Override
2      public void like(String username, PostDTO request) throws UserNotFoundException, PostNotFoundException {
3          User user = userRepository.findByUsername(username).orElseThrow(
4              () -> new UserNotFoundException("Username:" + username + "not found"));
5          Post post = postRepository.findById(request.getIdPost()).orElseThrow(
6              () -> new PostNotFoundException("Post not found"));
7          Boolean done = Boolean.FALSE;
8          for( Likes l : user.getLikes() ){
9              Post likedpost = postRepository.findById(l.getLikedPost().getIdPost()).orElseThrow(
10                  () -> new PostNotFoundException("Post not found"));
11              if(post.getIdPost() == likedpost.getIdPost()){
12                  done = Boolean.TRUE;
13                  likesRepository.delete(l);
14              }
15          }
16          if(!done){
17              Likes like = new Likes();
18              like.setLikesUser(user);
19              like.setLikedPost(post);
20              like.setCreationDate(new Date(System.currentTimeMillis()));
21              likesRepository.save(like);
22          }
23      }

```

Ilustración 49. Método like, Service

El método recibe dos parámetros: "username", que corresponde al nombre de usuario del usuario que realiza el "Me gusta", y "request", que contiene los datos de la publicación en forma de objeto "PostDTO".

Se busca en el repositorio de usuarios el usuario correspondiente al nombre de usuario proporcionado. Si no se encuentra el usuario, se lanza una excepción personalizada "UserNotFoundException" indicando que el usuario no ha sido encontrado.

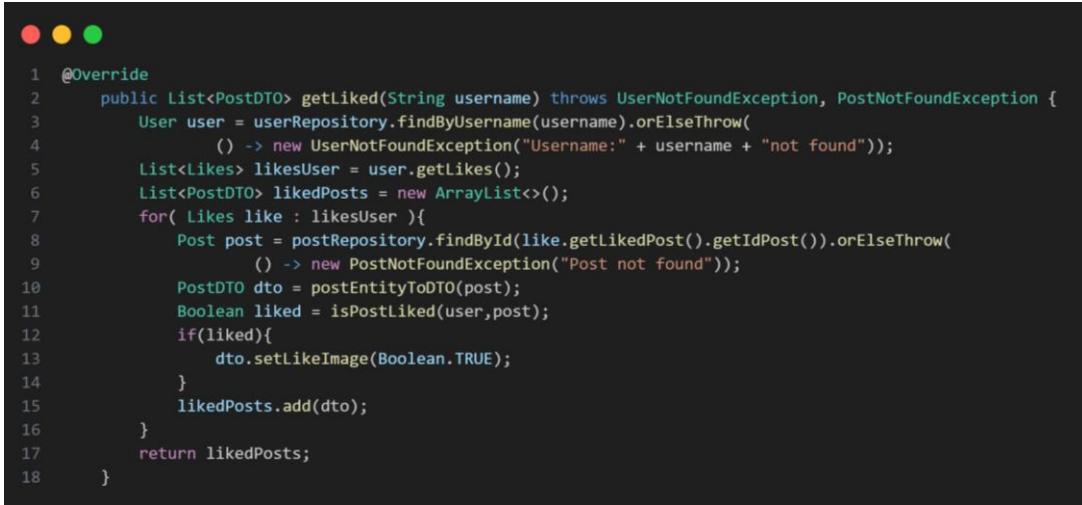
Se busca en el repositorio de publicaciones la publicación correspondiente al ID de la publicación proporcionado en el objeto "request". Si no se encuentra la publicación, se lanza una excepción personalizada "PostNotFoundException" indicando que la publicación no ha sido encontrada.

Se realiza un ciclo sobre los "Me gusta" del usuario para verificar si el usuario ya ha dado "Me gusta" a la publicación. Si se encuentra una coincidencia

entre la publicación a la que se le dio "Me gusta" y la publicación registrada en el "Me gusta" del usuario, se elimina el registro de "Me gusta" del repositorio.

Si no se encuentra una coincidencia, se crea un nuevo registro de "Me gusta" en el repositorio, indicando el usuario que dio "Me gusta" y la publicación correspondiente.

Método getLiked



```
1  @Override
2  public List<PostDTO> getLiked(String username) throws UserNotFoundException, PostNotFoundException {
3      User user = userRepository.findByUsername(username).orElseThrow(
4          () -> new UserNotFoundException("Username:" + username + " not found"));
5      List<Likes> likesUser = user.getLikes();
6      List<PostDTO> likedPosts = new ArrayList<>();
7      for( Likes like : likesUser ){
8          Post post = postRepository.findById(like.getLikedPost().getIdPost()).orElseThrow(
9              () -> new PostNotFoundException("Post not found"));
10         PostDTO dto = postEntityToDTO(post);
11         Boolean liked = isPostLiked(user,post);
12         if(liked){
13             dto.setLikeImage(Boolean.TRUE);
14         }
15         likedPosts.add(dto);
16     }
17     return likedPosts;
18 }
```

Ilustración 50. Método getLiked, Service

El método recibe un parámetro "username", que corresponde al nombre de usuario del usuario del cual se desea obtener las publicaciones que ha marcado como "Me gusta".

Se busca en el repositorio de usuarios el usuario correspondiente al nombre de usuario proporcionado. Si no se encuentra el usuario, se lanza una excepción personalizada "UserNotFoundException" indicando que el usuario no ha sido encontrado.

Se obtiene la lista de registros de "Me gusta" del usuario.

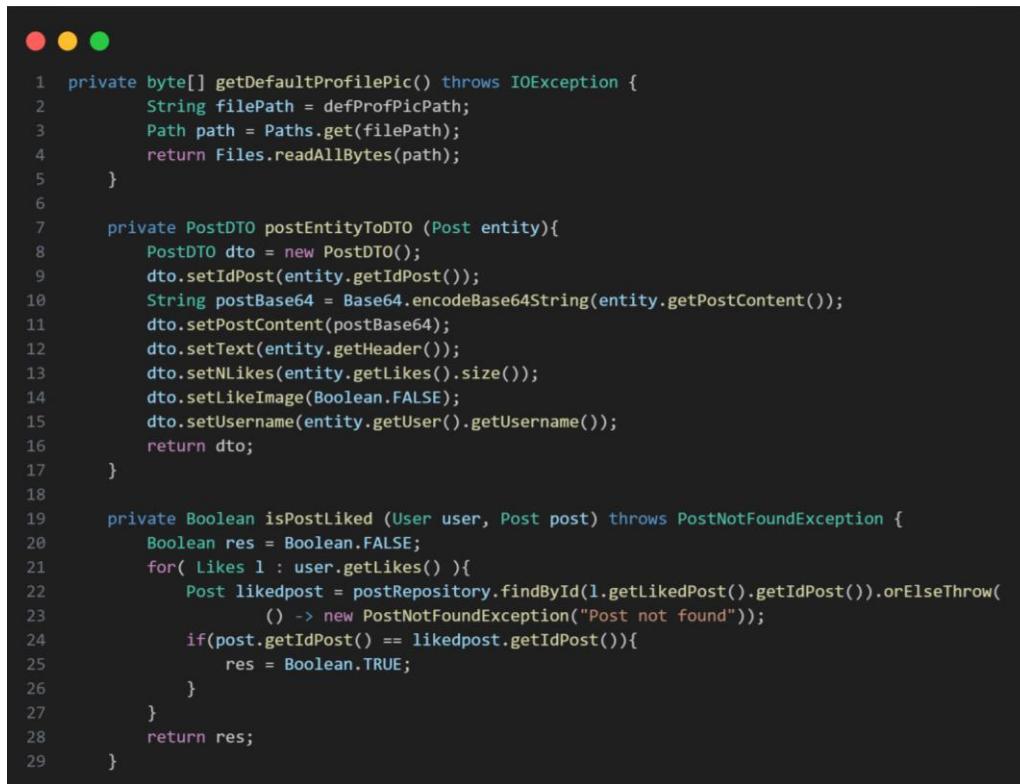
Se crea una lista vacía llamada "likedPosts" para almacenar las publicaciones que han sido marcadas como "Me gusta" por el usuario.

Se realiza un ciclo sobre los registros de "Me gusta" del usuario. Para cada registro, se busca la publicación correspondiente en el repositorio de publicaciones utilizando el ID de la publicación registrada en el "Me gusta". Si no se encuentra la publicación, se lanza una excepción personalizada "PostNotFoundException" indicando que la publicación no ha sido encontrada.

Se crea un objeto "PostDTO" a partir de la publicación encontrada y se agrega a la lista "likedPosts".

Se devuelve la lista "likedPosts" que contiene las publicaciones que han sido marcadas como "Me gusta" por el usuario.

Métodos privados



```
1  private byte[] getDefaultProfilePic() throws IOException {
2      String filePath = defProfPicPath;
3      Path path = Paths.get(filePath);
4      return Files.readAllBytes(path);
5  }
6
7  private PostDTO postEntityToDTO (Post entity){
8      PostDTO dto = new PostDTO();
9      dto.setIdPost(entity.getIdPost());
10     String postBase64 = Base64.encodeBase64String(entity.getPostContent());
11     dto.setPostContent(postBase64);
12     dto.setText(entity.getHeader());
13     dto.setNLikes(entity.getLikes().size());
14     dto.setLikeImage(Boolean.FALSE);
15     dto.setUsername(entity.getUser().getUsername());
16     return dto;
17 }
18
19 private Boolean isPostLiked (User user, Post post) throws PostNotFoundException {
20     Boolean res = Boolean.FALSE;
21     for( Likes l : user.getLikes() ){
22         Post likedpost = postRepository.findById(l.getLikedPost().getIdPost()).orElseThrow(
23             () -> new PostNotFoundException("Post not found"));
24         if(post.getIdPost() == likedpost.getIdPost()){
25             res = Boolean.TRUE;
26         }
27     }
28     return res;
29 }
```

Ilustración 51. Metodos privados Service

Métodos privados auxiliares: obtener la foto de perfil predeterminada, mapear la entidad *Post* a *PostDTO* y, por último, saber si el usuario con sesión iniciada ha dado “Me Gusta” a la publicación enviada

5.1.3.2 EmailService

El servicio *EmailService* se encarga de procesar las solicitudes relacionadas con él envío de correos electrónicos. Este servicio implementa los siguientes métodos:

Método sendEmailVerification

```
1 @Override
2     public void sendEmailVerification(User user, String token) throws MessagingException{
3         log.info("Start sendEmailVerification method");
4         String subject = "E-mail verification";
5         String url = confirmationUrl + "?token=" + token;
6         String emailBody = "Hello " + user.getUsername() + ",\n\n"
7             + "Thank you for registering on SocialHub. Click on the link below to verify your email address:\n\n"
8             + url + "\n\n"
9             + "If you have not requested this email, please ignore it.\n\n"
10            + "Regards,\n"
11            + "The SocialHub team";
12
13         SimpleMailMessage email = new SimpleMailMessage();
14         email.setFrom(emailSH);
15         email.setTo(user.getEmail());
16         email.setSubject(subject);
17         email.setText(emailBody);
18
19         log.info("Verification email is sent");
20         javaMailSender.send(email);
21         log.info("End sendEmailVerification method");
22     }
```

Ilustración 52. Método sendEmailVerification

El método recibe dos parámetros: "user", que representa al usuario al cual se enviará el correo electrónico, y "token", que es un token único generado para la verificación de correo electrónico.

Se crea el contenido del correo electrónico, que incluye un saludo al usuario, un mensaje de agradecimiento por registrarse en la red social y un enlace de verificación. El enlace contiene el token necesario para la verificación.

Se crea un objeto "SimpleMailMessage" que representa el correo electrónico a enviar. Se establece el remitente, el destinatario, el asunto y el contenido del correo.

Se envía el correo electrónico utilizando el objeto "javaMailSender".

Método sendPassResetEmail

```
1 @Override
2     public void sendPassResetEmail(User user, String token) throws MessagingException{
3         log.info("Start sendPassResetEmail method");
4         String subject = "Password Reset email";
5         String url = resetPassUrl + "?token=" + token;
6         String emailBody = "Hello " + user.getUsername() + ",\n\n"
7             + "We received a request to reset your account password. Click on the link below to set a new password:\n\n"
8             + url + "\n\n"
9             + "If you did not request a password reset, please ignore this email and your password will remain the same.\n\n"
10            + "Regards,\n"
11            + "The SocialHub team";
12
13         SimpleMailMessage email = new SimpleMailMessage();
14         email.setFrom(emailSH);
15         email.setTo(user.getEmail());
16         email.setSubject(subject);
17         email.setText(emailBody);
18
19         log.info("Password reset email is sent");
20         javaMailSender.send(email);
21         log.info("End sendPassResetEmail method");
22     }
```

Ilustración 53. Método sendPassEmail

El método recibe dos parámetros: "user", que representa al usuario al cual se enviará el correo electrónico, y "token", que es un token único generado para el restablecimiento de contraseña.

Se crea el contenido del correo electrónico, que incluye un saludo al usuario, un mensaje informando que se ha recibido una solicitud para restablecer la contraseña y un enlace para establecer una nueva contraseña. El enlace contiene el token necesario para el restablecimiento.

Se crea un objeto "SimpleMailMessage" que representa el correo electrónico a enviar. Se establece el remitente, el destinatario, el asunto y el contenido del correo.

Se envía el correo electrónico utilizando el objeto "javaMailSender".

5.1.4 Integración con la base de datos

En el contexto de la integración de datos en el proyecto, se emplean clases *Repository* que extienden la interfaz "JpaRepository". Estos *Repositories* son componentes clave que facilitan la interacción entre la capa de persistencia y el resto de la aplicación.

Cada uno de los *Repositories* utilizados en el proyecto desempeña un papel específico en el manejo de entidades y operaciones relacionadas con ellas. A continuación, se detallan los *Repositories* mencionados:

UserRepository: Este *Repository* está diseñado para trabajar con entidades de tipo User. Proporciona métodos para realizar operaciones como buscar usuarios por nombre de usuario o correo electrónico, guardar nuevos usuarios, actualizar información de usuarios existentes y eliminar usuarios.

PostRepository: Como su nombre indica, este *Repository* se encarga de la gestión de entidades de tipo Post. Permite realizar operaciones relacionadas con los posts, como guardar nuevos posts, buscar posts por ID o por criterios específicos, y eliminar posts.

LikesRepository: Este *Repository* se encarga de las operaciones relacionadas con los *likes* en el sistema. Permite realizar acciones como agregar un nuevo *like* a un post, verificar si un usuario ha dado *like* a un post específico y eliminar un *like* existente.

FriendRequestRepository: El propósito de este *Repository* es gestionar las solicitudes de amistad entre usuarios. Permite realizar operaciones como crear una nueva solicitud de amistad, buscar solicitudes por usuarios involucrados y eliminar solicitudes existentes.

FollowRepository: Este *Repository* está dedicado a las relaciones de seguimiento entre usuarios. Permite realizar acciones como establecer una nueva relación de seguimiento entre un seguidor y un usuario seguido, verificar si un usuario está siguiendo a otro y eliminar relaciones de seguimiento existentes.

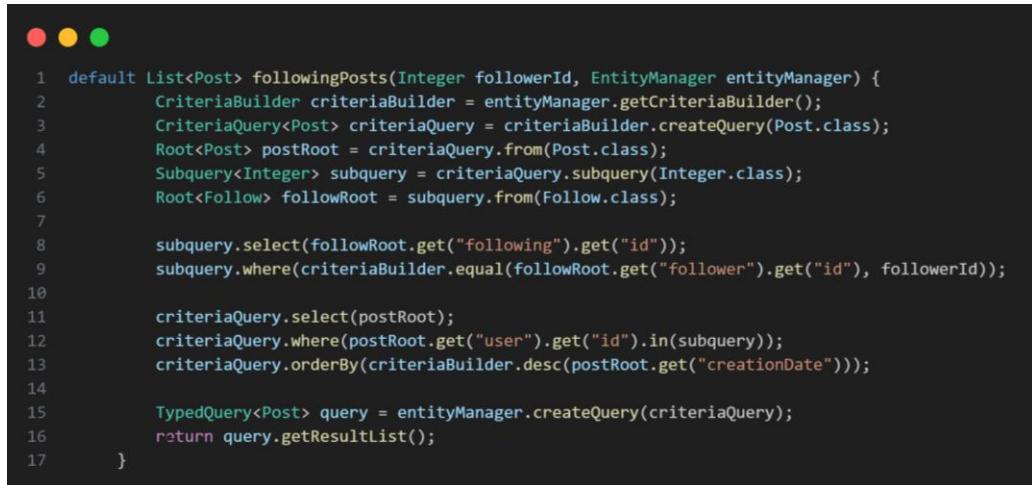
Es importante mencionar la formación de queries en los repositorios. En el proyecto se realizan queries de dos formas distintas:



```
1 @Query("SELECT fr FROM FriendRequest fr WHERE fr.userRequester.idUser = :requesterId AND fr.userReceiver.idUser = :receiverId")
2     Optional<FriendRequest> findByRequesterIdAndReceiverId(@Param("requesterId") int requesterId, @Param("receiverId") int receiverId);
```

Ilustración 54. Query 1

Esta anotación `@Query` se utiliza para definir una consulta personalizada en lenguaje SQL o JPQL (Java Persistence Query Language). En este caso, la consulta busca una entidad `FriendRequest` en función de los identificadores del solicitante (`requesterId`) y del receptor (`receiverId`).



```
1 default List<Post> followingPosts(Integer followerId, EntityManager entityManager) {
2     CriteriaBuilder criteriaBuilder = entityManager.getCriteriaBuilder();
3     CriteriaQuery<Post> criteriaQuery = criteriaBuilder.createQuery(Post.class);
4     Root<Post> postRoot = criteriaQuery.from(Post.class);
5     Subquery<Integer> subquery = criteriaQuery.subquery(Integer.class);
6     Root<Follow> followRoot = subquery.from(Follow.class);
7
8     subquery.select(followRoot.get("following").get("id"));
9     subquery.where(criteriaBuilder.equal(followRoot.get("follower").get("id"), followerId));
10
11    criteriaQuery.select(postRoot);
12    criteriaQuery.where(postRoot.get("user").get("id").in(subquery));
13    criteriaQuery.orderBy(criteriaBuilder.desc(postRoot.get("creationDate")));
14
15    TypedQuery<Post> query = entityManager.createQuery(criteriaQuery);
16    return query.getResultList();
17 }
```

Ilustración 55. Query 2

Este método utiliza el API Criteria de JPA para obtener una lista de publicaciones (`Post`) de los usuarios que un seguidor específico sigue. Toma como parámetros el identificador del seguidor (`followerId`) y una instancia de `EntityManager`.

Estos *Repositories* se utilizan como interfaces de acceso a la capa de persistencia de datos y proporcionan métodos predefinidos para realizar operaciones comunes de consulta y manipulación de entidades. De esta manera, simplifican el acceso y la manipulación de datos en el sistema,

permitiendo una integración más eficiente y efectiva de los datos en la aplicación.

5.1.5 Excepciones

En este proyecto se han creado varias excepciones para manejar de manera detallada y contextualizada los diferentes tipos de excepciones. Estas excepciones son:

EmailUsedException: Se lanza cuando se intenta utilizar un correo electrónico que ya está en uso.

UserUsedException: Se lanza cuando se intenta crear un usuario con un nombre de usuario que ya está en uso.

UserNotFoundException: Se lanza cuando no se encuentra un usuario específico.

UserAlreadyEnabledException: Se lanza cuando se intenta habilitar un usuario que ya está habilitado.

TokenExpiredException: Se lanza cuando un token ha expirado y no se puede utilizar.

ImagePostException: Se lanza cuando se produce un error relacionado con las imágenes en las publicaciones.

PostNotFoundException: Se lanza cuando no se encuentra una publicación específica.

IncorrectPwdException: Se lanza cuando se proporciona una contraseña incorrecta.

EmptyFileException: Se lanza cuando se intenta cargar un archivo vacío.

FriendRequestNotFoundException: Se lanza cuando no se encuentra una solicitud de amistad específica.

Estas excepciones permiten capturar y manejar de manera precisa situaciones específicas dentro del flujo de ejecución del programa, proporcionando mensajes de error descriptivos y facilitando la depuración y resolución de problemas.

5.2 Desarrollo del Frontend con Angular

5.2.1 Configuración del proyecto

Para comenzar, se creó el proyecto Angular utilizando el comando “ng new SocialHubApp” del Angular CLI. Una vez creado el proyecto, se agregó la referencia a Bootstrap para tener todo listo para el desarrollo de la aplicación. Esto permitirá utilizar las funcionalidades y estilos proporcionados por Bootstrap de manera sencilla y eficiente en el proyecto de Angular. Al tener esta configuración inicial, se establece una base sólida para desarrollar la interfaz de usuario y las funcionalidades de la aplicación de manera más rápida y organizada.

5.2.2 Implementación de las vistas y componentes

En el desarrollo del Frontend, se estructura en un módulo principal llamado “app.module” que engloba el funcionamiento general de la aplicación. Además, se dividen en tres submódulos principales: “auth.module”, “home.module” y “shared.module”. A continuación, se detalla la implementación de los componentes que componen cada uno de estos módulos:

5.2.2.1 Módulo “auth.module”

Este módulo se encarga de la autenticación y autorización de los usuarios. Además, se define un archivo de enrutamiento de los componentes pertenecientes a este módulo:



