universidad de león

Escuela de Ingenierías

Industrial, Informática y Aeroespacial

GRADO EN INGENIERÍA INFORMÁTICA

Trabajo de Fin de Grado

RS CHAT: APLICACIÓN WEB DE CHAT PARA COMUNICACIÓN EN TIEMPO REAL ENTRE ESTUDIANTES Y DOCENTES.

RS CHAT: REAL TIME CHAT WEB APPLICATION FOR STUDENTS AND TEACHERS COMMUNICATION.

Autor: Samuel Castrillo Domínguez

Tutor: Eva María Cuervo Fernández

Junio, 2023

UNIVERSIDAD DE LEÓN

Escuela de Ingenierías Industrial, Informática y Aeroespacial

GRADO EN INGENIERÍA INFORMÁTICA

Trabajo de Fin de Grado

ALUMNO: Samuel Castrillo Domínguez

TUTOR: Eva María Cuervo Fernández

TÍTULO: RS Chat: Aplicación web de chat para comunicación en tiempo real entre estudiantes y docentes.

TITLE: RS Chat: Real time chat web application for students and teachers communication.

CONVOCATORIA: Junio, 2023

RESUMEN:
El resumen reflejará las ideas principales de cada una de las partes del trabajo, pudiendo incluir un avance de los resultados obtenidos. Constará de un único párrafo y se recomienda una longitud no superior a 300 palabras. En cualquier caso, no deberá superar una página de longitud.

ABSTRACT:
Abstract will reflect the main ideas of each part of the work, including an advance of the results obtained. It will consist of a single paragraph and it is recommended a length not superior to 300 words. In any case, it should not exceed a page of length.

Palabras clave: Lorem, ipsum, dolor, sit, amet.

Firma del alumno:	**VºBº Tutor/es:**

Índice de contenidos

Índice de figuras

Índice de cuadros y tablas

Índice de bloques de código

Índice de diagramas UML

Glosario

Backend Es la parte de la aplicación que se ejecuta en el servidor.

Bot Es un programa que se ejecuta en un servidor y que puede interactuar con los usuarios.

Cluster Es un conjunto de servidores que trabajan juntos para realizar una tarea.

Docker Es una herramienta que permite crear contenedores que ejecutan servicios.

Docker Compose Es una herramienta que permite definir y ejecutar contenedores Docker.

Dockerfile Es un fichero que contiene las instrucciones para crear una imagen Docker.

Frontend Es la parte de la aplicación que se ejecuta en el navegador del usuario.

HTTP HyperText Transfer Protocol.

IDE Integrated Development Environment (Entorno de desarrollo integrado).

Instanciar Es la acción de crear un objeto en memoria principal.

JSON JavaScript Object Notation.

JVM Java Virtual Machine.

popup Es una ventana emergente que se abre en el navegador.

Script Es un programa que se ejecuta en un intérprete.

1. Introducción

1.1. MOTIVACIÓN

La idea de la aplicación surgió en el periodo final del curso 2021–2022. Se tenían diferentes opciones para realizar el backend de la aplicación. La primera opción era utilizar *NodeJS* con *Express*, tecnologías que se habían aprendido en la asignatura de *Aplicaciones Web*. Sin embargo, se decidió utilizar *Java* con *Spring Boot* debido a que se consideró que era una mejor forma de conectar con muchas más empresas en un futuro y una forma de aprender una nueva tecnología. Además, se consideró que era una tecnología más robusta y que se podía utilizar en muchos más proyectos. En cuanto a la parte frontend, se tenía claro que se utilizaría *React* debido a que se había aprendido en la misma asignatura y es más fácil de aprender y de emplear que otras tecnologías como *Angular* o *VueJS*. Este proyecto se comenzó en junio de 2022 como un proyecto personal para aprender *Spring Boot* y se dedicaba el tiempo libre a implementar todas las funcionalidades que se recogen en este documento.

1.2. OBJETIVOS

En un principio, el objetivo de la aplicación era su integración de forma completa con la plataforma de *Moodle* de la Universidad de León, para que tanto estudiantes y profesores pudieran acceder a ella. Sin embargo, debido a que era una idea demasiado ambiciosa, se ha optado por generalizarlo más a un chat de mensajes instantáneos para que cualquier persona con acceso a internet pueda utilizarlo (pero manteniendo la temática de estudiantes/profesores, en caso de llegarse a utilizar en un futuro). En cualquier chat, se permite compartir archivos, imágenes, vídeos, etc. con los demás usuarios. Se podrán tener varios chats abiertos a la vez, pudiendo cambiar entre ellos fácilmente (teniendo una disposición en pestañas, mostrando un pequeño icono con el número de mensajes que no han sido leídos todavía). Además, se podrán crear grupos de chat para que varias personas puedan comunicarse entre sí, pudiendo compartir un código de invitación para que otros usuarios se unan al grupo en específico. Estos canales podrán ser públicos o privados, pudiendo ser generados por cualquier usuario registrado en la aplicación. Los canales privados solo podrán ser vistos por los usuarios que tengan acceso a ellos (mediante invitación). Los canales públicos podrán ser vistos

por cualquier persona que tenga acceso a la aplicación. También se podrán enviar mensajes a una sola persona, pudiendo tener una conversación privada.

1.3. METODOLOGÍA DE TRABAJO

Para el desarrollo de la aplicación se ha utilizado el marco de trabajo **SCRUM**, que se emplea para la gestión de proyectos ágiles de manera **iterativa** e **incremental**. Iterativa porque se revisa y mejora el producto existente en cada ciclo de desarrollo (**sprint**), e incremental porque las nuevas funcionalidades se integran en la aplicación a medida que se completan. Todas las funcionalidades a implementar se denominan **historias de usuario** y forman lo que se denomina Pila del Producto (**Product Backlog**). En cada iteración se seleccionan las historias de usuario que se consideran más prioritarias y se asignan a los miembros del equipo de trabajo, formando la Pila del Sprint (**Sprint Backlog**).