```
1 const routes: Routes = [
2   {
3     path: '',
4     children: [
5       { path: 'loginhome', component: LoginhomeComponent },
6       { path: 'register', component: RegisterComponent },
7       { path: 'login', component: LoginComponent },
8       { path: 'verify-email', component: EmailverificationComponent },
9       { path: 'forgot-pwd', component: ForgotPwdComponent},
10      { path: 'reset-pwd', component: ResetpasswordComponent},
11      { path: '**', redirectTo: 'loginhome' }
12    ]
13  },
14];
15
16 @NgModule({
17   imports: [RouterModule.forChild(routes)],
18   exports: [RouterModule]
19 })
20 export class AuthRoutingModule { }
```

Ilustración 56. auth-routing

La variable “routes” es un arreglo de objetos que contiene las rutas definidas para el módulo de autenticación.

Cada objeto en el arreglo representa una ruta y contiene la propiedad “path”, que especifica la URL asociada a la ruta, y la propiedad “component”, que especifica el componente que se debe cargar cuando se accede a esa ruta.

El “path:” representa la ruta raíz del módulo de autenticación.

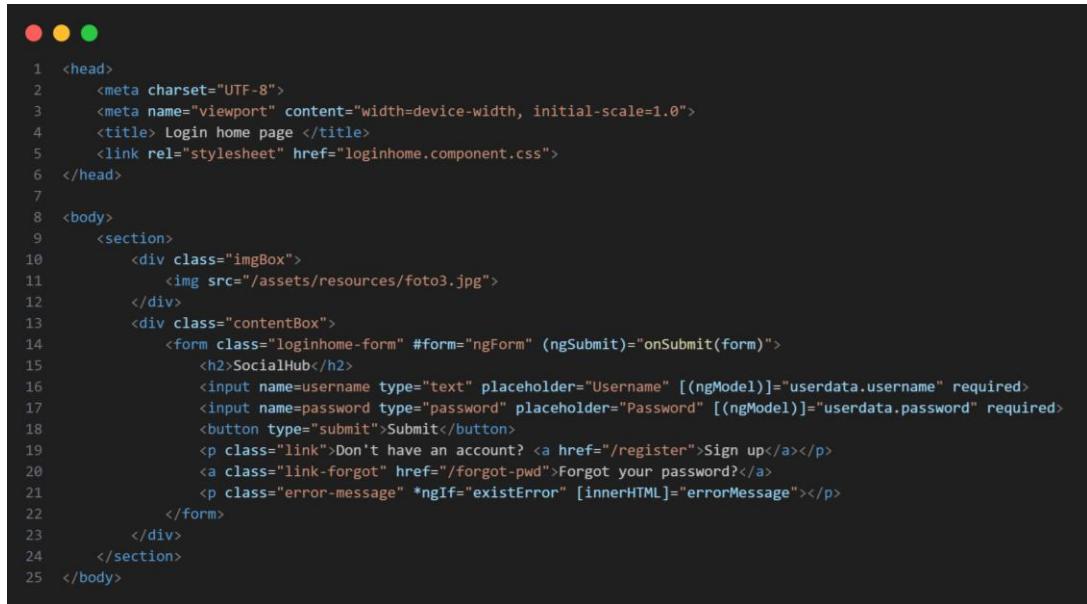
Dentro del objeto de la ruta raíz, se utiliza la propiedad “children” para definir rutas secundarias o hijas. Cada ruta secundaria tiene su propia URL y componente asociado.

El *path: '*' es una ruta comodín que se utiliza para redirigir a la ruta “loginhome” en caso de que la URL no coincida con ninguna de las rutas definidas anteriormente.*

El decorador “@NgModule” se utiliza para configurar el módulo de enrutamiento. Se importa el módulo “RouterModule” y se configuran las rutas definidas en la variable “routes”. Luego, se exporta el módulo “RouterModule” para que esté disponible para otros módulos de la aplicación.

A continuación, se expondrán los siguientes componentes:

LoginhomeComponent: Componente encargado de la página de inicio de sesión principal.



```
1 <head>
2   <meta charset="UTF-8">
3   <meta name="viewport" content="width=device-width, initial-scale=1.0">
4   <title> Login home page </title>
5   <link rel="stylesheet" href="loginhome.component.css">
6 </head>
7
8 <body>
9   <section>
10    <div class="imgBox">
11      
12    </div>
13    <div class="contentBox">
14      <form class="loginhome-form" #form="ngForm" (ngSubmit)="onSubmit(form)">
15        <h2>SocialHub</h2>
16        <input name=username type="text" placeholder="Username" [(ngModel)]="userdata.username" required>
17        <input name=password type="password" placeholder="Password" [(ngModel)]="userdata.password" required>
18        <button type="submit">Submit</button>
19        <p class="link-forget" href="/register">Sign up</a></p>
20        <a class="link-forgot" href="/forgot-pwd">Forgot your password?</a>
21        <p class="error-message" *ngIf="existError" [innerHTML]="errorMessage"></p>
22      </form>
23    </div>
24  </section>
25 </body>
```

Ilustración 57. loginhome HTML

El componente HTML se divide en 2 secciones: La mitad izquierda la cual muestra una fotografía, y la mitad derecha la cual muestra el formulario de inicio de sesión. En la pantalla se podrá cambiar a la pantalla de registro o solicitar el cambio de contraseña.

Se utilizan estructuras como “div”(divisor), “img”(imagen), “form”(formulario), h2(cabecera), input (input del formulario) o p(texto).

También se aprecia el uso de “(ngSubmit)”, el cual referencia al método “onSubmit(formulario)”, el cual se ejecutara al enviar el formulario.

Además, se utilizan estructuras de control como “if” -> “*ngIf”, u otro tipo de estructuras como “[innerHTML]” para introducir el valor de la variable “errorMessage” dentro de la pantalla.

```
 1  export class LoginhomeComponent {  
 2  
 3      userdata: Login = {  
 4          username: '',  
 5          password: ''  
 6      };  
 7  
 8      existError: boolean = false;  
 9      errorMessage: string = '';  
10  
11      errors: string[] = [  
12          'All fields must be filled in'  
13      ]  
14  
15      constructor(  
16          private authService: AuthService,  
17          private router: Router  
18      ) {  
19      }  
20  
21      onSubmit(form: NgForm) {  
22          this.existError = false;  
23          console.log('Form: ', this.userdata);  
24          if(isFullFilledLogin(this.userdata)){  
25              this.authService.loginUser(this.userdata)  
26                  .subscribe(response => {  
27                      this.router.navigate(['/home']);  
28                  },  
29                  (error) => {  
30                      if (error && error.error) {  
31                          this.errorMessage = error.error.message;  
32                      } else {  
33                          this.errorMessage = 'An error occurred.';  
34                      }  
35                      this.existError = true;  
36                      console.log(this.errorMessage);  
37                  }  
38          );  
39      }else{  
40          this.errorMessage = this.errors[0];  
41          this.existError = true;  
42      }  
43  }  
44  
45 }
```

Ilustración 58. loginhome TS

Componente “ts”: Contiene una propiedad llamada “userdata” que almacena los datos de inicio de sesión del usuario. Además, tiene variables como “existError” y “errorMessage” para controlar y mostrar mensajes de error en caso de problemas durante el inicio de sesión.

En el método “onSubmit”, se valida el formulario y se llama al servicio de autenticación “(authService)” para realizar la solicitud de inicio de sesión. Si la solicitud es exitosa, el usuario es redirigido a la página de inicio (/home). En caso de error, se captura el mensaje de error y se muestra en la página. Si el formulario no está completo, se muestra un mensaje indicando que todos los campos deben ser llenados.

LoginComponent: Componente encargado de la página de inicio de sesión.



```
1 <head>
2   <meta charset="UTF-8">
3   <meta name="viewport" content="width=device-width, initial-scale=1.0">
4   <title> Login home page </title>
5   <link rel="stylesheet" href="login.component.css">
6 </head>
7
8 <body>
9   <div class="formbox">
10    <form class="login-form" #form="ngForm" (ngSubmit)="onSubmit(form)">
11      <h2>Sign in</h2>
12      <input name=username type="text" placeholder="Username" [(ngModel)]="userdata.username" required>
13      <input name=password type="password" placeholder="Password" [(ngModel)]="userdata.password" required>
14      <button type="submit">Submit</button>
15      <p class="register-link">Don't have an account? <a href="/register">Sign up</a></p>
16      <a class="link-forgot" href="/forgot-pwd">Forgot your password?</a>
17      <p class="error-message" *ngIf="existError" [innerHTML]="errorMessage"></p>
18    </form>
19  </div>
20 </body>
```

Ilustración 59. login HTML

El código es una página de inicio de sesión con un formulario básico que incluye campos para el nombre de usuario y la contraseña. Los usuarios pueden enviar el formulario para iniciar sesión, y también se proporcionan enlaces para registrarse y restablecer la contraseña en caso de ser necesario.

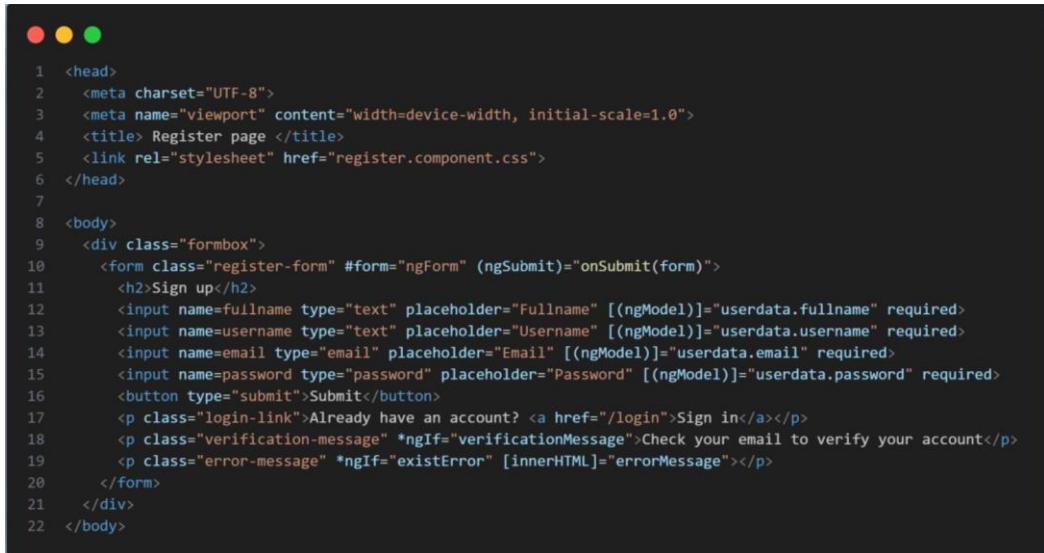
```
 1  export class LoginComponent {
 2
 3      userdata: Login = {
 4          username:'',
 5          password: ''
 6      };
 7
 8      existError: boolean = false;
 9      errorMessage: string = '';
10
11      errors: string[] = [
12          'All fields must be filled in'
13      ];
14
15      constructor(
16          private authService: AuthService,
17          private router: Router
18      ) {
19      }
20
21      onSubmit(form: NgForm) {
22          console.log('Form: ', form.value);
23          if(isFullFilledLogin(this.userdata)){
24              this.authService.loginUser(this.userdata)
25                  .subscribe(response => {
26                      this.router.navigate(['/home']);
27                  },
28                  (error) => {
29                      if (error && error.error) {
30                          this.errorMessage = error.error.message;
31                      } else {
32                          this.errorMessage = 'An error occurred.';
33                      }
34                      this.existError = true;
35                      console.log(this.errorMessage);
36                  }
37              );
38          }else{
39              this.errorMessage = this.errors[0];
40              this.existError = true;
41          }
42      }
}
```

Ilustración 60. login TS

El componente “ts” contiene propiedades para almacenar los datos de inicio de sesión del usuario, como el nombre de usuario y la contraseña. También incluye propiedades para manejar errores y mensajes de error.

En el método “onSubmit”, se realiza una validación de los campos de inicio de sesión y luego se llama al método “loginUser” del servicio “authService” para iniciar sesión. Si la solicitud es exitosa, se redirige al usuario a la página de inicio. En caso de error, se muestra un mensaje de error apropiado.

RegisterComponent: Componente para el registro de nuevos usuarios.



```
1 <head>
2   <meta charset="UTF-8">
3   <meta name="viewport" content="width=device-width, initial-scale=1.0">
4   <title> Register page </title>
5   <link rel="stylesheet" href="register.component.css">
6 </head>
7
8 <body>
9   <div class="formbox">
10    <form class="register-form" #form="ngForm" (ngSubmit)="onSubmit(form)">
11      <h2>Sign up</h2>
12      <input name="fullname" type="text" placeholder="Fullname" [(ngModel)]="userdata.fullname" required>
13      <input name="username" type="text" placeholder="Username" [(ngModel)]="userdata.username" required>
14      <input name="email" type="email" placeholder="Email" [(ngModel)]="userdata.email" required>
15      <input name="password" type="password" placeholder="Password" [(ngModel)]="userdata.password" required>
16      <button type="submit">Submit</button>
17      <p class="login-link">Already have an account? <a href="/login">Sign in</a></p>
18      <p class="verification-message" *ngIf="verificationMessage">Check your email to verify your account</p>
19      <p class="error-message" *ngIf="existError" [innerHTML]="errorMessage"></p>
20    </form>
21  </div>
22 </body>
```

Ilustración 61. Register HTML

En el cuerpo del HTML, se muestra un formulario de registro. Los campos incluyen nombre completo, nombre de usuario, correo electrónico y contraseña. Al enviar el formulario, se llama al método “onSubmit” definido en el componente.

Dentro del formulario, también hay enlaces para iniciar sesión si ya se tiene una cuenta. Además, hay mensajes condicionales que se muestran en caso de éxito o error en el registro.



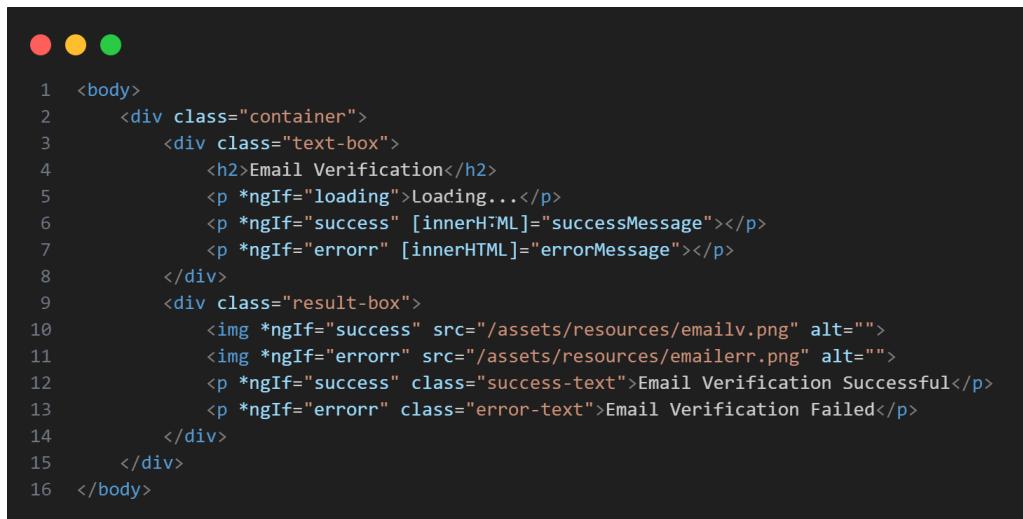
```
1  export class RegisterComponent {
2
3      userdata: Register = {
4          fullname: '',
5          username:'',
6          email: '',
7          password: ''
8      };
9      verificationMessage: boolean = false;
10     existError: boolean = false;
11
12    errors: string[] = [
13        'All fields must be filled in',
14        'Email does not have a valid format',
15        'Passwords must be at least <br> 8 characters long.'
16    ];
17
18    errorMessage: string = '';
19
20    constructor(private authService: AuthService) {
21    }
22
23    onSubmit(form: NgForm) {
24        this.existError = false;
25        console.log('Form: ', form.value);
26        if(isFullFilled(this.userdata)){
27            if(validateEmail(this.userdata.email)){
28                if(this.userdata.password.length >= 8){
29                    this.authService.registerUser(this.userdata);
30                    this.verificationMessage = true;
31                }else{
32                    this.errorMessage = this.errors[2];
33                    this.existError = true;
34                }
35            }else{
36                this.errorMessage = this.errors[1];
37                this.existError = true;
38            }
39        }else{
40            this.errorMessage = this.errors[0];
41            this.existError = true;
42        }
43
44    }
45}
46
47 }
```

Ilustración 62. register TS

El componente contiene una propiedad “userdata” que representa los datos ingresados por el usuario, como el nombre completo, nombre de usuario, correo electrónico y contraseña. También hay propiedades adicionales como “verificationMessage” y “existError” para controlar mensajes de verificación y errores.

El método “onSubmit” se ejecuta cuando se envía el formulario. Dentro de este método, se realiza una serie de validaciones para verificar que todos los campos estén completos, que el correo electrónico tenga un formato válido y que la contraseña tenga al menos 8 caracteres. Si todas las validaciones son exitosas, se llama al método “registerUser” del servicio “AuthService” para registrar al usuario y se muestra un mensaje de verificación. Si hay errores de validación, se muestra un mensaje de error correspondiente.

EmailverificationComponent: Componente para verificar la dirección de correo electrónico.



```
1 <body>
2   <div class="container">
3     <div class="text-box">
4       <h2>Email Verification</h2>
5       <p *ngIf="loading">Loading...</p>
6       <p *ngIf="success" [innerHTML]="successMessage"></p>
7       <p *ngIf="error" [innerHTML]="errorMessage"></p>
8     </div>
9     <div class="result-box">
10      
11      
12      <p *ngIf="success" class="success-text">Email Verification Successful</p>
13      <p *ngIf="error" class="error-text">Email Verification Failed</p>
14    </div>
15  </div>
16 </body>
```

Ilustración 63. email-verification HTML

En la sección "text-box", se muestra un título "Email Verification" y se condicionalmente muestran mensajes dependiendo de las variables "loading", "success" y "error". Si "loading" es verdadero, se muestra el mensaje "Loading...". Si "success" es verdadero, se muestra el contenido de "successMessage", que es un mensaje de éxito. Si "error" es verdadero, se muestra el contenido de "errorMessage", que es un mensaje de error.

En la sección "result-box", se muestran imágenes y mensajes dependiendo de las variables "success" y "error". Si "success" es verdadero, se muestra una imagen de verificación exitosa y un mensaje "Email Verification Successful". Si "error" es verdadero, se muestra una imagen de error y un mensaje "Email Verification Failed".

```

1  export class EmailverificationComponent implements OnInit {
2      loading: boolean = true;
3      success: boolean = false;
4      error: boolean = false;
5      // Mensaje de éxito
6      successMessage: string = "Your account has been successfully verified."
7      + "<br>" + "You can now log in with your account and start enjoying SocialHub.";
8      // Depende del backend
9      errorMessage: string = "Error verifying your email address";
10
11     constructor(private route: ActivatedRoute, private authService: AuthService) { }
12
13     ngOnInit(): void {
14         this.loading = true;
15
16         // Obtener el token de verificación del correo electrónico de los parámetros de la URL
17         this.route.queryParams.subscribe(params => {
18             const token = params['token']; // Accede al valor del parámetro 'token'
19             console.log(token);
20             // Hacer la solicitud de verificación de correo electrónico al backend
21             this.authService.verifyEmail(token).subscribe(
22                 (response) => {
23                     this.loading = false;
24                     this.success = true;
25                     this.successMessage = response.message;
26                 },
27                 (error) => {
28                     this.loading = false;
29                     this.error = true;
30                     this.errorMessage = error.error.message;
31                 }
32             );
33         });
34     });
35 }
36 }
```

Ilustración 64. email-verification TS

El componente ts implementa la interfaz OnInit y tiene propiedades y métodos para manejar la verificación de correo electrónico.

Propiedades:

loading: Indica si la página se está cargando.

success: Indica si la verificación de correo electrónico fue exitosa.

error: Indica si hubo un error en la verificación de correo electrónico.

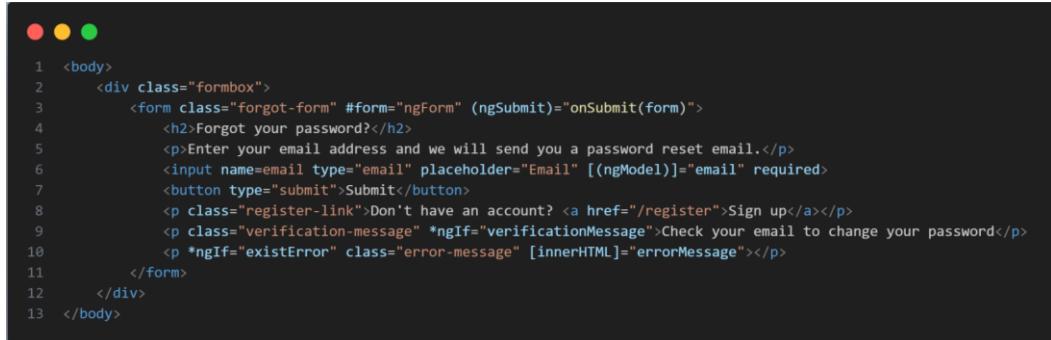
successMessage: Mensaje de éxito en caso de verificación exitosa.

errorMessage: Mensaje de error en caso de error en la verificación.

Constructor: Recibe instancias de ActivatedRoute y AuthService.

Método ngOnInit: Se ejecuta al inicializar el componente. Realiza la verificación de correo electrónico mediante una solicitud al backend y actualiza las propiedades según el resultado.

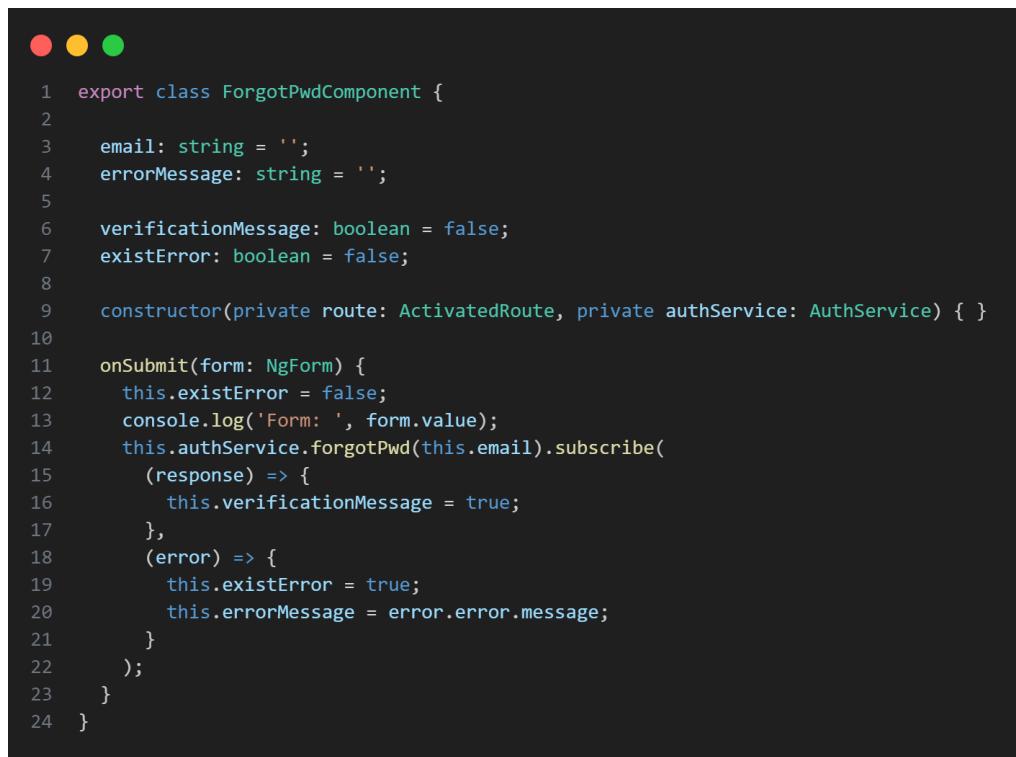
ForgotPwdComponent: Componente para restablecer la contraseña olvidada.



```
1 <body>
2   <div class="formbox">
3     <form class="forgot-form" #form="ngForm" (ngSubmit)="onSubmit(form)">
4       <h2>Forgot your password?</h2>
5       <p>Enter your email address and we will send you a password reset email.</p>
6       <input name=email type="email" placeholder="Email" [(ngModel)]="email" required>
7       <button type="submit">Submit</button>
8       <p class="register-link">Don't have an account? <a href="/register">Sign up</a></p>
9       <p class="verification-message" *ngIf="verificationMessage">Check your email to change your password</p>
10      <p *ngIf="existError" class="error-message" [innerHTML]="errorMessage"></p>
11    </form>
12  </div>
13 </body>
```

Ilustración 65. forgotPwd HTML

El componente HTML muestra un formulario de "Forgot Password" en una página web donde el usuario puede ingresar su dirección de correo electrónico. Al enviar el formulario, se muestra un mensaje de verificación y un mensaje de error si ocurre algún problema. También se proporciona un enlace para registrarse en caso de no tener una cuenta.



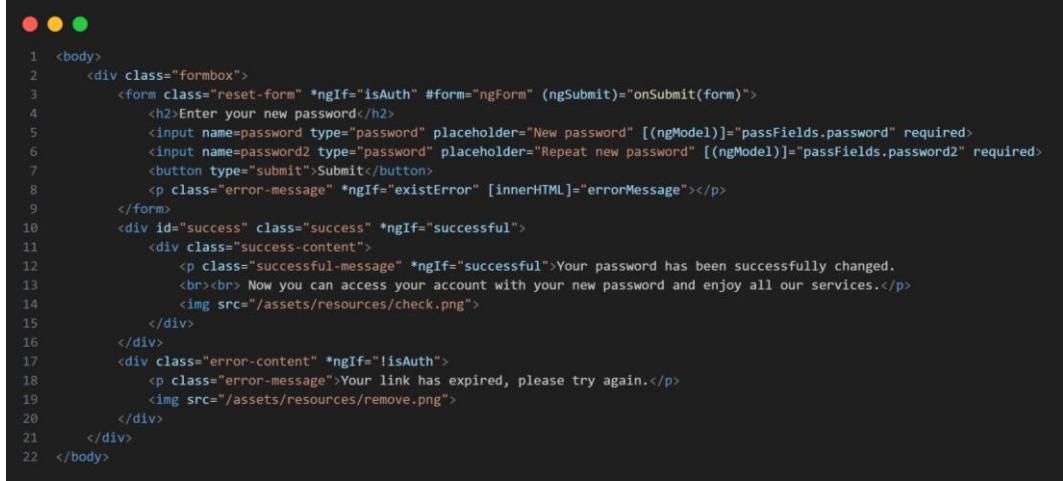
```
1 export class ForgotPwdComponent {
2
3   email: string = '';
4   errorMessage: string = '';
5
6   verificationMessage: boolean = false;
7   existError: boolean = false;
8
9   constructor(private route: ActivatedRoute, private authService: AuthService) { }
10
11  onSubmit(form: NgForm) {
12    this.existError = false;
13    console.log('Form: ', form.value);
14    this.authService.forgotPwd(this.email).subscribe(
15      (response) => {
16        this.verificationMessage = true;
17      },
18      (error) => {
19        this.existError = true;
20        this.errorMessage = error.error.message;
21      }
22    );
23  }
24}
```

Ilustración 66. forgotPwd TS

El componente "ts" se encarga de manejar la funcionalidad del formulario de "Forgot Password". Tiene propiedades para almacenar el correo electrónico ingresado por el usuario y los mensajes de error y verificación. Al enviar el formulario, se realiza una solicitud al servicio "AuthService" para restablecer la contraseña utilizando el correo electrónico proporcionado. Si hay éxito, se

muestra un mensaje de verificación, y si hay un error, se muestra un mensaje de error correspondiente.

ResetpasswordComponent: Componente para cambiar la contraseña.



```
1 <body>
2   <div class="formbox">
3     <form class="reset-form" *ngIf="isAuth" #form="ngForm" (ngSubmit)="onSubmit(form)">
4       <h2>Enter your new password</h2>
5       <input name=password type="password" placeholder="New password" [(ngModel)]="passFields.password" required>
6       <input name=password2 type="password" placeholder="Repeat new password" [(ngModel)]="passFields.password2" required>
7       <button type="submit">Submit</button>
8       <p class="error-message" *ngIf="existError" [innerHTML]="errorMessage"></p>
9     </form>
10    <div id="success" class="success" *ngIf="successful">
11      <div class="success-content">
12        <p class="successful-message" *ngIf="successful">Your password has been successfully changed.</p>
13        <br><br> Now you can access your account with your new password and enjoy all our services.</p>
14        
15      </div>
16    </div>
17    <div class="error-content" *ngIf="!isAuth">
18      <p class="error-message">Your link has expired, please try again.</p>
19      
20    </div>
21  </div>
22 </body>
```

Ilustración 67. resetPwd HTML

Si el usuario está autenticado (“isAuth” es verdadero), se muestra el formulario para ingresar la nueva contraseña. Los campos de contraseña son vinculados a propiedades “passFields.password” y passFields.password2 mediante “ngModel”. Al enviar el formulario, se realiza una acción “onSubmit” que procesa los datos. Si hay un error, se muestra un mensaje de error correspondiente.

Si el restablecimiento de contraseña es exitoso (“successful” es verdadero), se muestra un mensaje de éxito y una imagen de marca de verificación. Si el usuario no está autenticado, se muestra un mensaje de error y una imagen de marca de eliminación, indicando que el enlace ha expirado.

```

1  export class ResetpasswordComponent {
2
3    passFields: Passwords = {
4      password: '',
5      password2: ''
6    }
7
8    request: RequestChangePass = {
9      username: '',
10     password: ''
11   }
12
13   responseAuth: ResponseAuth = {
14     isAuthenticated: false,
15     username: ''
16   }
17
18   isAuthenticated: boolean = true;
19   successful: boolean = false;
20   existError: boolean = false;
21
22   errorMessage: string = '';
23
24   pwdError: string[] = [
25     'Passwords do not match',
26     'Passwords must be at least <br> 8 characters long.'
27 ];
28
29   token = '';
30
31   constructor(private route: ActivatedRoute, private authService: AuthService) { }
32
33   ngOnInit(): void {
34
35     // Obtiene el token
36     this.route.queryParams.subscribe(params => {
37       this.token = params['token'];
38       console.log(this.token);
39       // Autenticacion del token
40       this.authService.authToken(this.token).subscribe(
41         (responseAuth) => {
42           console.log(responseAuth);
43           this.isAuthenticated = responseAuth.isAuthenticated;
44         },
45         (error) => {
46           this.isAuthenticated = false;
47           console.log(error.error.message);
48         }
49       );
50     });
51   }
52
53
54   onSubmit(form: NgForm) {
55     this.existError = false;
56     console.log('Form: ', form.value);
57     if (this.passFields.password === this.passFields.password2) {
58       if (this.passFields.password.length >= 8) {
59         let decodedToken = jwtHelper.decodeToken(this.token);
60         console.log(decodedToken);
61         this.request.username = decodedToken.sub;
62         this.request.password = this.passFields.password;
63         this.authService.changePwd(this.request)
64           .subscribe(response => {
65             this.successful = true;
66           },
67             (error) => {
68               this.existError = true;
69               this.errorMessage = error.error.message;
70             }
71           );
72     } else {
73       this.errorMessage = this.pwdError[1];
74       this.existError = true;
75     }
76   } else {
77     this.errorMessage = this.pwdError[0];
78     this.existError = true;
79   }
80 }
81
82
83 }

```

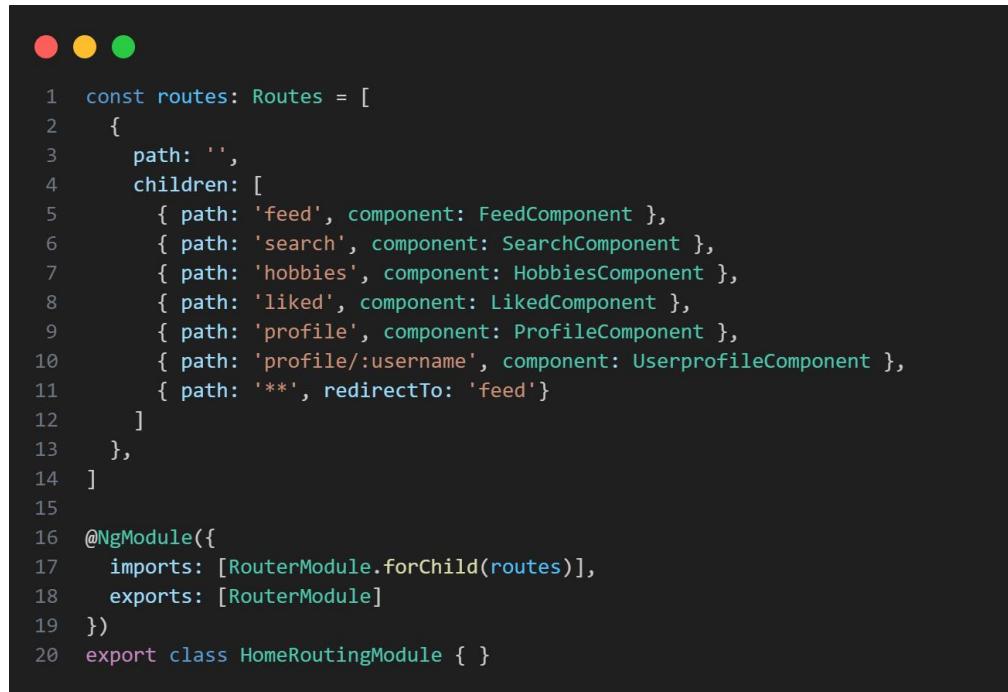
Ilustración 68. resetPwd TS

Este componente “ts” corresponde a un componente de restablecimiento de contraseña. Se define un formulario para ingresar una nueva contraseña. Al enviar el formulario, se realizan varias validaciones. Si las contraseñas ingresadas coinciden y tienen al menos 8 caracteres de longitud, se realiza una solicitud al Backend para cambiar la contraseña. Si el cambio de contraseña es exitoso, se muestra un mensaje de éxito. En caso de errores, se muestra un mensaje de error correspondiente. El componente también verifica la autenticidad de un token recibido en los parámetros de la URL y realiza una autenticación correspondiente.

5.2.2.2 Módulo “home.module”

Este módulo representa la página principal de la aplicación.

Además, se define un archivo de enrutamiento de los componentes pertenecientes a este módulo:



```
1 const routes: Routes = [
2   {
3     path: '',
4     children: [
5       { path: 'feed', component: FeedComponent },
6       { path: 'search', component: SearchComponent },
7       { path: 'hobbies', component: HobbiesComponent },
8       { path: 'liked', component: LikedComponent },
9       { path: 'profile', component: ProfileComponent },
10      { path: 'profile/:username', component: UserProfileComponent },
11      { path: '**', redirectTo: 'feed' }
12    ]
13  },
14]
15
16 @NgModule({
17   imports: [RouterModule.forChild(routes)],
18   exports: [RouterModule]
19 })
20 export class HomeRoutingModule { }
```

Ilustración 69. home-routing

Archivo de rutas del módulo “home.module”.

A continuación, se expondrán los siguientes componentes:

FeedComponent: Componente principal de la página de inicio.

```
1  <body>
2    <div class="row">
3      <div class="col-sm-2">
4        <app-navbar />
5      </div>
6
7      <div class="col-sm-10">
8        <h1 class="d-flex align-items-center justify-content-center">
9          SocialHub
10         </h1>
11        <!-- Feed content -->
12        <div class="posts" *ngIf="posts.length > 0">
13          <div class="d-flex align-items-center justify-content-center" *ngFor="let post of posts">
14            <div class="image-box">
15              <div class="image-container">
16                <img [src]="getBase64Image(post.postContent)" alt="Imagen" />
17              </div>
18              <div class="header-container">
19                <h3 class="header"><b>{this.post.username + " : "}</b>{ post.text }</h3>
20                <img [src]="post.likeImage ? '/assets/resources/fire-real.png' : '/assets/resources/fire.png'" alt="Like" class="like-button" (click)="likeImage(post)" />
21                <span class="like-count">{ post.nlikes }</span>
22              </div>
23            </div>
24          </div>
25        </div>
26        <div class="d-flex align-items-center justify-content-center">
27          <h2 class="no-posts" *ngIf="noPosts" [innerHTML]="noPostsMessage"></h2>
28        </div>
29      </div>
30    </div>
31
32  </div>
33 </body>
```

Ilustración 70. feed HTML

Este código representa la estructura de una página web que muestra un *feed* de publicaciones. El componente consta de un encabezado con un logotipo y una barra de navegación, seguido del contenido del *feed*. Las publicaciones se muestran en una cuadrícula, donde cada publicación incluye una imagen, un encabezado con el nombre de usuario y el texto de la publicación, y un botón de "Me gusta" con la cantidad de *likes*. Además, hay un mensaje opcional que se muestra cuando no hay publicaciones disponibles.

```

1  ngOnInit() {
2      this.getPosts();
3  }
4
5  getPosts() {
6      this.homeService.getPosts().subscribe(
7          (response: Post[]) => {
8              console.log(response);
9              if(response.length == 0){
10                  this.noPosts=true;
11              }else{
12                  this.posts = response;
13              }
14          },
15          (error: HttpErrorResponse) => {
16              if(error.status === 0){
17                  this.existError = true;
18                  this.errorMessage = 'YOU MUST BE LOGGED IN TO USE SOCIALHUB';
19              }
20              console.error('Error getting publications', error);
21              console.error(error.headers);
22          }
23      );
24  }
25
26  likeImage(post: any){
27      post.likeImage = !post.likeImage;
28      this.homeService.like(post).subscribe(
29          (response: any) => {
30              if(post.likeImage){
31                  post.nlikes = post.nlikes + 1;
32              }else{
33                  post.nlikes = post.nlikes - 1;
34              }
35              console.log(response);
36          },
37          (error) => {
38              this.existError = true;
39              this.errorMessage = error.error.message;
40              console.error(error);
41          }
42      );
43  }
44
45  getBase64Image(imageData: any){
46      if (!imageData) {
47          return '';
48      }
49
50      let base64Pic:string = 'data:image/png;base64,' + imageData;
51      return base64Pic;
52  }

```

Ilustración 71. feed TS

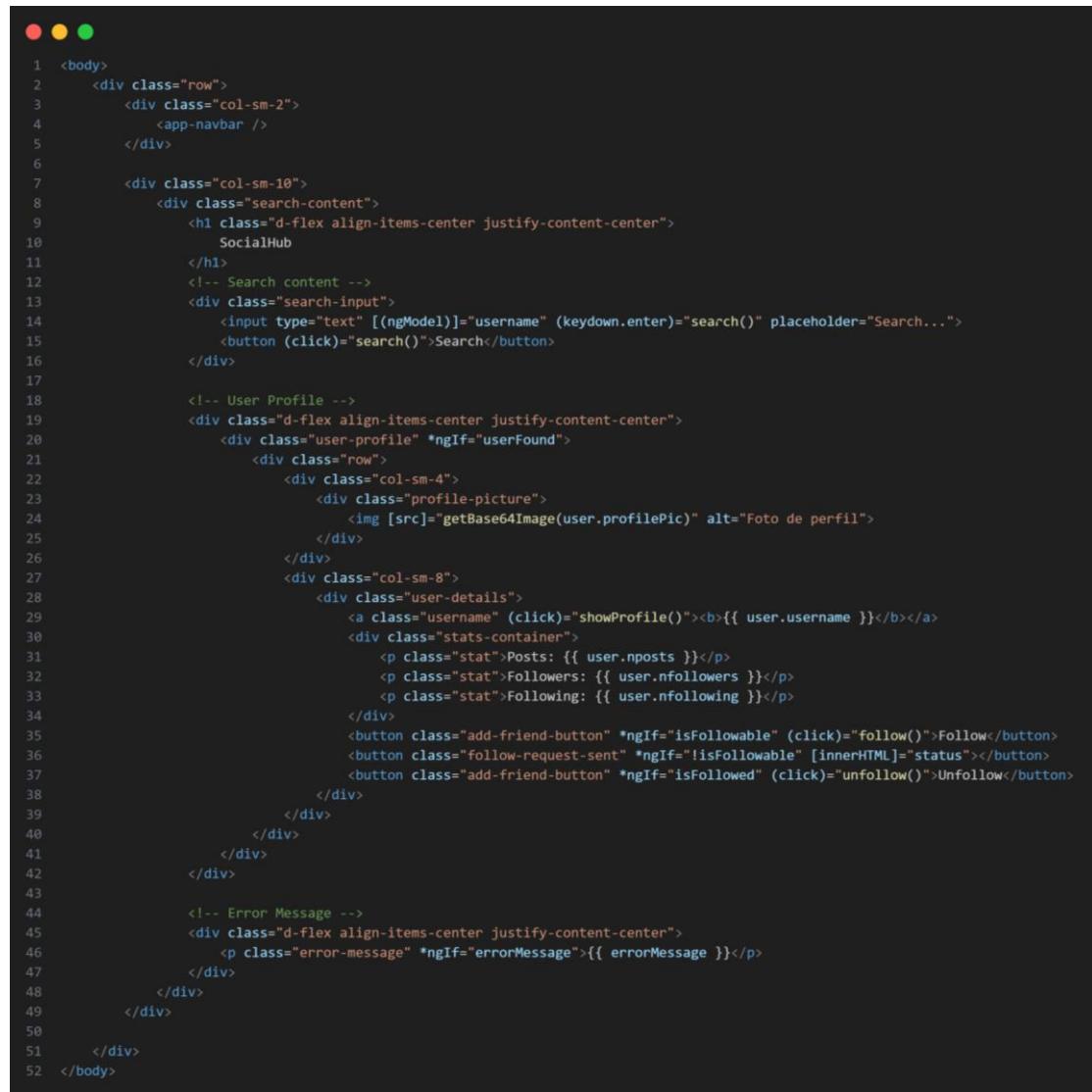
`ngOnInit()`: Este método se llama cuando el componente se inicializa. Llama al método “`getPosts()`” para obtener las publicaciones.

`getPosts()`: Este método llama al servicio “`homeService`” para obtener las publicaciones. Si se obtiene una respuesta vacía, se establece la variable `noPosts` en `true`; de lo contrario, se asigna la respuesta a la variable `posts`.

`likeImage(post)`: Este método cambia el estado de “`likeImage`” en la publicación y llama al servicio “`homeService`” para realizar la acción de “me gusta”. Actualiza el contador de `likes` según la respuesta.

`getBase64Image(“imageData”)`: Este método toma los datos de la imagen y genera una cadena de base64 para mostrar la imagen en la interfaz de usuario.

SearchComponent: Componente para mostrar la lista de publicaciones.