La duración de las iteraciones en SCRUM es variable, pero se recomienda que no sea inferior a 2 semanas. En este proyecto se ha optado por una duración de 4 semanas, ya que el tiempo de desarrollo de cada iteración debe ser suficiente para que el desarrollador pueda completar todas las tareas que se le han asignado durante la misma.

Puesto que el equipo de trabajo está formado por un único miembro, no ha sido necesario realizar reuniones de coordinación, porque se ha podido trabajar de manera autónoma y sin necesidad de supervisión. Además, se ha adaptado el marco de trabajo para que se ajuste a las necesidades del proyecto.

1.4. TECNOLOGÍAS Y HERRAMIENTAS

A continuación, se presenta una breve descripción de las tecnologías y herramientas más importantes entre todas las que se han utilizado para la elaboración de este trabajo:

- Java: es el lenguaje de programación utilizado para la implementación del backend de la aplicación.
- Spring Boot: es un framework de Java que permite la creación de aplicaciones web.
- Maven: es un gestor de dependencias de Java que facilita la descarga e integración de las librerías utilizadas en la aplicación. El fichero `pom.xml` tiene una etiqueta

donde se añaden todas las dependencias a utilizar, proporcionando su nombre, versión e identificador.

- React: es una biblioteca de JavaScript que permite la creación de interfaces de usuario mediante componentes.
- Vercel: es un servicio de hosting que permite el despliegue de la parte de frontend de la aplicación web.
- Git: es un sistema de control de versiones que ha facilitado el desarrollo de la aplicación desde diferentes ordenadores.
- GitHub: es una plataforma que permite el almacenamiento de repositorios de Git.
- IntelliJ IDEA: es el IDE que ha permitido la implementación de la aplicación de forma completa. Se ha utilizado para la creación de los proyectos de frontend y backend, así como para la elaboración de este documento mediante el uso de LaTeX.

1.5. PROBLEMAS

Durante el desarrollo de la aplicación han surgido varios problemas, sobre todo en la parte del despliegue a producción de la aplicación. En un comienzo, la aplicación se ha desplegado en dos clusters con el plan gratuito de Heroku (uno para el frontend y otro para el backend), pero se ha tenido que desplegar de una manera alternativa debido a que este servicio no es gratuito desde del 28 de noviembre de 2022. Actualmente, el frontend está desplegado en Vercel y en cuanto al backend, se ha tenido una reunión con un encargado de los servicios en la nube de la empresa Platform.sh, pero no se ha llegado a ningún acuerdo para conseguir un plan gratuito para desplegar la aplicación. Debido a esto, se ha optado por desplegar el backend en un pequeño ordenador personal propio, estando siempre encendido y con una dirección IP estática. Esto no es una solución óptima, pero es la mejor que se ha encontrado hasta el momento.

2. Estado del arte

2.1. APLICACIONES SIMILARES

Se ha realizado una búsqueda de diferentes sitios web y aplicaciones que permitan la comunicación entre usuarios a través de un chat en tiempo real. Las diferentes páginas web y aplicaciones que se han encontrado se presentan a continuación, analizando sus características principales:

- **Chateagratis.net**: Esta página web permite a los usuarios elegir un apodo (que se escoge antes de iniciar las conversaciones) con el que entrar en el chat, por lo que no es necesaria la creación de una cuenta de usuario (aunque sí que se disponga de esta funcionalidad). La aplicación se divide en diferentes salas, cada una de las cuales tiene una temática diferente y se pueden mantener multiples salas abiertas al mismo tiempo. Los usuarios pueden unirse a las salas existentes en un listado, disponible al acceder a la aplicación o en la pestaña asignada para ello. La página web permite a los usuarios enviar solamente mensajes de texto (con ciertos formatos, como negrita, colores, etc.) y emoticonos, sin la posibilidad de adjuntar imágenes o archivos. Se pueden programar bots para que administren una sala y que envíen mensajes automáticos cada cierto tiempo (como el que se visualiza en las siguientes imágenes, designado con el prefijo @).

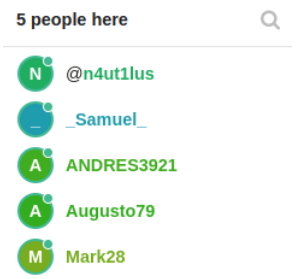

Figura 2.1: Lista de usuarios conectados en una sala de chat de chateagratis.net.

Figura 2.2: Mensajes de información del servidor en una sala de chat de chateagratis.net.

- **Dalechatea.me**: Esta página ofrece las mismas funciones que la anterior, pero con 2 diferencias: se puede modificar el tamaño del chat (ancho y alto) para adaptarlo a la pantalla y necesidades del usuario y se pueden mandar archivos multimedia temporales (imágenes, vídeos y audios), que se borran pasados 15 minutos. Esta funcionalidad solo está disponible en los chats privados, no en los públicos. Además, se pueden agregar amigos, bloquear usuarios y añadir reacciones a mensajes de otros usuarios (como mandar un saludo o añadir a marcadores), entre otras funciones.

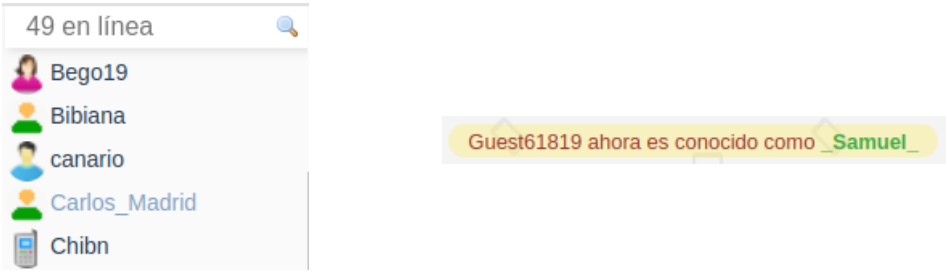

Figura 2.3: Lista de usuarios conectados y mensaje de información en una sala de chat de dalechatea.net.