```
1 <body>
2   <div class="row">
3     <div class="col-sm-2">
4       <app-navbar />
5     </div>
6
7     <div class="col-sm-10">
8       <div class="search-content">
9         <h1 class="d-flex align-items-center justify-content-center">
10           SocialHub
11         </h1>
12         <!-- Search content -->
13         <div class="search-input">
14           <input type="text" [(ngModel)]="username" (keydown.enter)="search()" placeholder="Search...">
15           <button (click)="search()>Search</button>
16         </div>
17
18         <!-- User Profile -->
19         <div class="d-flex align-items-center justify-content-center">
20           <div class="user-profile" *ngIf="userFound">
21             <div class="row">
22               <div class="col-sm-4">
23                 <div class="profile-picture">
24                   <img [src]=>getBase64Image(user.profilePic) alt="Foto de perfil">
25                 </div>
26               </div>
27               <div class="col-sm-8">
28                 <div class="user-details">
29                   <a class="username" (click)="showProfile()"><b>{{ user.username }}</b></a>
30                   <div class="stats-container">
31                     <p class="stat">Posts: {{ user.nposts }}</p>
32                     <p class="stat">Followers: {{ user.nfollowers }}</p>
33                     <p class="stat">Following: {{ user.nfollowing }}</p>
34                   </div>
35                   <button class="add-friend-button" *ngIf="isFollowable" (click)="follow()">Follow</button>
36                   <button class="follow-request-sent" *ngIf="!isFollowable" [innerHTML]="">status</button>
37                   <button class="add-friend-button" *ngIf="isFollowed" (click)="unfollow()">Unfollow</button>
38                 </div>
39               </div>
40             </div>
41           </div>
42         </div>
43
44         <!-- Error Message -->
45         <div class="d-flex align-items-center justify-content-center">
46           <p class="error-message" *ngIf="errorMessage">{{ errorMessage }}</p>
47         </div>
48       </div>
49     </div>
50
51   </div>
52 </body>
```

Ilustración 72. search HTML

Hay un componente de navegación (`app-navbar`) en la columna izquierda.

En la columna derecha, hay un campo de búsqueda de usuarios y un botón para buscar.

Si se encuentra un usuario, se muestra su perfil, que incluye su foto de perfil, detalles de usuario (nombre de usuario, estadísticas de publicaciones, seguidores y seguidos) y botones para seguir/ dejar de seguir.

Si se produce un error, se muestra un mensaje de error.

```
1  search(): void {
2      this.userFound = false;
3      console.log(this.username);
4      this.homeService.searchUser(this.username).subscribe(
5          (user: any) => {
6              this.errorMessage = '';
7              this.userFound = true;
8              this.isFollowable = this.followable(user.requestStatus);
9              this.isFollowed = user.requestStatus === FriendRequestStatus.FOLLOWED ? true : false;
10             console.log(user);
11             this.status = user.requestStatus;
12             console.log(this.status);
13             this.user = user;
14         },
15         (error: any) => {
16             console.log(error.status);
17             if(error.status == 404){
18                 this.errorMessage = 'Username: ' + this.username + ' not found';
19             }
20         }
21     );
22 }
23
24
25 follow(): void {
26     this.homeService.followRequest(this.username).subscribe(
27         (newStatus: any) => {
28             //this.isPrivate = isPriv;
29         },
30         (error: any) => {
31             console.log(error);
32         }
33     );
34 }
35
36 unfollow(): void {
37     this.homeService.unfollow(this.username).subscribe(
38         (newStatus: any) => {
39             //this.isPrivate = isPriv;
40             this.isFollowable = true;
41             this.isFollowed = false;
42         },
43         (error: any) => {
44             console.log(error);
45         }
46     );
47 }
48
49 private followable(status:string): boolean{
50     if(status === FriendRequestStatus.FOLLOWABLE){
51         return true;
52     }
53     return false;
54 }
55
56 getBase64Image(imageData: any){
57     if (!imageData) {
58         return '';
59     }
60
61     let base64Pic:string = 'data:image/png;base64,' + imageData;
62     return base64Pic;
63 }
64
65 showProfile(){
66     console.log(this.user.username);
67     this.router.navigate(['/home/profile', this.user.username]);
68     //this.router.navigateByUrl('/profile/' + this.user.username);
69 }
```

Ilustración 73. search TS

La función “search()” realiza una búsqueda de usuario utilizando el nombre de usuario ingresado en el campo de búsqueda. Si se encuentra un usuario, se actualizan las variables y propiedades correspondientes para mostrar el perfil del usuario y determinar si es posible seguirlo o si ya se está siguiendo.

La función “follow()” envía una solicitud de seguimiento al usuario.

La función “unfollow()” cancela el seguimiento a un usuario.

La función “followable()” verifica si el estado de la solicitud de amistad permite seguir al usuario.

La función “getBase64Image()” devuelve una imagen codificada en base64.

La función “showProfile()” redirige a la página de perfil del usuario seleccionado.

LikedComponent: Componente para mostrar una publicación individual.

```
1  <body>
2    <div class="row">
3      <div class="col-sm-2">
4        <app-navbar />
5      </div>
6
7      <div class="col-sm-10">
8        <h1 class="d-flex align-items-center justify-content-center">
9          SocialHub
10         </h1>
11        <div class="posts" *ngIf="posts.length > 0">
12          <div class="d-flex align-items-center justify-content-center" *ngFor="let post of posts">
13            <div class="image-box">
14              <div class="image-container">
15                <img [src]="getBase64Image(post.postContent)" alt="Imagen">
16              </div>
17              <div class="header-container">
18                <h3 class="header"><b>{{this.post.username + " : "}}</b>{{ post.text }}</h3>
19                <img [src]="post.likeImage ? '/assets/resources/fire-real.png' : '/assets/resources/fire.png'" alt="Like" class="like-button" (click)="likeImage(post)">
20                <span class="like-count">{{ post.likes }}</span>
21              </div>
22            </div>
23          </div>
24        </div>
25      </div>
26    </div>
27
28  </div>
29 </body>
```

Ilustración 74. like HTML

La página consta de una barra de navegación representada por <app-navbar>.

El título de la página se muestra como "SocialHub" utilizando el elemento <h1>.

Las publicaciones se muestran dentro de un contenedor <div class="posts">.

Para cada publicación, se muestra una imagen representada por y el texto de la publicación se muestra dentro de un elemento <h3>.

Cada publicación también incluye un botón de "Like" representado por y una cuenta de "likes" representada por .

Se utiliza un bucle *ngFor para iterar sobre las publicaciones y mostrarlas en la página.

Las imágenes y el conteo de "likes" se actualizan dinámicamente según la lógica definida en la función likeImage(post).

```
● ○ ●
1  ngOnInit() {
2      this.getPosts();
3  }
4
5  likeImage(post: any){
6      post.likeImage = !post.likeImage;
7      this.homeService.like(post).subscribe(
8          (response: any) => {
9              if(post.likeImage){
10                  post.nlikes = post.nlikes + 1;
11              }else{
12                  post.nlikes = post.nlikes - 1;
13              }
14              console.log(response);
15          },
16          (error) => {
17              this.existError = true;
18              this.errorMessage = error.error.message;
19              console.error(error);
20          }
21      );
22  }
23
24  getPosts() {
25      this.homeService.getLiked().subscribe(
26          (response: Post[]) => {
27              console.log(response);
28              if(response.length == 0){
29                  this.noPosts=true;
30              }else{
31                  this.posts = response;
32              }
33          },
34          (error: HttpErrorResponse) => {
35              if(error.status === 0){
36                  this.existError = true;
37                  this.errorMessage = 'YOU MUST BE LOGGED IN TO USE SOCIALHUB';
38              }
39              console.error('Error getting publications', error);
40              console.error(error.headers);
41          }
42      );
43  }
44
45  getBase64Image(imageData: any){
46      if (!imageData) {
47          return '';
48      }
49
50      let base64Pic:string = 'data:image/png;base64,' + imageData;
51      return base64Pic;
52  }
```

Ilustración 75. like TS

La función “ngOnInit” se llama al inicializar el componente y llama a la función “getPosts” para obtener las publicaciones.

La función “likeImage” se utiliza para cambiar el estado del "Like" de una publicación y realizar una solicitud al servicio “homeService” para actualizar el estado del "Like" en el servidor. También actualiza el conteo de "likes" en la publicación.

La función getPosts se utiliza para obtener las publicaciones que han sido marcadas como "Liked" por el usuario a través del servicio “homeService”.

La función “getBase64Image” se utiliza para convertir los datos de imagen en formato base64 y devolver la URL de la imagen.

ProfileComponent: Componente para mostrar el perfil del usuario.

```

1 <body>
2   <div class="row">
3     <div class="col-sm-2">
4       <app-navbar />
5     </div>
6
7     <div class="col-sm-10">
8       <h1 class="d-flex align-items-center justify-content-center">
9         SocialHub
10        </h1>
11       <div class="d-flex justify-content-center">
12         <div class="profile-container">
13           <div class="profile">
14             <div class="profile-details">
15               <div class="profile-picture">
16                 <img [src]="getBase64Image(profile.profilePic)" alt="Foto de perfil">
17               </div>
18               <div class="profile-info">
19                 <div class="username">{{ profile.username }}</div>
20                 <div class="fullname">{{ profile.fullname }}</div>
21                 <div class="edit-profile" *ngIf="!isEditMode">
22                   <button (click)="editProfile()">Edit profile</button>
23                 </div>
24                 <div class="edit-profile-form" *ngIf="isEditMode">
25                   <input [(ngModel)]="editRequest.username" class="form-control" placeholder="new username">
26                   <input [(ngModel)]="editRequest.fullname" class="form-control" placeholder="new fullname">
27                   <button class="change-privacy" (click)="changePrivacy()"></button>
28                   <div class="buttons">
29                     <button class="btn btn-primary" (click)="saveProfile()">Save</button>
30                     <button class="btn btn-secondary" (click)="cancelEditProfile()">Cancelar</button>
31                   </div>
32                 </div>
33               </div>
34             </div>
35             <br> <!-- \n -->
36             <div class="change-password">
37               <button (click)="changePassword()">Change Password</button>
38             </div>
39             <p class="error-message" *ngIf="existError" [innerHTML]={{ errorMessage }}></p>
40           </div>
41         </div>
42         <div class="profile-stats">
43           <div class="stats">
44             <div class="stat">
45               <div class="value">{{ profile.nfollowers }}</div>
46               <div class="label">Followers</div>
47             </div>
48             <div class="stat">
49               <div class="value">{{ profile.nfollowing }}</div>
50               <div class="label">Following</div>
51             </div>
52             <div class="stat">
53               <div class="value">{{ profile.nposts }}</div>
54               <div class="label">Posts</div>
55             </div>
56           </div>
57         </div>
58       </div>
59     <div class="posts" *ngIf="profile.posts.length > 0">
60       <div class="d-flex align-items-center justify-content-center" *ngFor="let post of profile.posts">
61         <div class="image-box">
62           <div class="image-container">
63             <img [src]="getBase64Image(post.postContent)" alt="Imagen">
64           </div>
65           <div class="header-container">
66             <h3 class="header"><b>{{ this.profile.username + " : " }}</b>{{ post.text }}</h3>
67             <img [src]="post.likeImage ? '/assets/resources/fire-real.png' : '/assets/resources/fire.png'" alt="Like" class="like-button" (click)=>likeImage(post)>
68             <span class="like-count">{{ post.nlikes }}</span>
69           </div>
70         </div>
71       </div>
72     </div>
73   </div>
74   <div class="no-posts" *ngIf="profile.posts.length === 0">
75     No publications
76   </div>
77   <div class="follow-requests" *ngIf="profile.followRequests.length > 0">
78     <div class="request" *ngFor="let request of profile.followRequests">
79       <div class="username">{{ request }}</div>
80       <div class="actions">
81         <button class="btn btn-primary" (click)=>acceptFollowRequest(request)>Aceptar</button>
82         <button class="btn btn-danger" (click)=>rejectFollowRequest(request)>Rechazar</button>
83       </div>
84     </div>
85   </div>
86 </div>
87 </div>
88 </div>
89 </div>
90 </div>
91 </div>
92 </body>
93 
```

Ilustración 76. profile HTML

El componente “app-navbar” se muestra en una columna de ancho 2 para la navegación.

El título de la página se muestra centrado en la parte superior.

La información de perfil se muestra en el contenedor principal.

La imagen de perfil se muestra utilizando los datos de imagen en formato base64.

Se muestra el nombre de usuario y el nombre completo del perfil.

Si no está en modo de edición, se muestra un botón para editar el perfil. Al hacer clic en el botón, se activa el modo de edición y se muestran campos para editar el nombre de usuario y el nombre completo.

Hay un botón para cambiar la contraseña.

Se muestra el conteo de seguidores, seguidos y publicaciones del perfil.

Las publicaciones del perfil se muestran en una sección de publicaciones. Cada publicación muestra una imagen, el texto de la publicación y un botón de "Like" que cambia entre diferentes imágenes dependiendo del estado de "Like" de la publicación.

Si no hay publicaciones en el perfil, se muestra un mensaje indicando que no hay publicaciones.

Si hay solicitudes de seguimiento pendientes en el perfil, se muestra una sección con las solicitudes, mostrando el nombre de usuario y los botones para aceptar o rechazar cada solicitud.

```
1  ngOnInit() {
2    this.getProfile();
3  }
4
5  getProfile(){
6    this.homeService.getProfile('').subscribe(
7      (p: any) => {
8        this.profile = p;
9        console.log(this.profile);
10       },
11       (error: any) => {
12         console.log(error);
13         if(error.status == 404){
14           this.errorMessage = 'Username: ' + this.username + ' not found';
15         }
16       }
17     );
18   }
19
20  editProfile() {
21    this.isEditMode = true;
22  }
23
24  saveProfile() {
25    this.existError = false;
26    console.log(this.editRequest)
27    this.isEditMode = false;
28    this.homeService.editUser(this.editRequest).subscribe(
29      (response: any) => {
30        console.log(response);
31      },
32      (error) => {
33        this.existError = true;
34        this.editRequest.fullname = '';
35        this.editRequest.username = '';
36        this.errorMessage = error.error.message;
37        console.error(error);
38      }
39    );
40  }
41
42  cancelEditProfile() {
43    this.isEditMode = false;
44    this.editRequest.fullname = '';
45    this.editRequest.username = '';
46  }
47
48  changePassword() {
49    let token = localStorage.getItem('token')
50    this.router.navigateByUrl(`'/reset-pwd?token=${token}`)
51  }
52
53  acceptFollowRequest(request: string) {
54
55  }
56
57  rejectFollowRequest(request: string) {
58
59  }
60
61  likeImage(post: any){
62    post.likeImage = !post.likeImage;
63    this.homeService.like(post).subscribe(
64      (response: any) => {
65        if(post.likeImage){
66          post.nlikes = post.nlikes + 1;
67        }else{
68          post.nlikes = post.nlikes - 1;
69        }
70        console.log(response);
71      },
72      (error) => {
73        this.existError = true;
74        this.errorMessage = error.error.message;
75        console.error(error);
76      }
77    );
78  }
79
80  changePrivacy(){
81
82  }
83
84  getBase64Image(imageData: any){
85    if (!imageData) {
86      return '';
87    }
88
89    let base64Pic:string = 'data:image/png;base64,' + imageData;
90    return base64Pic;
91  }
92
93 }
```

Ilustración 77. profile TS

En el método “ngOnInit()”, se llama al método “getProfile()” para obtener el perfil del usuario actual.

El método “getProfile()” utiliza el servicio “homeService” para obtener el perfil del usuario. Al recibir la respuesta exitosa, se asigna el perfil a la variable *profile*.

Si ocurre un error durante la obtención del perfil y el error tiene un código de estado 404, se establece un mensaje de error indicando que el nombre de usuario no se encontró.

El método “editProfile()” activa el modo de edición al establecer la variable *isEditMode* en true.

El método “saveProfile()” se utiliza para guardar los cambios realizados en el perfil. Se restablecen las variables *existError*, *editRequest.fullname* y *editRequest.username* antes de enviar la solicitud al servicio “homeService” para editar el perfil. En caso de respuesta exitosa, se muestra la respuesta en la consola.

El método “cancelEditProfile()” cancela el modo de edición y restablece los valores de las variables relacionadas.

El método “changePassword()” redirige a la página de cambio de contraseña, pasando un token de autenticación en la URL.

Los métodos “acceptFollowRequest()” y “rejectFollowRequest()” se utilizan para aceptar y rechazar solicitudes de seguimiento, respectivamente. Sin embargo, no se proporciona ninguna implementación en el código proporcionado.

El método “likeImage(post)” se utiliza para cambiar el estado de "Me gusta" de una publicación. Se invoca el servicio “homeService” para realizar la acción de "Me gusta" en la publicación. Si la acción es de "Me gusta", se incrementa el conteo de "Me gusta" en la publicación; de lo contrario, se decrementa el conteo.

El método “changePrivacy()” está vacío y no contiene ninguna implementación.

El método *getBase64Image(imageData)* se utiliza para convertir datos de imagen en formato base64 en una URL de imagen válida.

UserprofileComponent: Componente para crear una nueva publicación.

Tanto el componente HTML como *TypeScript*, son análogos en funcionamiento a los del componente de “ProfileComponent”. Únicamente se mostrara el código:

```

1 <body>
2   <div class="row">
3     <div class="col-sm-2">
4       <app-navbar />
5     </div>
6
7     <div class="col-sm-10">
8       <div class="userprofile-content">
9         <h1 class="d-flex align-items-center justify-content-center">
10          SocialHub
11        </h1>
12        <div class="d-flex justify-content-center">
13          <div class="profile-container">
14            <div class="profile">
15              <div class="profile-details">
16                <div class="profile-picture">
17                  <img [src]="getBase64Image(profile.profilePic)" alt="Foto de perfil">
18                </div>
19                <div class="profile-info">
20                  <div class="username">{{ profile.username }}</div>
21                  <div class="fullname">{{ profile.fullname }}</div>
22                </div>
23              </div>
24              <div class="profile-stats">
25                <div class="stats">
26                  <div class="stat">
27                    <div class="value">{{ profile.nfollowers }}</div>
28                    <div class="label">Followers</div>
29                  </div>
30                  <div class="stat">
31                    <div class="value">{{ profile.nfollowing }}</div>
32                    <div class="label">Following</div>
33                  </div>
34                  <div class="stat">
35                    <div class="value">{{ profile.nposts }}</div>
36                    <div class="label">Posts</div>
37                  </div>
38                </div>
39              </div>
40            </div>
41            <div class="posts" *ngIf="profile.posts.length > 0">
42              <div class="d-flex align-items-center justify-content-center"
43                *ngFor="let post of profile.posts">
44                <div class="image-box">
45                  <div class="image-container">
46                    <img [src]="getBase64Image(post.postContent)" alt="Imagen">
47                  </div>
48                  <div class="header-container">
49                    <h3 class="header"><b>{{this.profile.username + ": "}}</b>{{ post.text }}</h3>
50                    <img [src]="post.likeImage ? '/assets/resources/fire-real.png' : '/assets/resources/fire.png'"
51                      alt="Like" class="like-button" (click)="likeImage(post)">
52                    <span class="like-count">{{ post.nlikes }}</span>
53                  </div>
54                </div>
55              </div>
56            </div>
57          </div>
58        </div>
59      </div>
60    </div>
61
62  </div>
63 </body>

```

Ilustración 78. userprofile HTML

```
● ○ ● ●  
1  ngOnInit() {  
2      this.username = this.route.snapshot.paramMap.get('username');  
3      console.log(this.username);  
4      this.getProfile();  
5  }  
6  
7  getProfile(){  
8      let user: string = '' + this.username;  
9      this.homeService.getProfile(user).subscribe(  
10         (p: any) => {  
11             this.profile = p;  
12             console.log(this.profile);  
13         },  
14         (error: any) => {  
15             console.log(error);  
16             if(error.status == 404){  
17                 this.errorMessage = 'Username: ' + this.username + ' not found';  
18             }  
19         }  
20     );  
21  }  
22  
23  likeImage(post: any){  
24      post.likeImage = !post.likeImage;  
25      this.homeService.like(post).subscribe(  
26         (response: any) => {  
27             if(post.likeImage){  
28                 post.nlikes = post.nlikes + 1;  
29             }else{  
30                 post.nlikes = post.nlikes - 1;  
31             }  
32             console.log(response);  
33         },  
34         (error) => {  
35             this.existError = true;  
36             this.errorMessage = error.error.message;  
37             console.error(error);  
38         }  
39     );  
40  }  
41  
42  getBase64Image(imageData: any){  
43      if (!imageData) {  
44          return '';  
45      }  
46  
47      let base64Pic:string = 'data:image/png;base64,' + imageData;  
48      return base64Pic;  
49  }
```

Ilustración 79. *userprofile TS*

5.2.2.3 Módulo “shared.module”

Módulo "shared.module": Este módulo contiene componentes, servicios y otros elementos compartidos en toda la aplicación. Algunos de los componentes incluidos son:

NavbarComponent: Componente de la barra de navegación.