- **Chatsfriends.com**: Esta página web ofrece las mismas funcionalidades que las anteriores, pero con un diseño renovado y diferente. Además, en la parte inferior (donde se escribe el mensaje) se muestra el nombre del usuario que será mostrado cuando se envíe el mensaje. A continuación se muestra una imagen de la página web, en la que se puede observar el chat en la parte central de la pantalla y un listado de usuarios en la parte derecha.

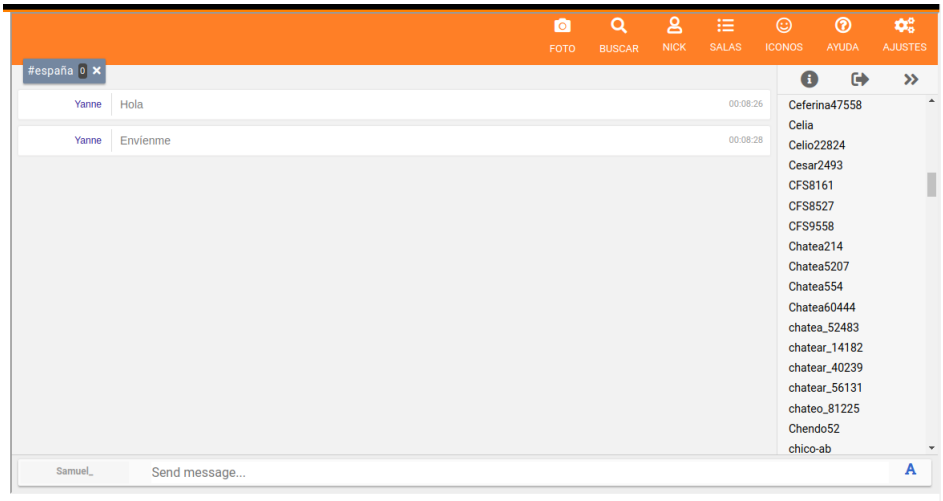

Figura 2.4: Chat de la página web de chatsfriends.com.

Todas las páginas tienen un diseño similar, con el chat en la parte central de la pantalla y un listado de usuarios en la parte derecha (ver imagen 2.5). La página web de chateagratis.net tiene un diseño más sencillo y minimalista, mientras que la de dalechatea.me tiene un diseño más moderno y con más funcionalidades. Ambos tienen un sistema de mensajes de información, que se envían por parte del servidor del chat, para notificar a todos los usuarios conectados al mismo de cualquier evento. Cuando se hace clic en un usuario de la lista, se muestra un pequeño popup con la información del usuario seleccionado, que muestra su nombre de usuario y ciertos botones para realizar las acciones previamente mencionadas.

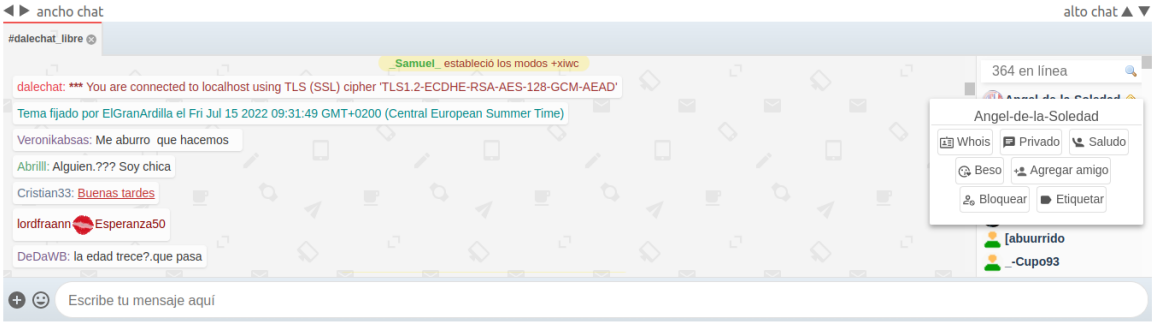

Figura 2.5: Chat completo de la página web de dalechatea.me.

2.2. OPINIÓN SOBRE EL ESTADO DEL ARTE

Haciendo una comparación entre las diferentes páginas web presentadas en la sección anterior, se observa que ninguna de ellas dispone de las siguientes funcionalidades: un sistema de notificación (auditiva) de mensajes recibidos, creación de grupos de chat personalizados y la posibilidad de enviar archivos multimedia en los chats públicos.

- **Notificación de mensajes**: En estas páginas web, cuando un usuario recibe un mensaje, no se le notifica de forma inmediata, sino que debe estar pendiente del chat para ver si aparece un mensaje nuevo. Esto puede ser molesto para los usuarios que no pueden estar pendientes de la aplicación durante todo el tiempo que están conectados. Por ello, se ha decidido implementar un sistema de notificación de mensajes, que alerte al usuario cuando recibe un mensaje, de forma que pueda continuar con sus tareas sin tener que estar pendiente de la pantalla.

- **Creación de grupos de chat**: En la mayoría de las páginas web y aplicaciones

de chat, los usuarios pueden unirse a salas de chat públicas, pero no pueden crear salas privadas. Esto puede no ser de agrado para algunos usuarios, que no pueden crear grupos de chat personalizados para comunicarse con sus amigos o familiares. Por ello, se ha decidido implementar un sistema de creación de grupos de chat, que permita a todos los usuarios crear grupos de chat personalizados para comunicarse con las personas que deseen.

- **Envío de archivos multimedia en los chats públicos**: En estas páginas web de chat, los usuarios pueden enviar distintos tipos de archivos multimedia en los chats privados, pero no en los públicos. Esto puede no agradar a todos los usuarios, ya que limita la comunicación entre los usuarios en los chats públicos. Por ello, se ha decidido implementar en todos los chats (ya sean públicos o privados) un sistema de envío de archivos multimedia.

3. Gestión del proyecto software

En este capítulo se describen los aspectos relativos a la gestión del proyecto software. En primer lugar, se describe el alcance del proyecto, que incluye los objetivos y los productos que se van a desarrollar, además de la definición del proyecto, limitaciones de uso y entregables, entre otros. A continuación, se describe la planificación del proyecto llevada a cabo, es decir, las actividades que se van a realizar y los recursos que se van a necesitar para realizarlas de manera efectiva. Después, se enumeran los recursos humanos y materiales que se van a necesitar para continuar con el proyecto, además de describir el presupuesto del mismo, incluyendo el coste que se va a necesitar para llevar a cabo el proyecto. Por último, se describen los riesgos que se van a tener que afrontar durante el desarrollo del proyecto.

3.1. ALCANCE

3.1.1. DEFINICIÓN DEL PROYECTO

El proyecto consiste en el desarrollo de una aplicación web para la comunicación mediante mensajes e intercambio de archivos entre los usuarios. Gracias a los diferentes roles de los usuarios dentro de la aplicación, se puede realizar una comunicación más fluida entre los mismos, ya que los profesores pueden crear grupos (además del ya existente para cada asignatura) para, por ejemplo, dividir a los alumnos en grupos de trabajo y así no interferir con el resto del alumnado. También cabe la posibilidad de que cada usuario pueda abrir un chat privado con cualquier otro que esté disponible en los chats a los que pertenece, abriendo una nueva conversación. Los administradores son los encargados de crear las asignaturas y los profesores, así como de gestionar los usuarios de la aplicación.

Con respecto al diseño, se ha optado por una interfaz sencilla y fácil de usar, minimalista y funcional. Se dispone de adaptabilidad a diferentes dispositivos, por lo que la aplicación es accesible desde cualquier dispositivo con conexión a Internet. Además, se ha optado por un diseño *responsive*, por lo que la aplicación se adapta a la pantalla del dispositivo en el que se esté usando, ya sea un ordenador de sobremesa, un portátil, una tablet o un teléfono móvil.

La implementación del proyecto se divide en frontend y backend. El frontend es el

encargado de mostrar las vistas y de gestionar las interacciones con el usuario, mientras que el backend es el encargado de gestionar la lógica de la aplicación, la base de datos y la comunicación con el frontend.

Debido a que la aplicación necesita que los usuarios se registren para poder utilizarla, se implementa una política de privacidad para que los usuarios conozcan cómo se van a tratar sus datos. Estos datos no se compartirán con ninguna otra entidad, salvo que el usuario lo autorice. Además, se implementa un sistema de recuperación de contraseña para que los usuarios puedan recuperar su contraseña en caso de que la hayan olvidado.

3.1.2. OBJETIVOS

Los objetivos del proyecto son los siguientes:

- Desarrollar una aplicación web para la comunicación entre los usuarios.
- Establecer un sistema de roles para los usuarios, para que puedan interactuar entre ellos de manera efectiva y fluida. Esto ayuda a los alumnos a determinar los usuarios a los que comunicar incidencias u otra información.
- Se debe ofrecer un sistema de grupos y chats individuales para los usuarios.
- Los administradores deben tener un panel de control para gestionar todo lo relacionado con la aplicación y el uso de recursos por parte del servidor, para evitar sobrecargas o caídas.
- Garantizar el debido cumplimiento de la política de privacidad de los usuarios.

3.1.3. LIMITACIONES DE USO

Para garantizar el correcto funcionamiento de la aplicación, se establecen las siguientes restricciones de uso:

- Se garantiza una buena visualización de la aplicación en dispositivos de cualquier resolución, tanto dispositivos móviles como de escritorio, siempre que dispongan de conexión a Internet y un navegador web actualizado.

- El funcionamiento del sistema se ha probado para los navegadores basados en el motor de Chromium, como Google Chrome y Brave. No se garantiza el correcto funcionamiento en otros navegadores.
- No se requiere instalación de ningún programa externo. De esta manera, se simplifica el acceso a la aplicación.
- Para el almacenamiento de archivos, se utilizará un sistema de almacenamiento en la nube, siguiendo una estructura de directorios dependiendo del tipo de archivo que los usuarios suban.

3.1.4. ENTREGABLES

3.1.5. CRITERIOS DE ACEPTACIÓN

Para que el proyecto se considere aceptado, se deben cumplir una serie de requisitos, que son los siguientes:

- Los puntos mencionados en el apartado 3.1.2 deben estar implementados.
- Con el fin de garantizar la ausencia de errores, se debe realizar una serie de pruebas por parte del equipo de desarrollo y también por parte de los usuarios. En caso de existir algún error, se debe corregir antes de que el proyecto sea aceptado.
- Los casos de uso deben estar implementados y funcionando correctamente.
- Los test unitarios, de aceptación e integración deben pasar sin errores.

3.1.6. RESTRICCIONES

3.2. PLANIFICACIÓN

3.3. GESTIÓN DE RECURSOS

En este capítulo se describen los recursos humanos y materiales que se van a necesitar para el desarrollo del proyecto.

3.3.1. ESPECIFICACIÓN DE RECURSOS

Cuando se habla de recursos humanos, se hace referencia a las personas que van a participar en el proyecto, y a los recursos materiales, que son los equipos y herramientas que se van a utilizar para llevar a cabo el proyecto. En el caso de este, que ha sido implementado enteramente por una persona, se simplifican los cálculos de los recursos. A continuación, se presenta una tabla con la especificación de los recursos para cada rol que se ha desempeñado en el desarrollo del proyecto:

Recursos	**Anual**	**Mensual**	**Diario**	**hora**
Desarrollador backend	30,554€	2,182.42€	109.12€	13.64€
Desarrollador frontend	34,486€	2,463.28€	123.16€	15.39€
Diseñador	22,150€	1,582.14€	79.10€	9.88€
Tester	29,406€	2,100.42€	105.02€	13.13€
Técnico de sistemas	23,729€	1,694.92€	84.74€	10.59€

Cuadro 3.1: Especificación de recursos

Se ha considerado que la jornada laboral es de 8 horas diarias, 5 días a la semana, y 14 pagas al año. Los datos de la columna *Salario anual* se han obtenido del buscador de sueldos de *Indeed* [1] a fecha 5 de febrero de 2023. Esta web se encarga de calcular el sueldo promedio de los trabajos ofrecidos en España para un puesto en concreto. El resto de datos se han calculado con las siguientes fórmulas:

$$\text{Salario mensual} = \frac{\text{Salario anual}}{14}$$
$$\text{Salario diario} = \frac{\text{Salario mensual}}{20}$$

Una estimación del trabajo que se ha realizado con el coste asociado se puede ver en la tabla 3.2. La jornada laboral se ha considerado de 8 horas diarias, 5 días a la semana, y 14 pagas al año.