```
1 <body>
2     <!-- Sidebar -->
3     <div class="sidebar">
4         <ul>
5             <li>
6                 <a href="/home/feed">
7                     
8                     <span>Home</span>
9                 </a>
10            </li>
11            <li>
12                <a href="/home/search">
13                    
14                    <span>Search</span>
15                </a>
16            </li>
17            <li>
18                <a href="/home/hobbies">
19                    
20                    <span>Hobbies</span>
21                </a>
22            </li>
23            <li>
24                <a href="/home/liked">
25                    
26                    <span>Liked</span>
27                </a>
28            </li>
29            <li>
30                <a href="/home/profile">
31                    
32                    <span>Profile</span>
33                </a>
34            </li>
35            <li>
36                <a (click)="openCreatePost()">
37                    
38                    <span>Post Content</span>
39                </a>
40            </li>
41            <br><br>
42            <li>
43                <a (click)="logOut()">
44                    
45                    <span>Log out</span>
46                </a>
47            </li>
48        </ul>
49    </div>
50
51     <!-- Post content -->
52     <div id="upload" class="upload" *ngIf="isCreatePostOpen">
53         <div class="upload-content">
54             <span class="close" (click)="closeCreatePost()">&times;
```

Ilustración 80. navbar HTML

La barra lateral contiene una lista de elementos con enlaces a diferentes secciones de la página. Cada elemento consta de una imagen de ícono y un texto descriptivo. Los enlaces son para las siguientes secciones: "Home", "Search", "Hobbies", "Liked" y "Profile". También hay un elemento adicional para crear una nueva publicación.

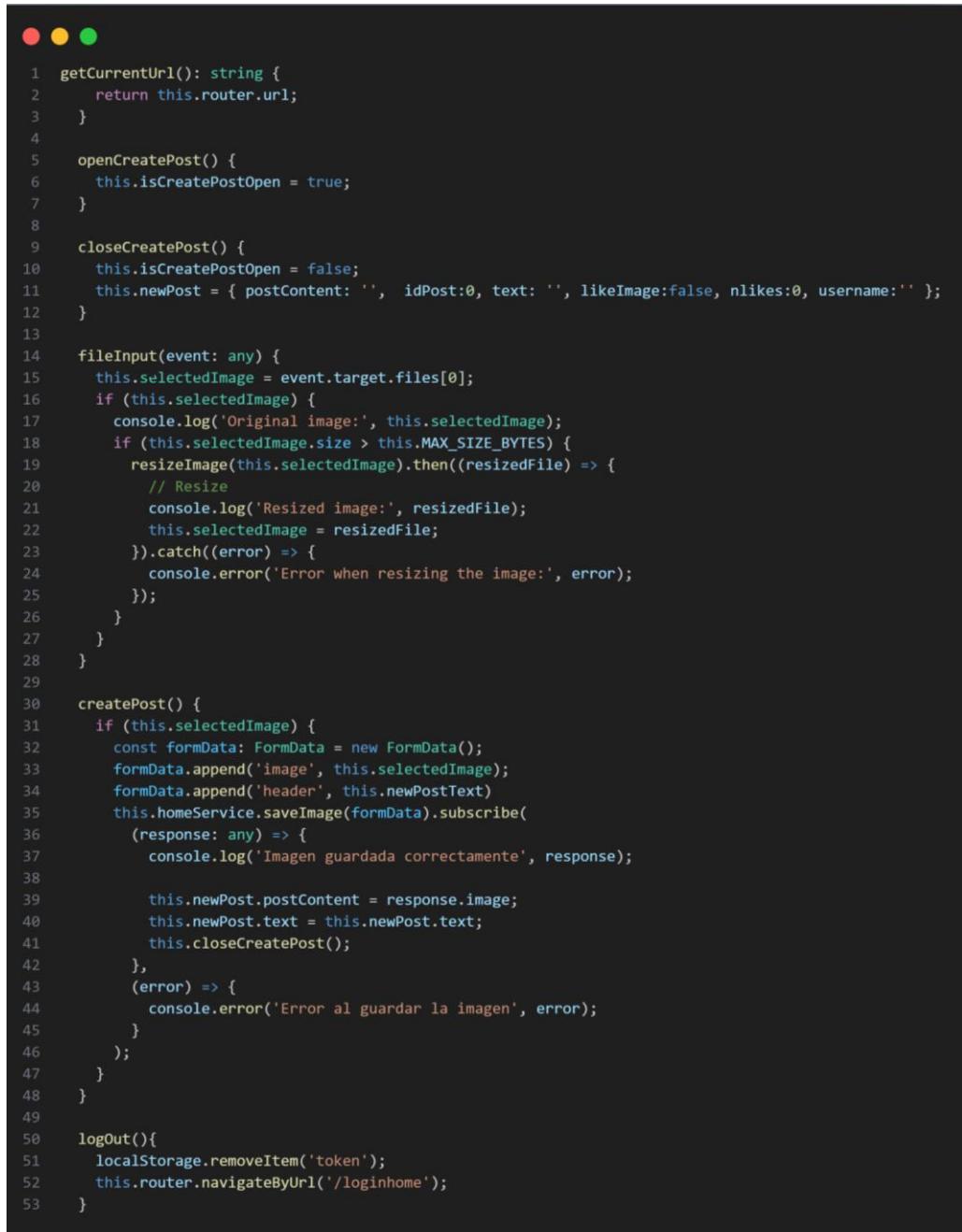
Al hacer clic en el enlace de "Post Content", se activa el método "openCreatePost()" para abrir el formulario de carga de contenido de publicación.

El formulario de carga de contenido de publicación se muestra si la variable "isCreatePostOpen" es true. Contiene un título, una descripción y un campo de entrada de texto para el encabezado de la publicación. También hay un campo para seleccionar una imagen y dos botones: uno para seleccionar una imagen y otro para cargar la imagen y crear la publicación.

Al hacer clic en la "X" en la esquina superior derecha del formulario de carga de contenido de publicación, se activa el método "closeCreatePost()" para cerrar el formulario.

El método "fileInput(\$event)" se activa cuando se selecciona un archivo de imagen y actualiza la variable *imageFile* con la imagen seleccionada.

El método "createPost()" se activa al hacer clic en el botón "Upload image" y crea una nueva publicación utilizando el encabezado de la publicación ingresado y la imagen seleccionada.



```

1  getCurrentUrl(): string {
2      return this.router.url;
3  }
4
5  openCreatePost() {
6      this.isCreatePostOpen = true;
7  }
8
9  closeCreatePost() {
10     this.isCreatePostOpen = false;
11     this.newPost = { postContent: '', idPost:0, text: '', likeImage:false, nLikes:0, username:'' };
12 }
13
14 fileInput(event: any) {
15     this.selectedImage = event.target.files[0];
16     if (this.selectedImage) {
17         console.log('Original image:', this.selectedImage);
18         if (this.selectedImage.size > this.MAX_SIZE_BYTES) {
19             resizeImage(this.selectedImage).then((resizedFile) => {
20                 // Resize
21                 console.log('Resized image:', resizedFile);
22                 this.selectedImage = resizedFile;
23             }).catch((error) => {
24                 console.error('Error when resizing the image:', error);
25             });
26         }
27     }
28 }
29
30 createPost() {
31     if (this.selectedImage) {
32         const formData: FormData = new FormData();
33         formData.append('image', this.selectedImage);
34         formData.append('header', this.newPostText)
35         this.homeService.saveImage(formData).subscribe(
36             (response: any) => {
37                 console.log('Imagen guardada correctamente', response);
38
39                 this.newPost.postContent = response.image;
40                 this.newPost.text = this.newPost.text;
41                 this.closeCreatePost();
42             },
43             (error) => {
44                 console.error('Error al guardar la imagen', error);
45             }
46         );
47     }
48 }
49
50 logOut(){
51     localStorage.removeItem('token');
52     this.router.navigateByUrl('/loginhome');
53 }

```

Ilustración 81. navbar TS

“getCurrentUrl()”: Devuelve la URL actual utilizando el enrutador (router) de la aplicación.

“openCreatePost()”: Establece la variable *isCreatePostOpen* en true, lo que abre el formulario de creación de publicaciones.

“closeCreatePost()”: Establece la variable *isCreatePostOpen* en false y reinicia la variable *newPost* a un objeto vacío. Esto cierra el formulario de creación de publicaciones y reinicia los valores del formulario.

“fileInput(event: any)”: Se activa cuando se selecciona un archivo en el campo de entrada de archivo del formulario de creación de publicaciones. Captura el archivo seleccionado (*selectedImage*) y realiza una comprobación de tamaño. Si el archivo supera el tamaño máximo permitido (MAX_SIZE_BYTES), se

redimensiona utilizando la función “resizeImage()”. El archivo redimensionado se asigna nuevamente a *selectedImage*.

“createPost()”: Se activa al hacer clic en el botón "Upload image" en el formulario de creación de publicaciones. Verifica si hay una imagen seleccionada (*selectedImage*). Si existe, crea un objeto *FormData* y agrega la imagen y el encabezado de la publicación a este formulario. Luego, utiliza el servicio “homeService” para guardar la imagen en el servidor y recibe una respuesta. La URL de la imagen guardada se asigna a *newPost.postContent* y el texto de la publicación se asigna a “*newPost.text*”. Finalmente, se cierra el formulario de creación de publicaciones mediante la llamada a *closeCreatePost()*.

“logOut()”: Elimina el token almacenado en el almacenamiento local y redirige al usuario a la página de inicio de sesión ('/loginhome') utilizando el enrutador (router).

5.3 Pruebas de uso

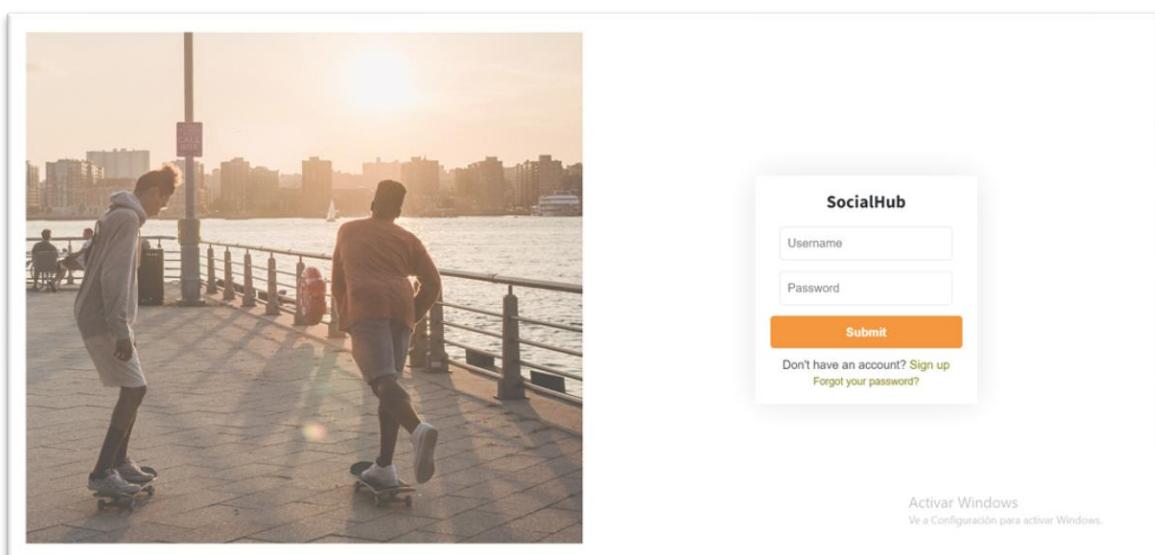


Ilustración 82. pantalla inicio sesion principal

Página de inicio de sesión principal

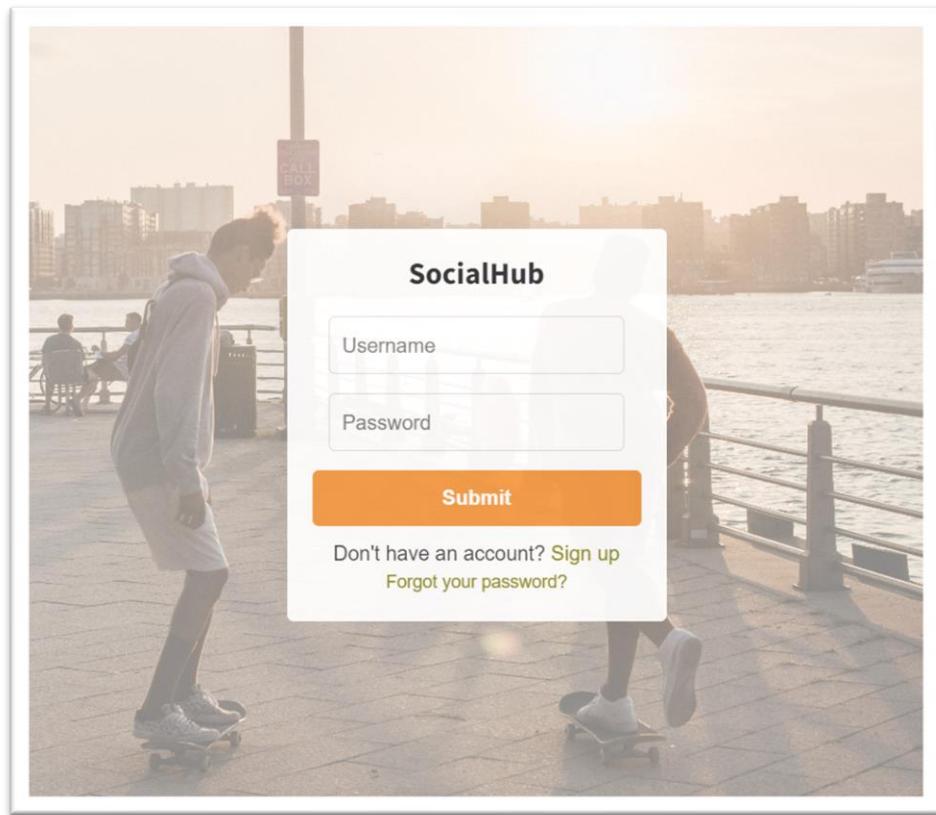


Ilustración 83. pantalla inicio sesión responsive

La pagina de inicio de sesión principal es responsive, se adapta según el tamaño de la pantalla.

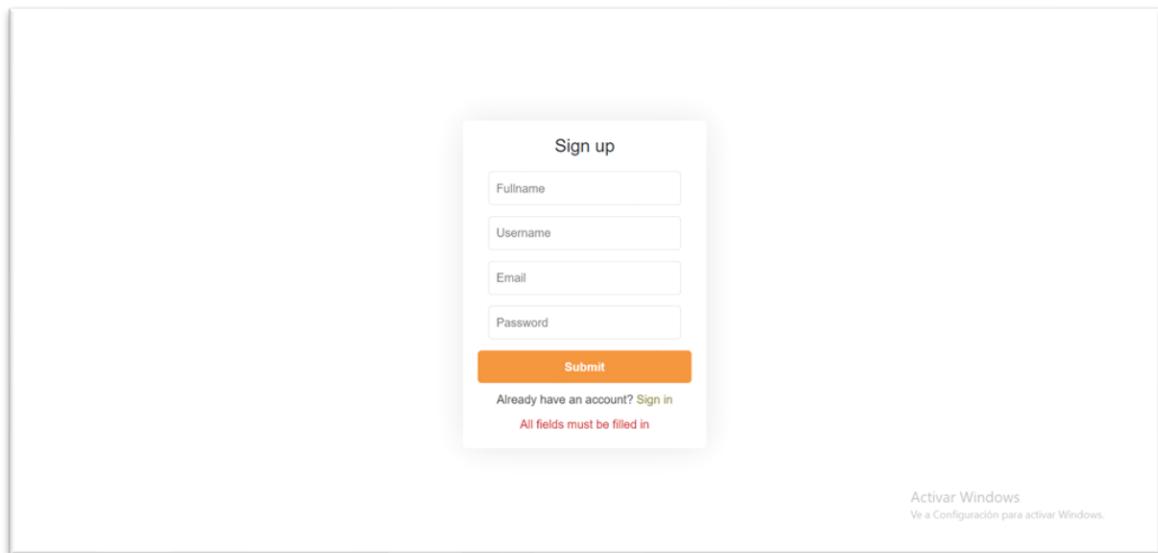


Ilustración 84. Pantalla registro error 1

Los campos deben estar completos.

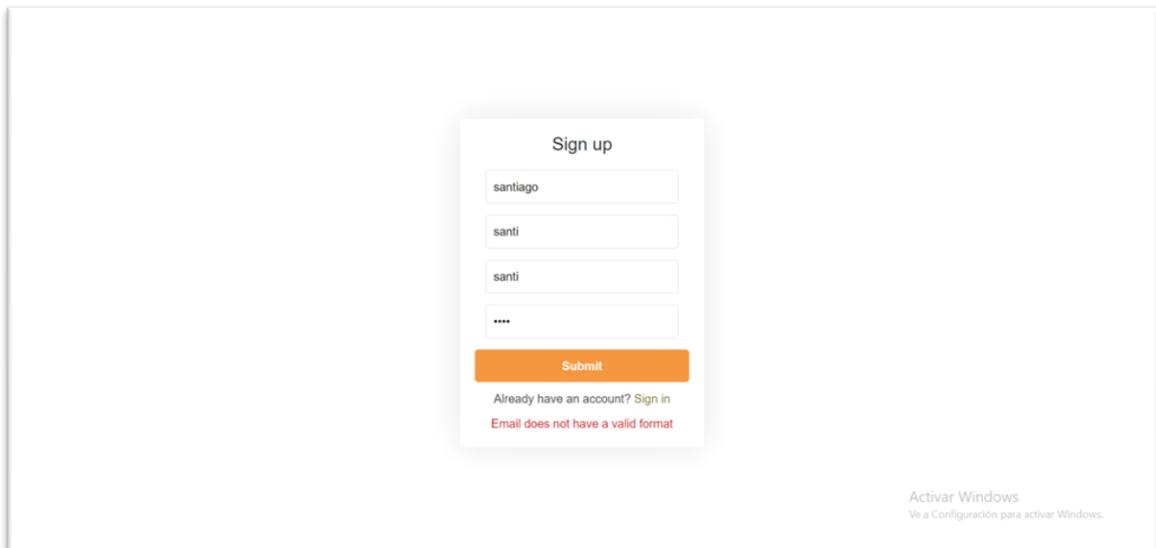


Ilustración 85. pantalla registro error 2

El formato del email debe ser correcto

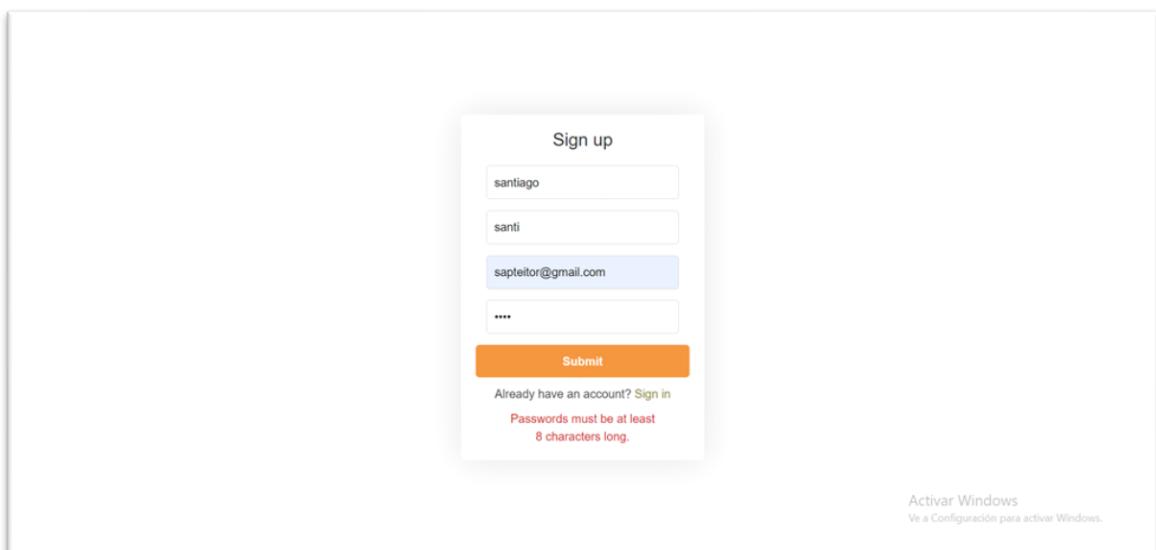


Ilustración 86. pantalla registro error 3

La contraseña debe tener mínimo 8 caracteres.

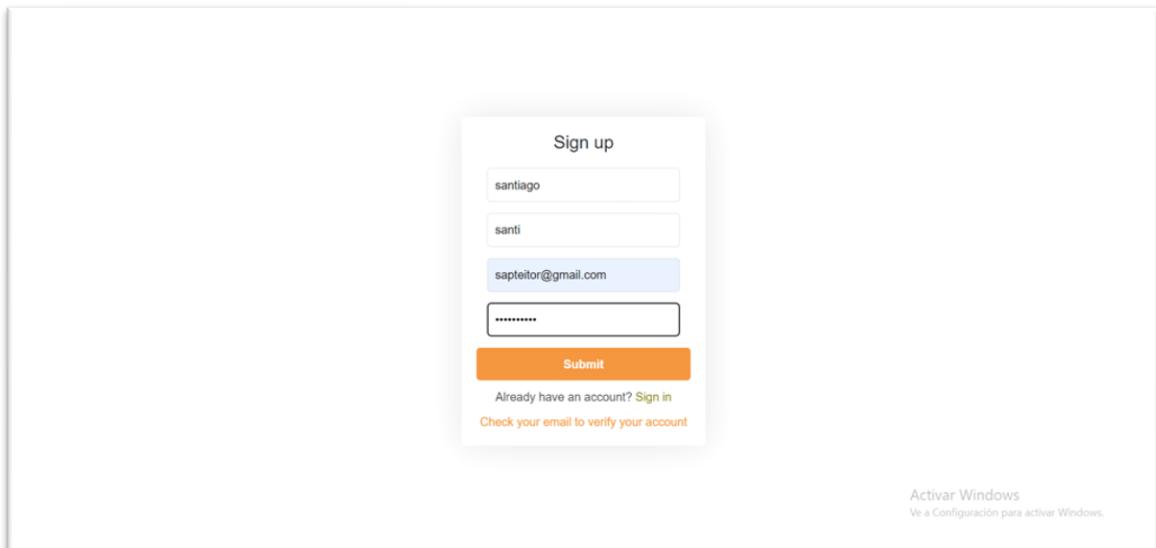


Ilustración 87. pantalla registro

Mensaje de verificación de email.

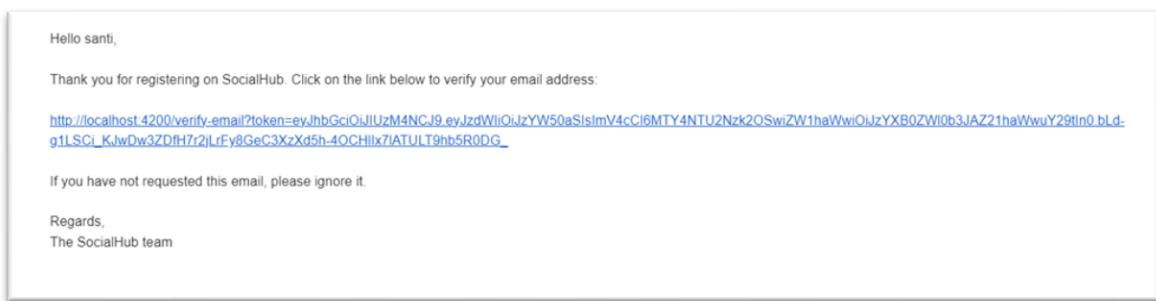


Ilustración 88. email verificación

Correo de verificación de email.

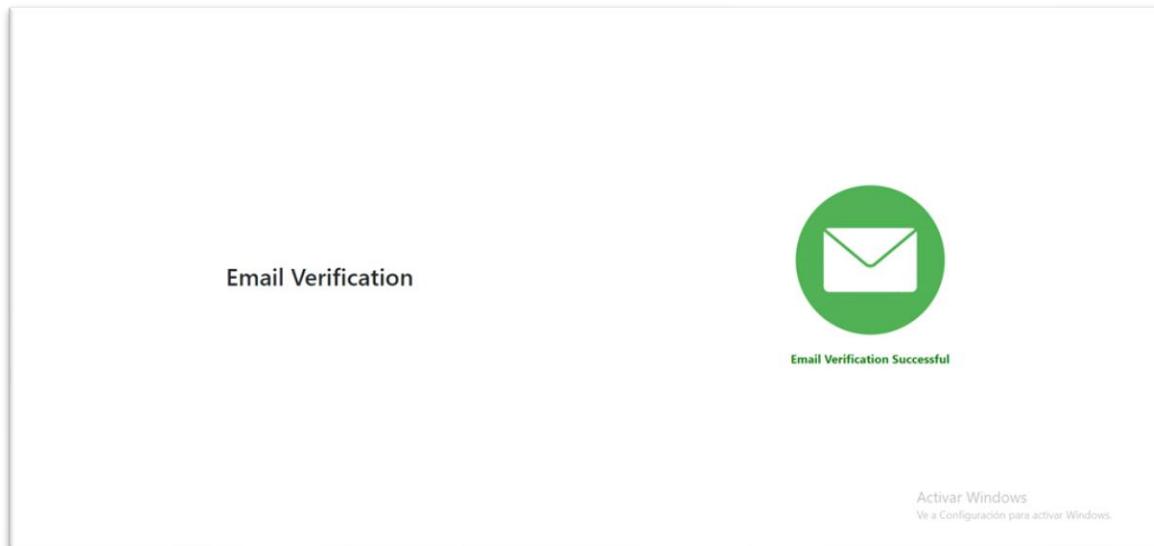


Ilustración 89. pantalla verificacion correcta

Email verificado correctamente.

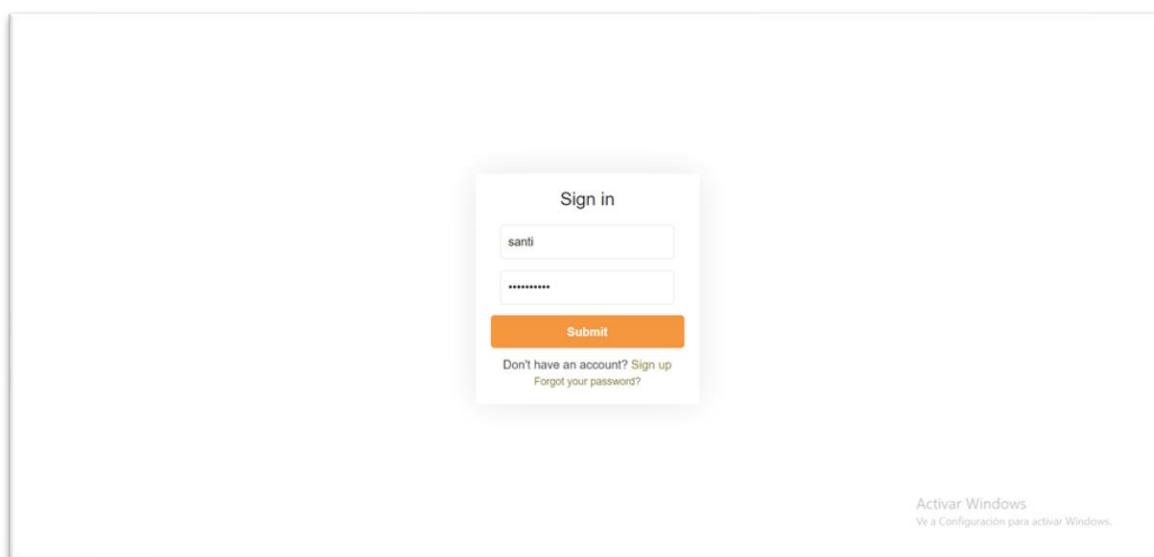


Ilustración 90. pantalla de inicio de sesion alternativa

Pantalla de inicio de sesión alternativa.

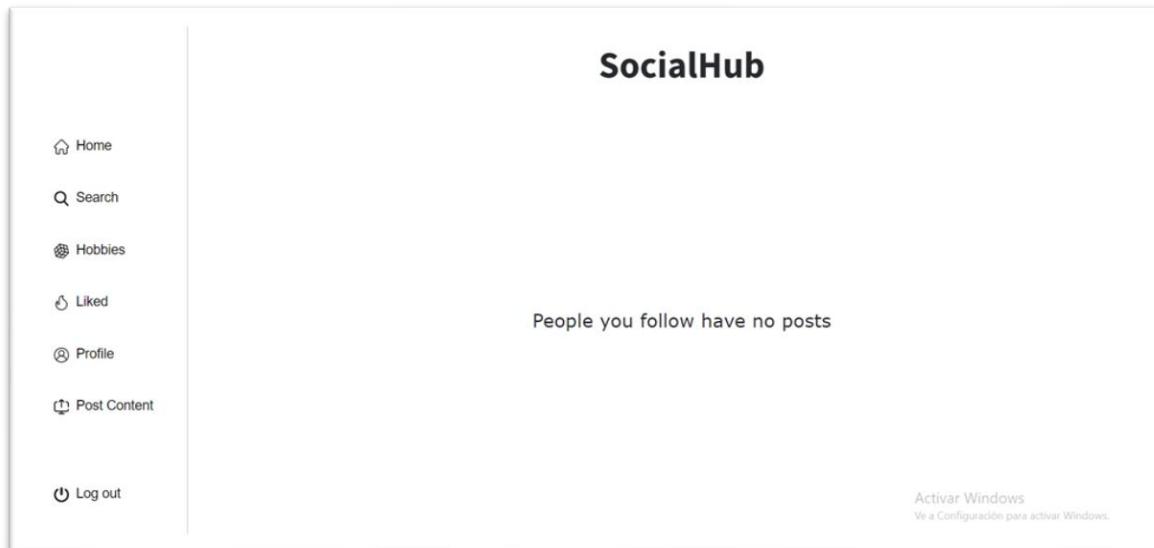


Ilustración 91. feed vacía

Página de inicio (feed)

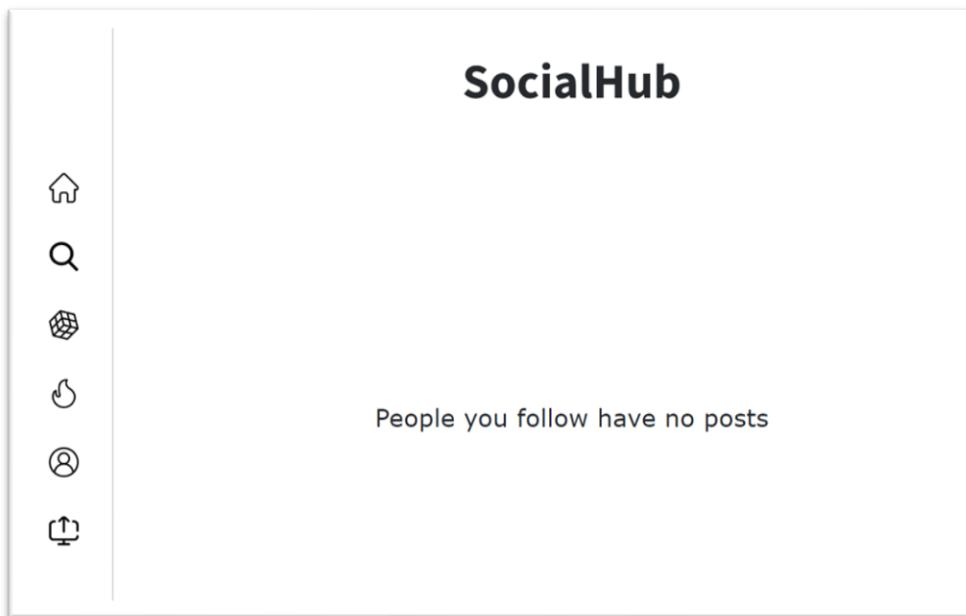


Ilustración 92. barra navegacion responsive

La barra de navegación es responsive, dependiendo del tamaño se adapta.

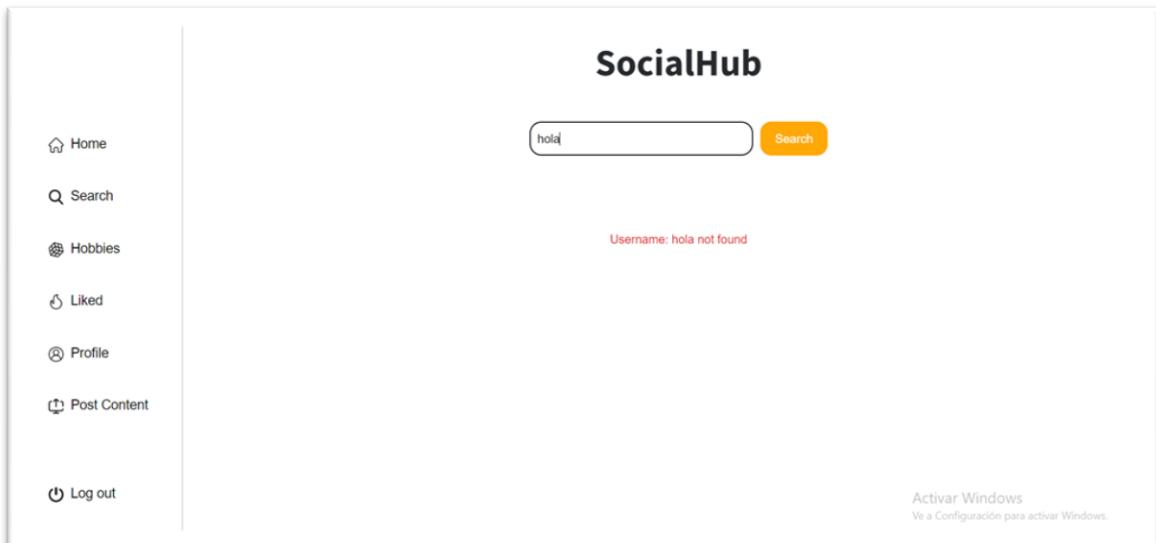


Ilustración 93. buscar usuario error

Pantalla de búsqueda, al no encontrar usuario salta el mensaje.

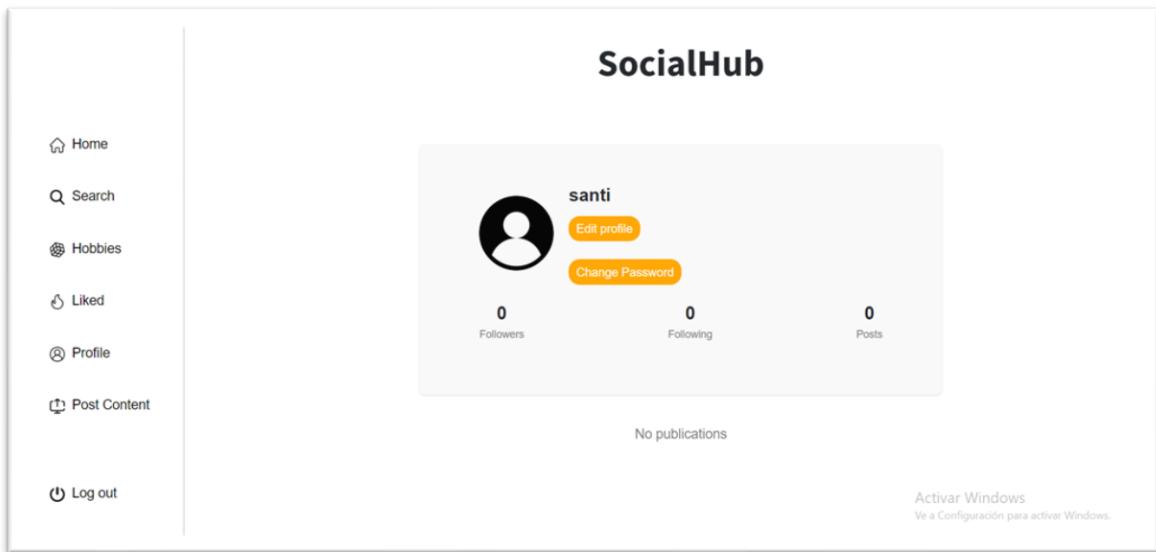


Ilustración 94. perfil personal vacío

Perfil personal.

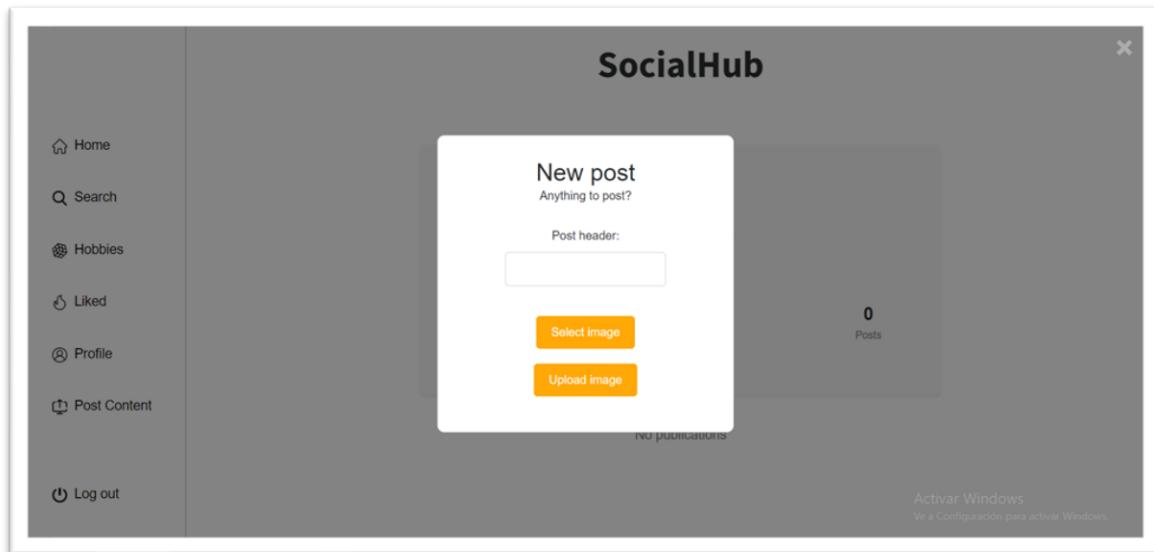


Ilustración 95. nueva publicacion

Subir una fotografía al sistema.

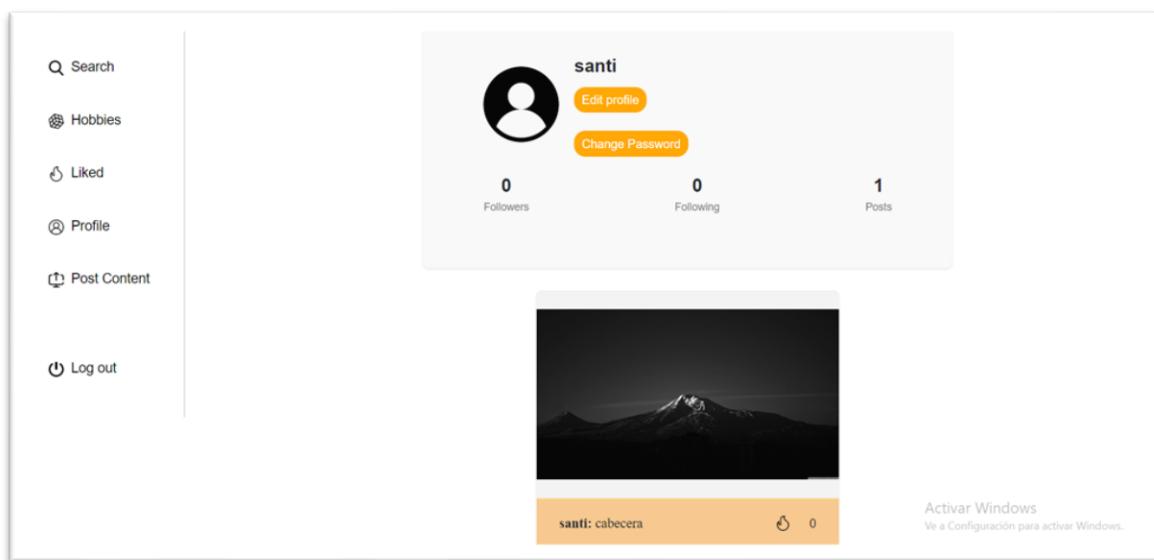


Ilustración 96. perfil personal con publicación

Imagen recién subida mostrada en el perfil personal.

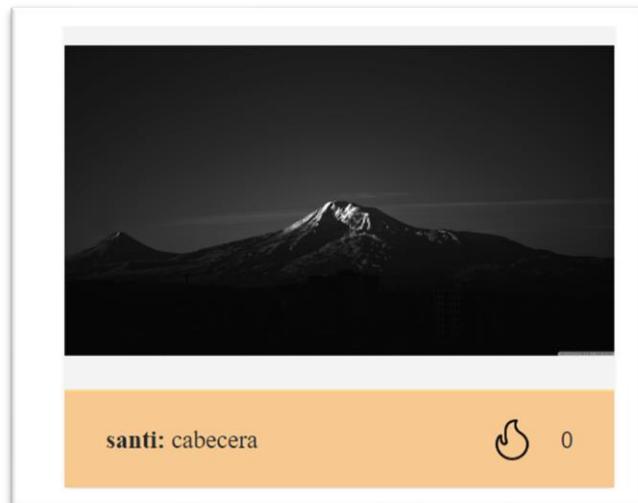


Ilustración 97. publicación

Cuando estas con el ratón encima aumenta de tamaño el botón de *like*.

Ilustración 98. editar perfil

Editar el perfil.

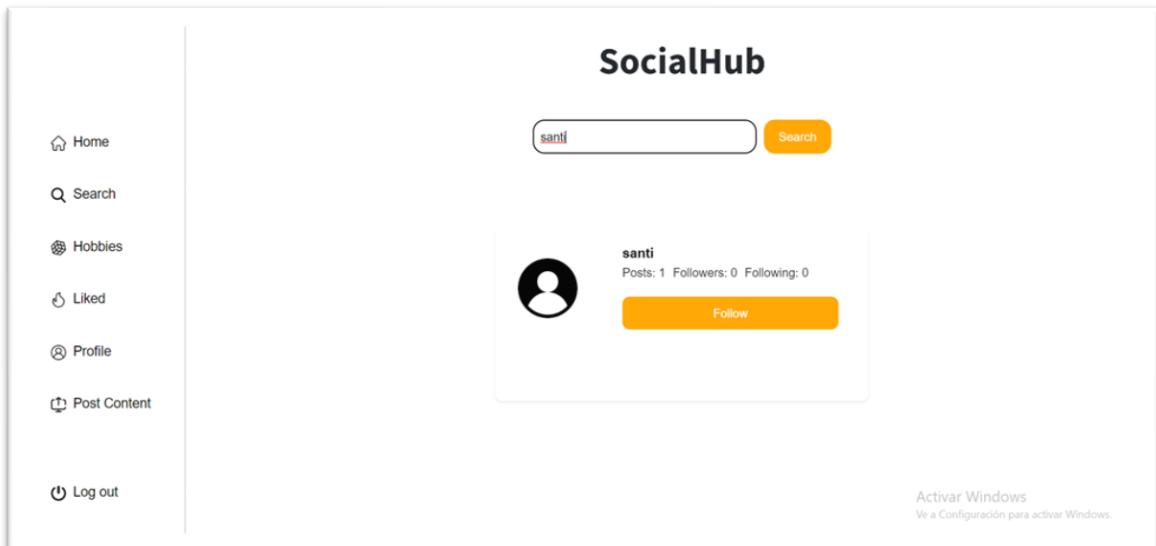


Ilustración 99. buscar usuario encontrado

Buscar usuario encontrado.

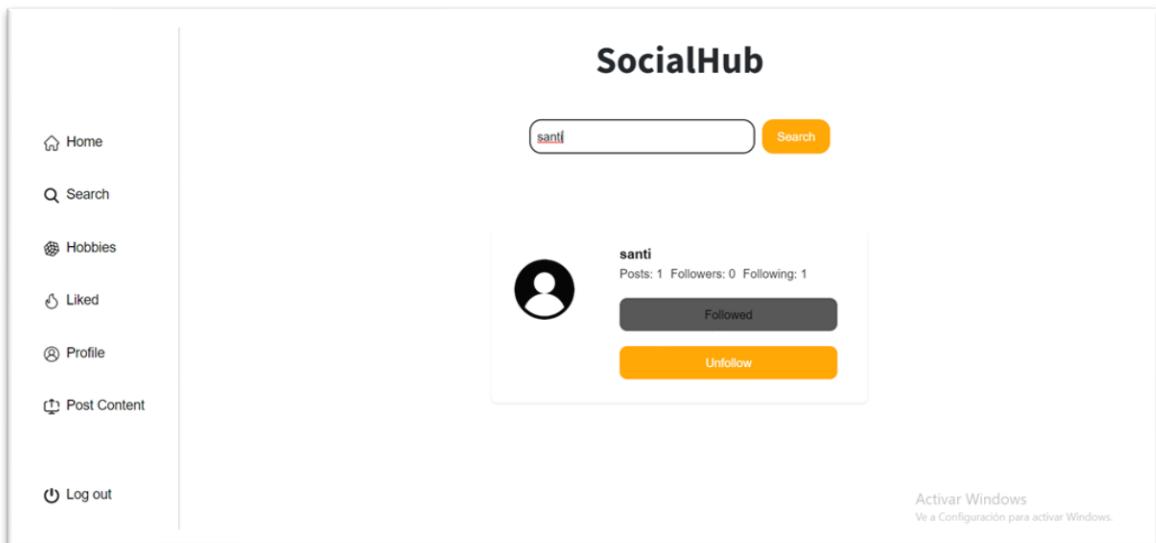


Ilustración 100. usuario seguido

Usuario seguido.

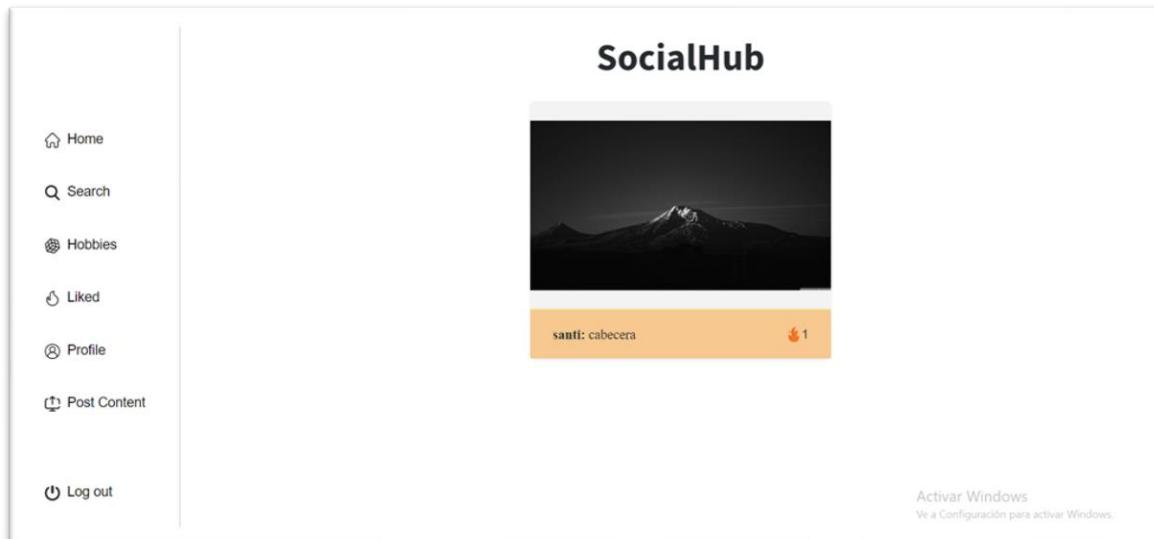


Ilustración 101. feed con publicación

Feed con la publicación del usuario seguido.

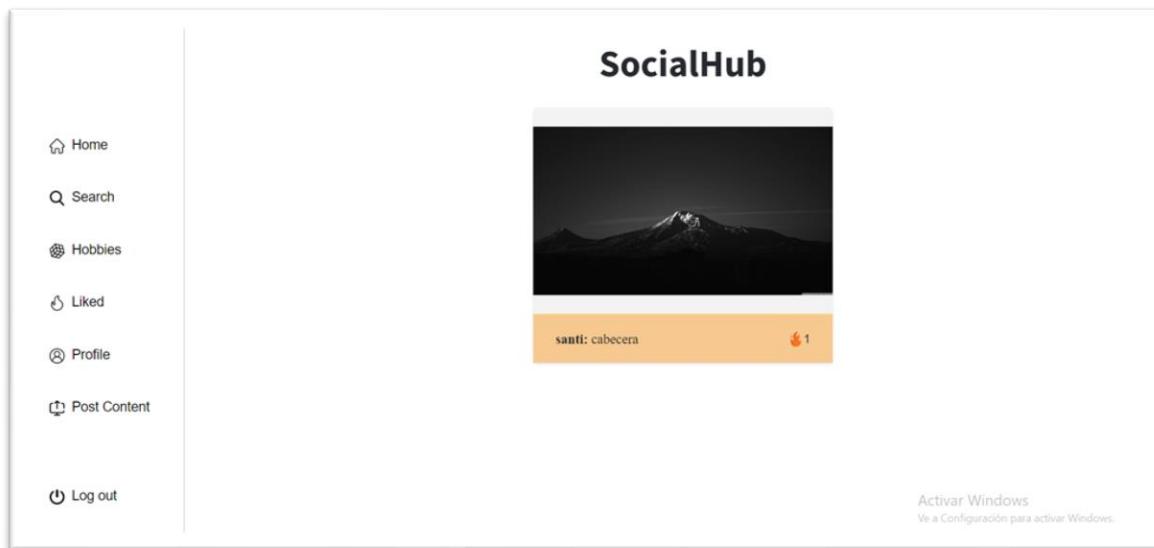


Ilustración 102. Liked publicaciones

Sección de las publicaciones con me gusta.

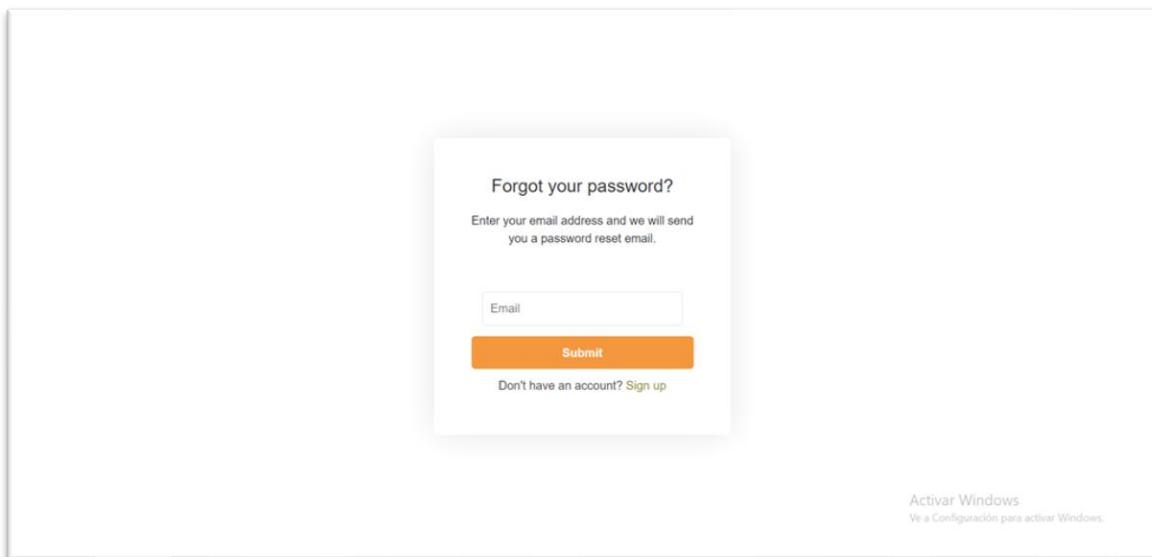


Ilustración 103. Página de olvido de contraseña

Página que corresponde con el enlace de “*Forgot your password?*” del inicio de sesión.

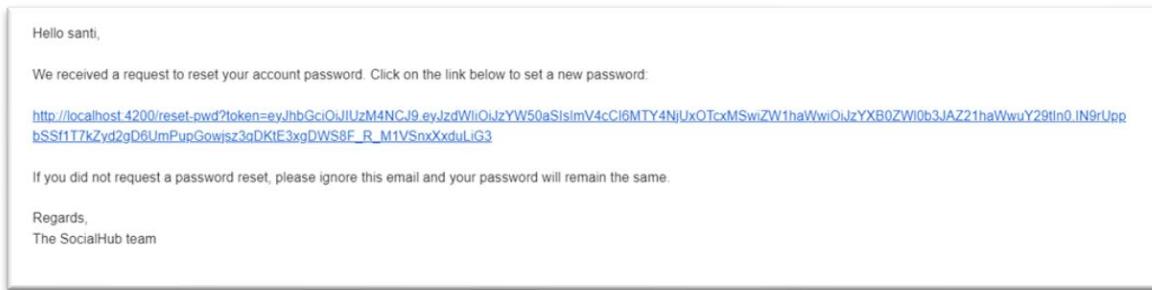


Ilustración 104. Correo de olvido de contraseña

Correo electrónico que se recibe al haber introducido el correo electrónico en la página mostrada anteriormente.

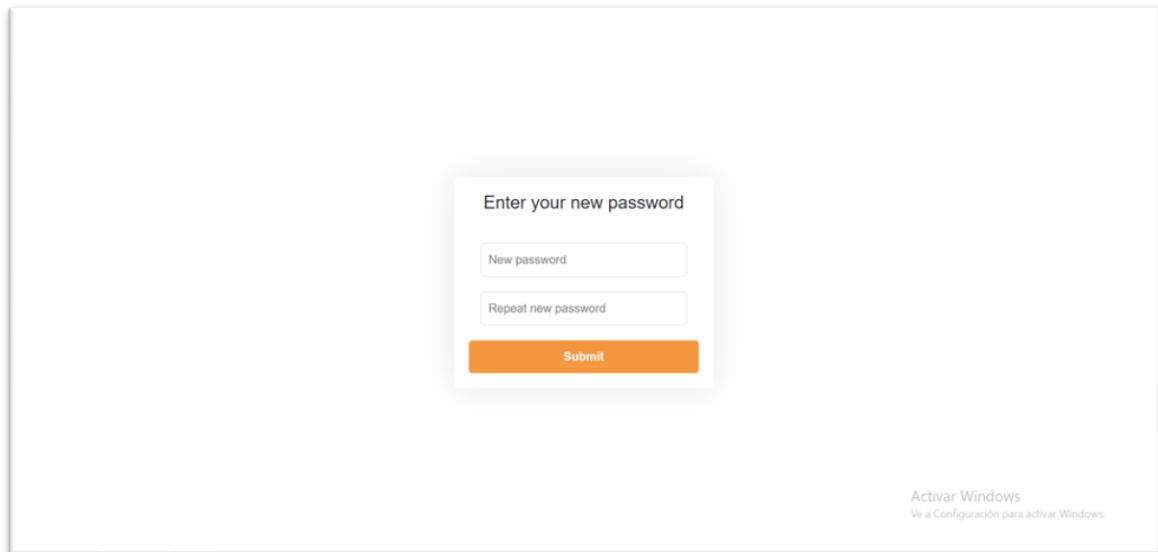


Ilustración 105. Introducir nueva contraseña

Página de cambio de contraseña.

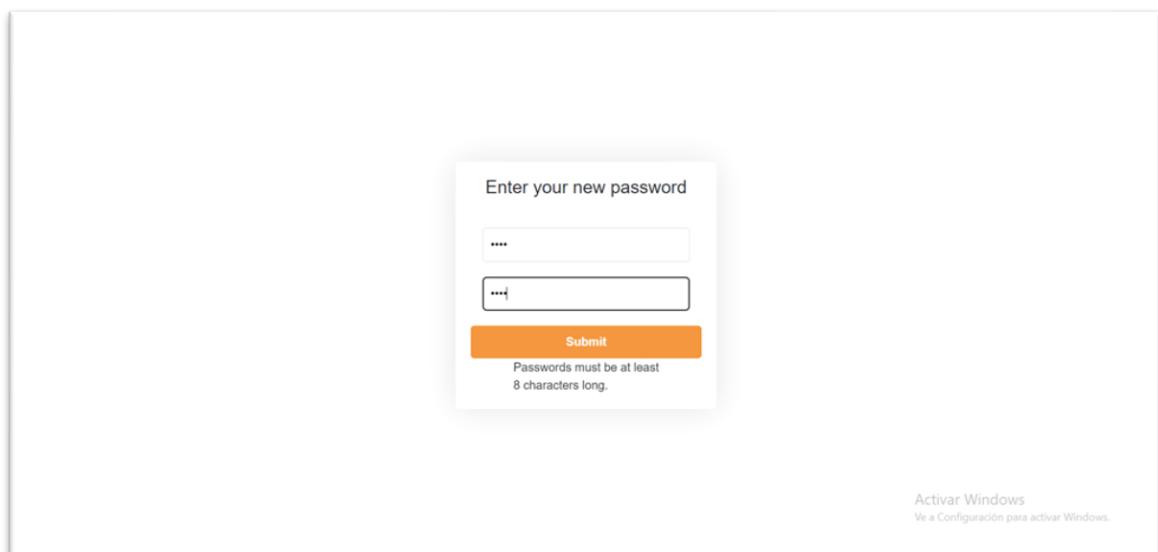


Ilustración 106. Comprobacion1 cambio contraseña

Comprobación de la longitud de la contraseña requerida.

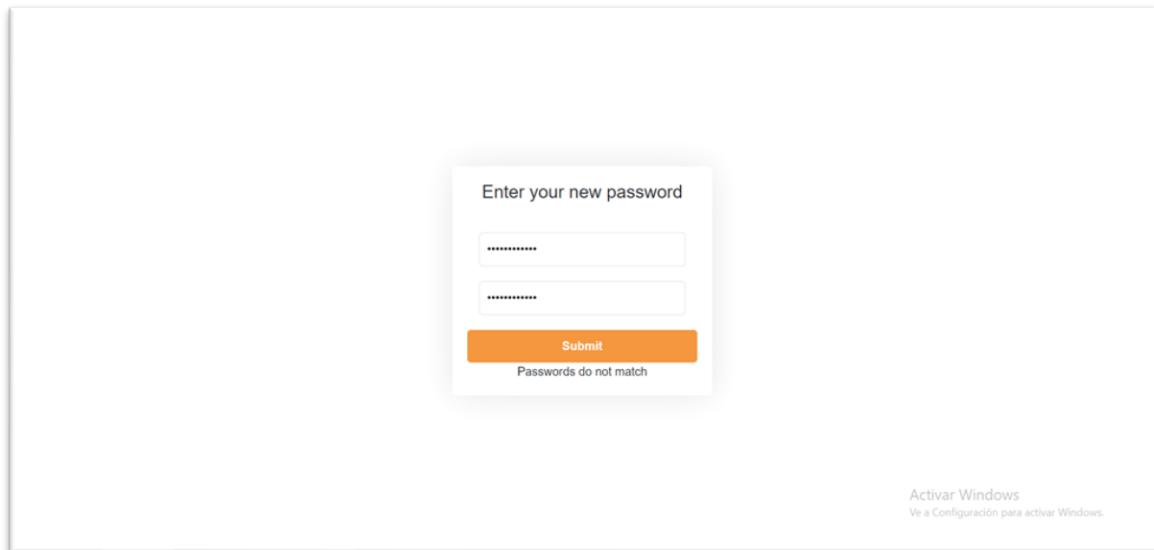


Ilustración 107. Comprobacion2 cambio contraseña

Comprobación de que las contraseñas introducidas coincidan.

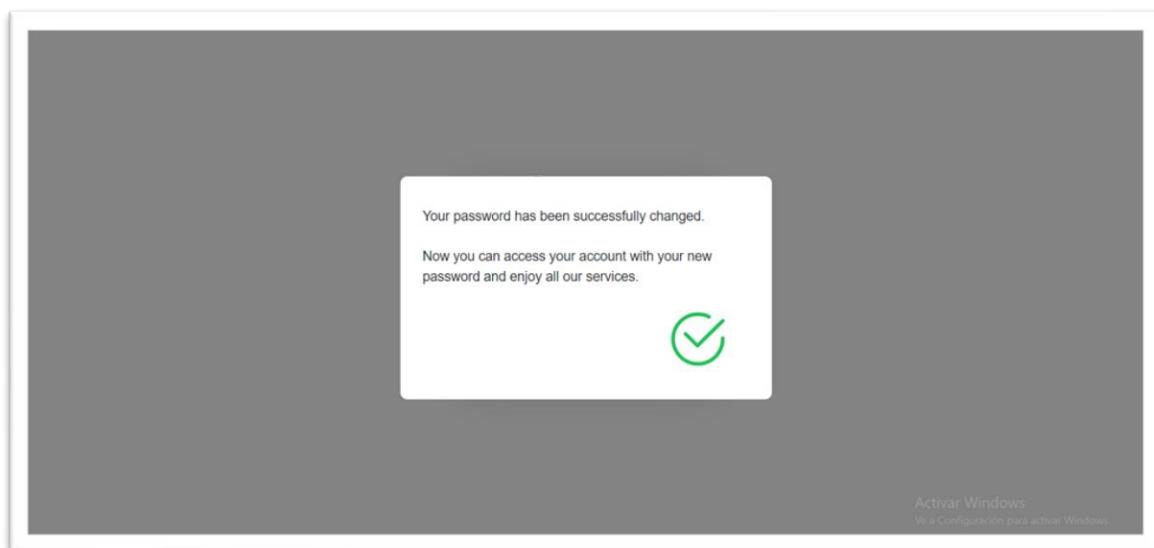


Ilustración 108. Cambio de contraseña exitoso

Mensaje de cambio de contraseña exitoso.

6 Conclusiones

En este trabajo de fin de grado, se ha abordado el desarrollo de una API REST y un Frontend para una red social utilizando Java con Spring Boot y Angular. A lo largo del proyecto, se ha aplicado un enfoque de ingeniería de software para garantizar la eficiencia, escalabilidad y calidad del sistema desarrollado. Además, se han utilizado buenas prácticas de la programación para asegurar la calidad y legibilidad del código. A continuación, se presentan las principales conclusiones obtenidas:

6.1 Logro de objetivos

Logro de los objetivos: Se lograron los objetivos establecidos al inicio del trabajo. Se diseñó e implementó una API REST robusta y eficiente utilizando Java con Spring Boot, que permite gestionar las funcionalidades clave de una red social, como la creación de usuarios, publicaciones y comentarios. Además, se desarrolló un Frontend atractivo y fácil de usar con Angular, que proporciona una interfaz intuitiva para los usuarios.

Aplicación de la ingeniería de software: Se aplicaron los principios y las prácticas de la ingeniería de software en todas las etapas del proyecto. Se realizó un análisis exhaustivo de los requisitos, se diseñó una arquitectura sólida, se implementó el sistema siguiendo buenas prácticas de codificación y se realizaron pruebas exhaustivas para garantizar la calidad del software.

Uso de tecnologías modernas: El uso de Java con Spring Boot en el desarrollo de la API REST y Angular en el desarrollo del Frontend permitió aprovechar tecnologías modernas y ampliamente adoptadas en la industria. Estas tecnologías ofrecieron una gran cantidad de funcionalidades y facilitaron la integración de diferentes componentes del sistema.