Recursos	**Trabajo (horas)**	**Coste (€)**
Desarrollador backend	320	4,364.8 €
Desarrollador frontend	120	1,846.8 €
Diseñador	10	98.8 €
Tester	30	393.9 €
Técnico de sistemas	20	211.8 €
Totales	500	6,916.1 €

Cuadro 3.2: Coste de los recursos

3.4. PRESUPUESTO

3.5. GESTIÓN DE RIESGOS

4. Contenido

4.1. PATRONES DE DISEÑO

Un patrón de diseño es una solución que se puede aplicar a diferentes contextos y que se puede reutilizar en diferentes proyectos.
Hay varias categorías de patrones de diseño, cada una con una finalidad diferente [2]:

- **Patrones de creación**: se utilizan para crear objetos de una forma flexible y reutilizando código existente.
- **Patrones estructurales**: se utilizan para convertir clases y objetos en estructuras más complejas.
- **Patrones de comportamiento**: se utilizan para definir la interacción entre objetos.

En este proyecto se han utilizado varios patrones de diseño, para permitir una mejor escalabilidad, mantenibilidad y reutilización del código. A continuación se detalla la siguiente información de los patrones utilizados:

- Definición / categoría.
- Explicación de cómo se ha implementado en el proyecto.
- Diagrama UML.
- Justificación de su uso en la aplicación.

4.1.1. BUILDER

Es un patrón de **creación** que permite instanciar objetos complejos de una forma sencilla. Se ha creado una interfaz (`Builder`) genérica para su reutilización en caso de ser necesaria para cualquier otra clase. Esta interfaz define el método `build()` que devolverá la instancia de un objeto con los datos establecidos previamente. El diagrama UML del patrón implementado es el siguiente:

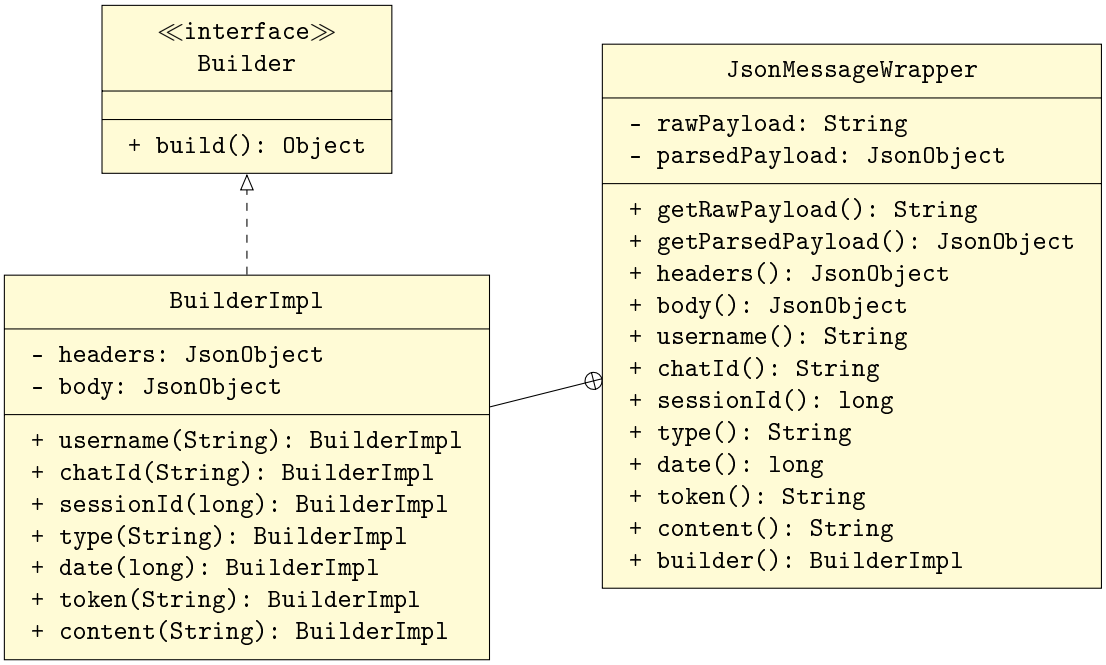

Diagrama UML 4.1: Patrón Builder empleado en la aplicación.

El patrón builder se ha usado en la aplicación para simplificar la creación de objetos de tipo `JsonMessageWrapper`, por el momento. Esta clase es la encargada de encapsular los mensajes que se envían a través de la red en formato JSON.

4.1.2. SINGLETON

También es un patrón de **creación** y se emplea para garantizar que una clase concreta tenga una única instancia y proporciona un punto de acceso global a ella [3]. En el contexto de esta aplicación, se utiliza en ciertas clases de utilidad y en las clases que asocian rutas a un método HTTP (por ejemplo `/login` con el método POST). Estas últimas son clases internas de `Routes.java` y los nombres dependen del método HTTP que se debe utilizar para realizar una petición a una ruta específica.

Cuadro 4.1: Relación entre método HTTP y ruta.

Método HTTP	Clase de la ruta
GET	`GetRoute`
POST	`PostRoute`
PUT	`PutRoute`
DELETE	`DeleteRoute`

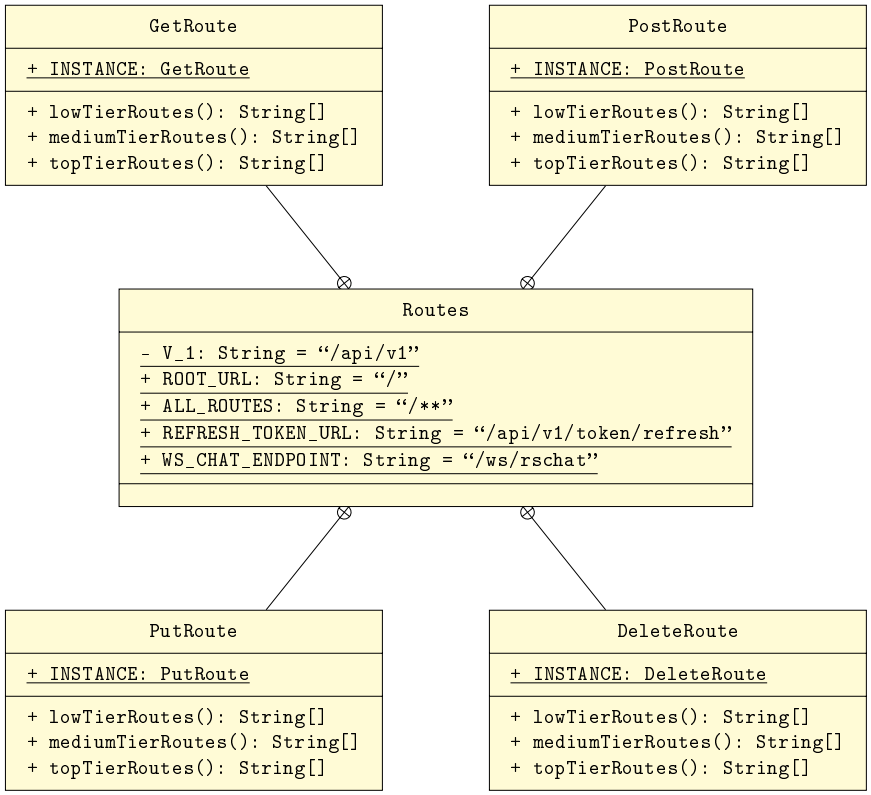

Diagrama UML 4.2: Patrón Singleton empleado en la aplicación.

Se han utilizado diferentes formas de acceso a la instancia de las clases. En el caso de las clases que asocian rutas a métodos HTTP, el modificador de acceso a la instancia es público. En otros casos, se provee un método estático para obtener la instancia de la clase.

4.1.3. STRATEGY

Este patrón de **comportamiento** se utiliza para encapsular un algoritmo dentro de una clase, de forma que pueda ser intercambiado por otro algoritmo en tiempo de ejecución. En la aplicación, se emplea para encapsular el proceso de manejo de los mensajes que se envían entre los usuarios. Existen varias clases que se encargan de realizarlo, por lo que todas ellas implementan la interfaz `MessageStrategy`, que define el método `handle(...)`, encargado de realizar el procesamiento del mensaje y recibe 3 parámetros:

- El mensaje que se debe manejar.
- La lista con todos los chats para determinar al que se debe enviar el mensaje.

- Otros datos que pueden ser necesarios para el manejo del mensaje. Dependiendo de la clase, este parámetro puede contener más o menos datos.

La relación entre el tipo de mensaje y las clases encargadas de procesarlo, que implementan esta interfaz, son las siguientes:

Cuadro 4.2: Relación Mensaje - Estrategia

Mensaje	**Clase de la estrategia**
`USER_JOINED`	`UserJoinedStrategy`
`USER_LEFT`	`UserLeftStrategy`
`TEXT_MESSAGE`	`TextMessageStrategy`
`IMAGE_MESSAGE`	`ImageMessageStrategy`
`AUDIO_MESSAGE`	`AudioMessageStrategy`
`VIDEO_MESSAGE`	`VideoMessageStrategy`
`ACTIVE_USERS_MESSAGE`	`ActiveUsersStrategy`
`GET_HISTORY_MESSAGE`	`GetHistoryStrategy`
`ERROR_MESSAGE`	`ErrorMessageStrategy`
`PING_MESSAGE`	`PingStrategy`

A continuación se muestra el diagrama UML de la interfaz `MessageStrategy`:

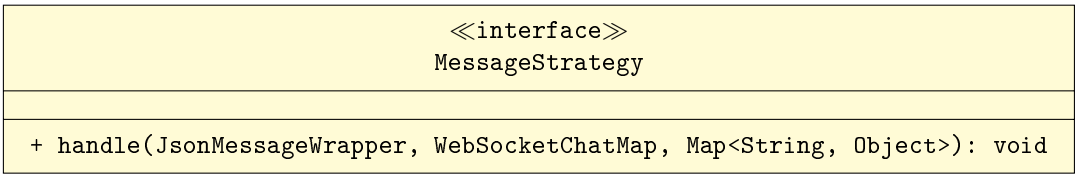

Diagrama UML 4.3: Interfaz MessageStrategy.

Este patrón se ha utilizado para simplificar el manejo de los mensajes recibidos en el servidor y para que sea más fácil de extender. En el caso de que se desee agregar un nuevo tipo de mensaje, se debe crear una nueva clase que implemente la interfaz `MessageStrategy` y agregarla a la lista de estrategias que se encuentran en la clase `WebSocketHandler`. Esta clase es la encargada de determinar qué estrategia se debe utilizar para manejar el mensaje. Para esto, se utiliza el método `decideStrategy(receivedMessageType: WSMessage)` que recibe como parámetro el tipo mensaje y devuelve la estrategia que se debe utilizar para manejar el mensaje.