Importancia de la documentación y las pruebas: Durante el desarrollo del proyecto, se hizo hincapié en la documentación adecuada y en la realización de pruebas. La documentación clara y completa facilita la comprensión y el mantenimiento del código, mientras que las pruebas garantizan el correcto funcionamiento de las funcionalidades y ayudan a detectar y corregir errores antes de la implementación en producción.

Aprendizaje y mejora continua: El desarrollo de este trabajo permitió adquirir un amplio conocimiento sobre el diseño e implementación de APIs REST y Frontend utilizando Java con Spring Boot y Angular. Además, se identificaron áreas de mejora y se destacaron posibles futuras extensiones del sistema, como la implementación de funcionalidades adicionales o la optimización del rendimiento.

6.2 Aprendizajes y dificultades encontradas

La implementación de este proyecto ha sido una experiencia fascinante y enriquecedora para mí. A lo largo del proceso, he tenido la oportunidad de sumergirme en el mundo del desarrollo web y he adquirido un conocimiento invaluable en esta área. La amplitud del proyecto me ha permitido trabajar tanto en el desarrollo del Backend como en el Frontend, lo que ha sido una experiencia completa y altamente gratificante.

Durante este camino, me he encontrado con desafíos interesantes que me han impulsado a superar mis límites y aprender de manera continua. Uno de estos desafíos fue la implementación del envío de imágenes utilizando “*Multipart*” y bytes de *arrays*. Fue una tarea que me llevó tiempo comprender y dominar, pero a medida que investigaba y probaba diferentes enfoques, logré encontrar soluciones efectivas y lograr la funcionalidad deseada.

Otra área en la que enfrenté dificultades fue en la creación de diseños utilizando CSS y HTML. Teniendo poca experiencia previa en esta área, tuve que dedicar tiempo y esfuerzo en aprender los fundamentos y explorar las mejores prácticas. A medida que avanzaba en el proyecto, fui capaz de mejorar mis habilidades en la maquetación y diseño de páginas web, logrando resultados estéticamente agradables y funcionales.

Además, las configuraciones de seguridad de la aplicación presentaron un desafío adicional. Asegurar la integridad y protección de los datos fue una prioridad, y tuve que investigar y aplicar medidas de seguridad adecuadas. Aunque inicialmente me enfrenté a cierta complejidad, pude implementar las configuraciones necesarias para garantizar un entorno seguro y protegido.

Por último, otro obstáculo al que me enfrenté fue mostrar correctamente las imágenes en pantalla. Requirió una comprensión profunda del procesamiento de imágenes y la manipulación de datos en el Frontend. A través de pruebas y análisis cuidadoso, logré encontrar la solución adecuada y asegurarme de que las imágenes se visualizaran de manera correcta y efectiva para los usuarios.

Finalizando, a pesar de los desafíos encontrados durante el proceso de implementación, he logrado superarlos con éxito y alcanzar los objetivos establecidos. Cada dificultad ha sido una oportunidad para aprender y crecer en mi desarrollo profesional. Esta experiencia me ha brindado un mayor conocimiento y habilidades en el campo del desarrollo web, y me siento orgulloso de haber enfrentado y superado los obstáculos con determinación y resolución.

6.3 Evaluación del código

Líneas de código utilizadas en Frontend:

language	files	code	comment	blank	total
TypeScript	39	1,355	12	263	1,630
CSS	17	1,343	23	252	1,618
HTML	17	549	13	66	628

Líneas de código en Backend:

language	files	code	comment	blank	total
Java	63	1,626	71	455	2,152

7 Análisis de Impacto

En este capítulo, se realizará un análisis del impacto potencial de los resultados obtenidos durante la realización del TFG en diversos contextos, considerando los aspectos personal, empresarial, social, económico, medioambiental y cultural. Además, se destacarán los beneficios esperados y los posibles efectos perjudiciales derivados de la solución propuesta.

7.1 Impacto Personal

La implementación exitosa del proyecto ha supuesto un impacto significativo en mi crecimiento personal y profesional. Durante este proceso, he tenido la oportunidad de sumergirme en el mundo del desarrollo web y aprender una amplia gama de conceptos y técnicas que han enriquecido mi conjunto de habilidades.

A medida que avanzaba en la implementación, me enfrenté a desafíos que me obligaron a investigar, experimentar y resolver problemas de manera creativa. Cada obstáculo superado me brindó una sensación de logro y aumentó mi confianza en mis capacidades como desarrollador web.

Además de adquirir conocimientos técnicos, la implementación exitosa del proyecto me ha brindado una valiosa experiencia en la gestión del tiempo, la planificación de tareas y la coordinación de recursos. A lo largo del proceso, he aprendido a establecer metas realistas, organizar eficientemente mis actividades y adaptarme a los cambios y desafíos imprevistos que surgieron en el camino.

La finalización exitosa de la implementación de calidad no solo me ha proporcionado una sensación de logro personal, sino que también ha fortalecido mi confianza en mi capacidad para enfrentar proyectos futuros con determinación y profesionalismo. Estoy orgulloso de haber superado los obstáculos y haber logrado una implementación que cumple con altos estándares de calidad.

7.2 Impacto Empresarial

Desde una perspectiva empresarial, la implementación de la red social se presenta como una valiosa oportunidad para aprovechar y capitalizar diversas ventajas y beneficios.

La implementación de una red social ofrece la oportunidad de generar ingresos a través de diferentes modelos de negocio. Entre ellos se encuentran la publicidad segmentada, donde se pueden ofrecer espacios publicitarios, y la monetización de contenido premium o servicios adicionales. Estas estrategias pueden contribuir al crecimiento financiero.

Asimismo, una red social bien implementada puede convertirse en una poderosa herramienta de marketing y promoción. La posibilidad de compartir contenido relevante y viral, así como la capacidad de generar recomendaciones y opiniones positivas de los usuarios, puede generar una importante visibilidad y reputación para la marca.

7.3 Impacto Social

La red social propuesta puede tener un impacto social significativo. Facilita la interconexión de personas de diferentes lugares, culturas e intereses, promoviendo la diversidad y la inclusión. Los usuarios pueden compartir experiencias, conocimientos y recursos, fomentando la colaboración y el aprendizaje mutuo. Además, la red social puede servir como plataforma para la difusión de información relevante, la promoción de causas sociales y la sensibilización sobre problemas globales.

7.4 Impacto Económico

En términos económicos, la implementación exitosa de la solución propuesta puede tener un impacto positivo. La generación de oportunidades de negocio para empresas y emprendedores puede impulsar el crecimiento económico local y regional. Además, la creación de empleos directos e indirectos relacionados con el mantenimiento, soporte y expansión de la red social puede contribuir a la generación de empleo y al desarrollo económico.

7.5 Impacto Medioambiental

La solución propuesta tiene un impacto medioambiental indirecto, pero importante. Al proporcionar una plataforma virtual para la interacción social, se reduce la necesidad de desplazamientos físicos y, por lo tanto, se disminuye la emisión de gases de efecto invernadero y se contribuye a la reducción de la huella de carbono. Además, la sensibilización y promoción de prácticas sostenibles en la red social pueden influir positivamente en la conciencia ambiental de los usuarios.

7.6 Impacto Cultural

La red social propuesta puede tener un impacto cultural al permitir a los usuarios compartir y apreciar diferentes expresiones culturales, como música, arte, tradiciones y costumbres. Esto puede contribuir a la preservación y difusión de la diversidad cultural, promoviendo la tolerancia y el entendimiento intercultural.

7.7 Referencia a los Objetivos de Desarrollo Sostenible (ODS)

Se recomienda analizar el potencial impacto de la solución propuesta haciendo referencia a los Objetivos de Desarrollo Sostenible (ODS) de la Agenda 2030 establecidos por las Naciones Unidas. En particular, se pueden identificar los ODS relevantes para la solución propuesta, como la igualdad de género (ODS 5), la reducción de la desigualdad (ODS 10), la acción por el clima (ODS 13) y la paz, justicia e instituciones sólidas (ODS 16), entre otros.