4.2. PROCESAMIENTO DE LOS MENSAJES

Como hemos visto en la sección anterior, los mensajes que se reciben en el servidor pueden ser de diferentes tipos, cambiando la forma en que se procesan. A continuación se muestra una lista con las acciones que se realizan para cada tipo de mensaje que se trata:

- **Text, Image, Video y Audio**: se envían al resto de usuarios del chat en el mismo formato en que llegaron al servidor. Estos cuatro tipos de mensaje contienen texto exclusivamente, siendo un mensaje o el enlace a un archivo almacenado en el bucket de S3.
- **ActiveUsers**: se envía solo al cliente que lo ha solicitado, y contiene una lista con los usuarios que están conectados en ese momento al chat, ordenados alfabéticamente.
- **GetHistory**: se envía al usuario que lo solicita, y contiene una lista con los últimos 65 mensajes que se han enviado al chat. Se realiza una lectura del fichero de texto que contiene el historial de mensajes (almacenado en disco) y un filtrado de los mensajes de actividad (los 2 siguientes) del solicitante, ya que no son relevantes.
- **UserJoined y UserLeft**: se notifica a las personas conectadas el nombre del usuario que se ha unido o ha salido del chat,
- **Ping**: se envía un mensaje con un breve texto. Se utiliza exclusivamente para mantener la conexión WebSocket abierta.

4.3. CICLO DE VIDA DE LA CONEXIÓN DE USUARIOS

Cuando un usuario accede a un chat de la aplicación, se inicia una conexión entre el cliente y el servidor a través del protocolo de comunicación bidireccional **WebSocket** [4]. Esta conexión se mantiene abierta mientras el usuario esté en el chat, y se cierra cuando el usuario lo abandona. La secuencia de eventos que ocurren durante la conexión de un usuario al chat es la siguiente:

4.3.1. FRONTEND

1. Se realiza una solicitud de conexión WebSocket al servidor.
2. Se realiza una petición HTTP para comprobar que el usuario puede acceder al chat. Esto se realiza para que, en caso de que el usuario introduzca la URL de un chat al que no tiene acceso de forma manual, se le redirija a la página de inicio de la aplicación.
3. Si se confirma que el usuario **puede acceder** al chat:
   - 3.1. Se establece la conexión WebSocket.
   - 3.2. Se envía un mensaje de tipo `USER_JOINED` al servidor.
   - 3.3. Se consultan los últimos mensajes del historial de mensajes del chat con el mensaje de tipo `GET_HISTORY_MESSAGE`.
   - 3.4. Se realiza una petición de la lista de usuarios activos con un mensaje de tipo `ACTIVE_USERS_MESSAGE`.
   - 3.5. Se configura un temporizador para mandar un mensaje de tipo `PING_MESSAGE` cada 30 segundos. Esto se realiza para que el servidor no cierre la conexión por inactividad.
4. Si el usuario **no puede acceder** al chat:
   - 4.1. Se cierra la conexión WebSocket, en caso de llegarse a abrir.
   - 4.2. Se redirige al usuario a la página principal.

4.3.2. BACKEND

Cuando comienza la ejecución del programa, se indica a Spring Boot que el manejador de mensajes a través de WebSocket del servidor es una instancia de la clase `WebSocketHandler`, en la ruta '/ws/rschat'. Al instanciar esta clase, se crea un objeto **chatMap** de tipo `WebSocketChatMap`, que contiene el atributo `chats` (ver Diagrama UML 4.4). Es una tabla de dispersión que contiene los chats activos en la aplicación. Cada entrada asocia a una cadena de texto (identificador del chat) la instancia de un objeto de tipo `Chat`. El proceso de **conexión** al servidor sigue el siguiente flujo de eventos:

1. Se establece la conexión WebSocket con el usuario. Esto ocurre de forma transparente al programador debido a que la implementación se realiza en el framework de Spring Boot.

2. Se recibe el mensaje `USER_JOINED` del cliente y se crea un objeto de tipo `WSClient` (formado por la instancia de `WebSocketSession` y el `WSClientID` del usuario). Este nuevo objeto se añade a la lista de usuarios conectados al chat.

    2.1. Si el usuario es el primero que se ha conectado al chat, se crea uno nuevo, guardándose en la lista de chats.

    2.2. Si no, se añade al chat correspondiente de la lista de chats.

3. Se recibe el mensaje `GET_HISTORY_MESSAGE` del cliente y se envían como respuesta los últimos 65 mensajes del historial de mensajes del chat.

4. Se recibe el mensaje `ACTIVE_USERS_MESSAGE` del cliente y se devuelve la lista con los usuarios activos en el chat.

Y el proceso de **desconexión** se realiza como sigue:

1. Se recibe el mensaje `USER_LEFT` del cliente.

2. Se informa al resto de los usuarios del chat de la desconexión del usuario.

3. Al eliminar un usuario del chat se pueden dar 2 casos:

    3.1. Si el usuario es el último que se ha desconectado del chat, se elimina el chat de la tabla de dispersión `chats`. Cuando esto ocurre, el historial de mensajes del chat que se haya registrado desde que se inició, se envía al almacenamiento en la nube.