7.8 Consideración del Impacto en las Decisiones Tomadas

A lo largo del trabajo, se tuvo en cuenta el impacto en la toma de decisiones. Se consideraron aspectos como la seguridad de los usuarios, la eficiencia energética, el diseño intuitivo y la escalabilidad del sistema. Las decisiones tomadas buscaron maximizar los beneficios potenciales y minimizar los posibles efectos perjudiciales en los diferentes contextos mencionados anteriormente.

8 Bibliografía

Publicaciones utilizadas en el estudio y desarrollo del trabajo.

- [1] "Estadísticas uso de redes sociales en 2023 (informe España y mundo)". Una Vida Online. <https://unavidaonline.com/estadisticas-redes-sociales/#:~:text=En%20el%20mundo%20hay%20de,4%20de%20la%20población%20mundial>
- [2] "Generalidades del protocolo HTTP - HTTP | MDN". MDN Web Docs. <https://developer.mozilla.org/es/docs/Web/HTTP/Overview>
- [3] "IBM documentation". IBM - Deutschland | IBM. <https://www.ibm.com/docs/es/baw/20.x?topic=formats-javascript-object-notation-json-format>
- [4] Y. Muradas. "Qué es spring framework y por qué usarlo". OpenWebinars.net. <https://openwebinars.net/blog/conoce-que-es-spring-framework-y-por-que-usarlo/>
- [5] "¿Qué es java? | IBM". IBM - Deutschland | IBM. <https://www.ibm.com/es-es/topics/java>
- [6] Postman <https://www.postman.com/>
- [7] "¿Qué es Angular y cuáles son sus ventajas?" Tutoriales Hostinger. <https://www.hostinger.es/tutoriales/que-es-angular>
- [8] "HTML: Lenguaje de etiquetas de hipertexto | MDN". MDN Web Docs. <https://developer.mozilla.org/es/docs/Web/HTML>
- [9] "Css | mdn". MDN Web Docs. <https://developer.mozilla.org/es/docs/Web/CSS>
- [10] "JavaScript with syntax for types." TypeScript: JavaScript With Syntax For Types. <https://www.typescriptlang.org/> (accedido el 30 de mayo de 2023).
- [11] "Angular". Angular. <https://angular.io/cli>
- [12] Contributors to Wikimedia projects. "Web development tools - Wikipedia". Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Web_development_tools
- [13] "¿Qué es MySQL? Explicación detallada para principiantes". Tutoriales Hostinger. <https://www.hostinger.es/tutoriales/que-es-mysql>
- [14] "What Is GitHub? A Beginner's Introduction to GitHub". Kinsta®. <https://kinsta.com/knowledgebase/what-is-github/>
- [15] Contributors to Wikimedia projects. "Requirements analysis - Wikipedia". Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Requirements_analysis
- [16] "Qué es la especificación de requisitos: Definición, mejores herramientas y técnicas | Guía - Soluciones Visure". Visure Solutions. <https://visuresolutions.com/es/blog/requirements-specification/>
- [17] "MVC - glosario de MDN web docs: Definiciones de términos relacionados con la web | MDN". MDN Web Docs. <https://developer.mozilla.org/es/docs/Glossary/MVC>

[18] "Uso de diferentes tipos de índices SQL Server". SQL Shack - articles about database auditing, server performance, data recovery, and more. <https://www.sqlshack.com/es/uso-de-diferentes-tipos-de-indices-sql-server/>

[19] "Free icons and stickers - millions of images to download". Flaticon. <https://www.flaticon.com/>

[20] "Budgeron Bach post". Pexels. <https://www.pexels.com/es-es/foto/hombres-deportivos-sin-rostro-patinando-sobre-terraplen-alatardecer-5157169/>

Hay que utilizar un sistema internacional para referencias bibliográficas, de acuerdo con las indicaciones del tutor. Por ejemplo, el [sistema de IEEE](#).

9 Índice de ilustraciones

<i>Ilustración 1. Diagrama de Gantt</i>	4
<i>Ilustración 2. JSON logo</i>	6
<i>Ilustración 3. Spring logo</i>	6
<i>Ilustración 4. Java logo</i>	7
<i>Ilustración 5. Postman logo</i>	7
<i>Ilustración 6. Angular logo</i>	8
<i>Ilustración 7. HTML logo</i>	8
<i>Ilustración 8. CSS logo</i>	8
<i>Ilustración 9. TypeScript logo</i>	9
<i>Ilustración 10. MySQL logo</i>	9
<i>Ilustración 11. GitHub logo</i>	10
<i>Ilustración 12. Git logo</i>	10
<i>Ilustración 13. Diagrama MVC</i>	18
<i>Ilustración 14. Diagrama de Entidad-Relación</i>	20
<i>Ilustración 15. Prototipo Baja Fidelidad 1</i>	30
<i>Ilustración 16. Prototipo Baja Fidelidad 2</i>	30
<i>Ilustración 17. Prototipo Baja Fidelidad 3</i>	31
<i>Ilustración 18. Prototipo Baja Fidelidad 4</i>	31
<i>Ilustración 19. Prototipo Baja Fidelidad 5</i>	32
<i>Ilustración 20. Prototipo Baja Fidelidad 6</i>	32
<i>Ilustración 21. Proceso de diseño del logo</i>	33
<i>Ilustración 22. SocialHub logo</i>	33
<i>Ilustración 23. SocialHub logo en ventana</i>	33
<i>Ilustración 24. Código pom.xml</i>	37
<i>Ilustración 25. application.properties</i>	38
<i>Ilustración 26. CorsConfig</i>	39
<i>Ilustración 27. Security Config</i>	40
<i>Ilustración 28. JWTAuthenticationFilter</i>	41
<i>Ilustración 29. JWTAuthorizationFilter</i>	42
<i>Ilustración 30. Paquete Seguridad API</i>	42
<i>Ilustración 31. Anotaciones Controlles Class</i>	43
<i>Ilustración 32. Método register API</i>	43
<i>Ilustración 33. UserController 1</i>	44
<i>Ilustración 34. UserController 2</i>	45
<i>Ilustración 35. UserController 3</i>	46
<i>Ilustración 36. Método Register Service</i>	47
<i>Ilustración 37. Método verifyEmail, Service</i>	48
<i>Ilustración 38. Metodo login, Service</i>	49
<i>Ilustración 39. Metodo forgotPass, Service</i>	50
<i>Ilustración 40. Metodo authToken, Service</i>	50
<i>Ilustración 41. Metodo changePass, Service</i>	51
<i>Ilustración 42. Metodo createPost, Service</i>	51
<i>Ilustración 43. Metodo getFollowingPosts, Service</i>	52
<i>Ilustración 44. Metodo searchUser, Service</i>	53
<i>Ilustración 45. Metodo followRequest, Service</i>	54
<i>Ilustración 46. Metodo unfollow, Service</i>	55
<i>Ilustración 47. Metodo showProfile, Service</i>	56
<i>Ilustración 48. Metodo editUser, Service</i>	57
<i>Ilustración 49. Metodo like, Service</i>	58
<i>Ilustración 50. Metodo getLiked, Service</i>	59
<i>Ilustración 51. Metodos privados Service</i>	60
<i>Ilustración 52. Metodo sendEmailVerification</i>	61
<i>Ilustración 53. Metodo sendPassEmail</i>	61
<i>Ilustración 54. Query 1</i>	63
<i>Ilustración 55. Query 2</i>	63

<i>Ilustración 56. auth-routing</i>	66
<i>Ilustración 57. loginhome HTML</i>	67
<i>Ilustración 58. loginhome TS</i>	68
<i>Ilustración 59. login HTML</i>	69
<i>Ilustración 60. login TS</i>	70
<i>Ilustración 61. Register HTML</i>	71
<i>Ilustración 62. register TS</i>	72
<i>Ilustración 63. email-verification HTML</i>	73
<i>Ilustración 64. email-verification TS</i>	74
<i>Ilustración 65. forgotPwd HTML</i>	75
<i>Ilustración 66. forgotPwd TS</i>	75
<i>Ilustración 67. resetPwd HTML</i>	76
<i>Ilustración 68. resetPwd TS</i>	77
<i>Ilustración 69. home-routing</i>	78
<i>Ilustración 70. feed HTML</i>	79
<i>Ilustración 71. feed TS</i>	80
<i>Ilustración 72. search HTML</i>	81
<i>Ilustración 73. search TS</i>	83
<i>Ilustración 74. like HTML</i>	84
<i>Ilustración 75. like TS</i>	85
<i>Ilustración 76. profile HTML</i>	87
<i>Ilustración 77. profile TS</i>	89
<i>Ilustración 78. userprofile HTML</i>	91
<i>Ilustración 79. userprofile TS</i>	92
<i>Ilustración 80. navbar HTML</i>	93
<i>Ilustración 81. navbar TS</i>	95
<i>Ilustración 82. pantalla inicio sesion principal</i>	96
<i>Ilustración 83. pantalla inicio sesión responsive</i>	97
<i>Ilustración 84. Pantalla registro error 1</i>	97
<i>Ilustración 85. pantalla registro error 2</i>	98
<i>Ilustración 86. pantalla registro error 3</i>	98
<i>Ilustración 87. pantalla registro</i>	99
<i>Ilustración 88. email verificacion</i>	99
<i>Ilustración 89. pantalla verificacion correcta</i>	100
<i>Ilustración 90. pantalla de inicio de sesion alternativa</i>	100
<i>Ilustración 91. feed vacia</i>	101
<i>Ilustración 92. barra navegacion responsive</i>	101
<i>Ilustración 93. buscar usuario error</i>	102
<i>Ilustración 94. perfil personal vacio</i>	102
<i>Ilustración 95. nueva publicacion</i>	103
<i>Ilustración 96. perfil personal con publicación</i>	103
<i>Ilustración 97. publicación</i>	104
<i>Ilustración 98. editar perfil</i>	104
<i>Ilustración 99. buscar usuario encontrado</i>	105
<i>Ilustración 100. usuario seguido</i>	105
<i>Ilustración 101. feed con publicación</i>	106
<i>Ilustración 102. Liked publicaciones</i>	106
<i>Ilustración 103. Página de olvido de contraseña</i>	107
<i>Ilustración 104. Correo de olvido de contraseña</i>	107
<i>Ilustración 105. Introducir nueva contraseña</i>	108
<i>Ilustración 106. Comprobacion1 cambio contraseña</i>	108
<i>Ilustración 107. Comprobacion2 cambio contraseña</i>	109
<i>Ilustración 108. Cambio de contraseña exitoso</i>	109