    3.2. Si no, se elimina el usuario de la lista de usuarios del chat.

4. Se cierra la conexión WebSocket con el usuario, de forma transparente al programador (al igual que la conexión).

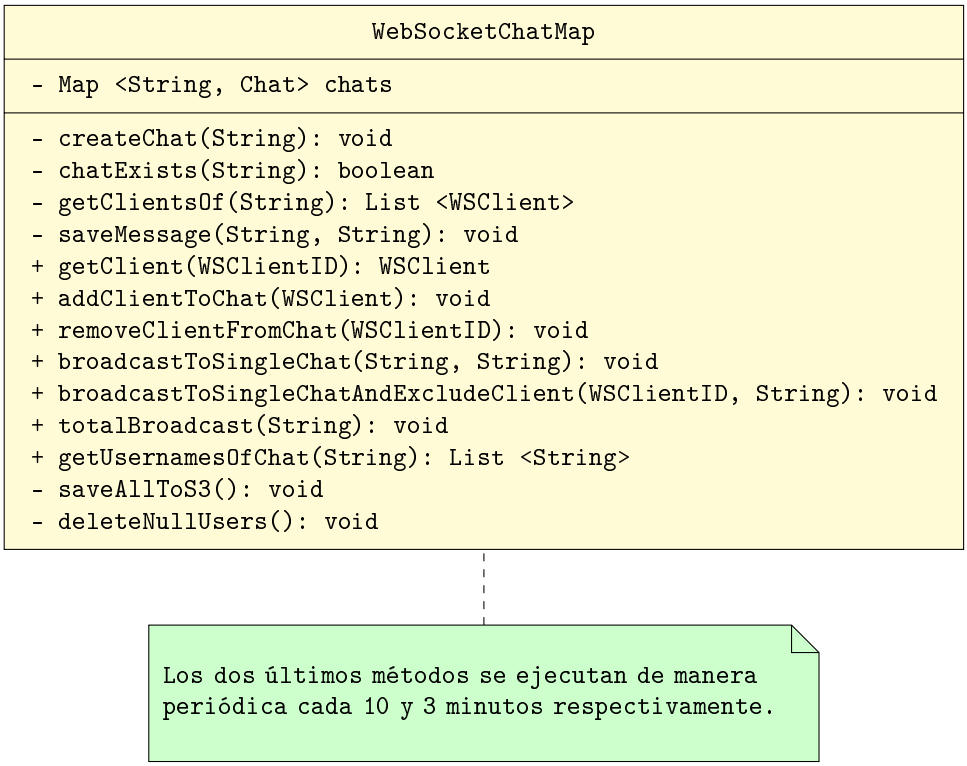

Diagrama UML 4.4: Clase `Chat` para almacenar los usuarios activos

4.4. DOCKER

Docker es una plataforma software que permite el desarrollo, prueba y ejecución de aplicaciones de forma rápida y cómoda, separando la aplicación de la infraestructura permitiendo ejecutar múltiples aplicaciones en un mismo servidor (ver imagen 4.1). Esto se realiza empaquetando la aplicación, sus dependencias y otras herramientas necesarias para su ejecución en lo que se denominan **contenedores**, que son instancias ejecutables de una imagen de Docker. Las imágenes son ficheros de solo lectura que contienen las instrucciones necesarias para crear un contenedor y normalmente se basan en otras imágenes añadiendo o modificando configuraciones [5].

En el desarrollo de la aplicación se ha utilizado Docker para la creación de una imagen que contenga todo el código de la aplicación así como las variables de entorno necesarias para su correcto funcionamiento. Se ha utilizado la herramienta `docker-compose`, que permite definir y ejecutar contenedores Docker de manera sencilla. Para la configuración de los contenedores, se dispone del fichero `docker-compose.yaml` que se encuentra en la raíz del proyecto y define los servicios que se ejecutarán en los contenedores, así como las dependencias entre ellos.

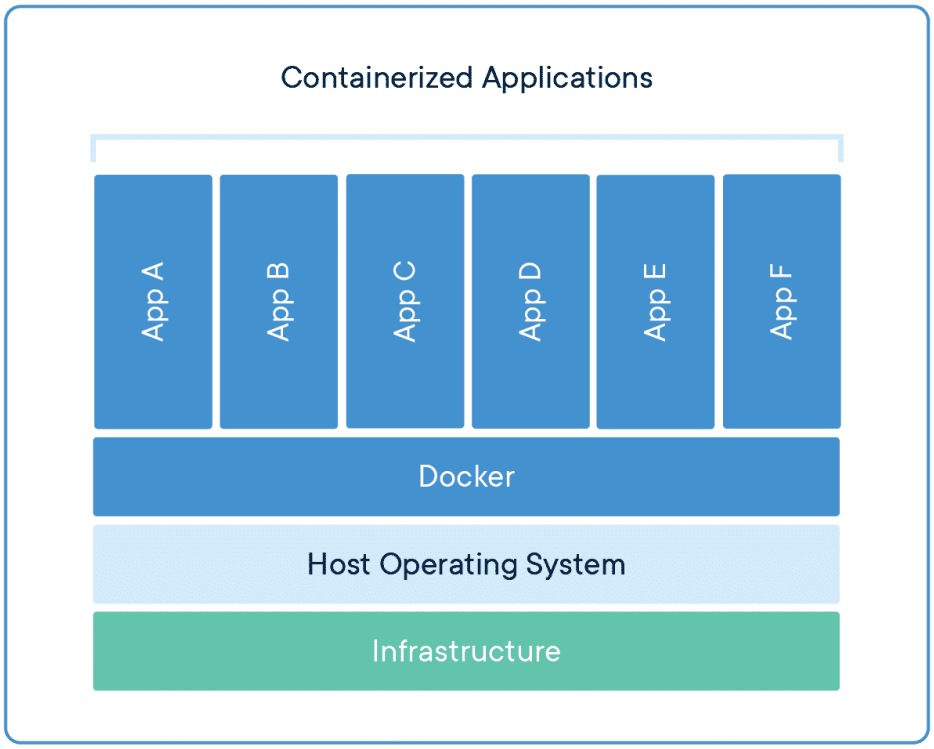

Figura 4.1: Infrastructura docker. Fuente: `https://www.docker.com/resources/what-container`

Debido a que cada vez que se elimina un contenedor (al reiniciar el sistema o al generar de nuevo la imagen de la aplicación para incluir cambios del código) se pierden los datos, la persistencia de los mismos y de sus configuraciones es necesaria. Para lograr esto, se usan **volúmenes**, que son directorios creados y gestionados por Docker que se almacenan en el sistema de archivos del host, y que se montan en los contenedores cuando se inicializan [6]. De esta forma, los datos no se pierden al eliminar el contenedor. Los volúmenes se definen en el mismo fichero `docker-compose.yaml` mediante la opción `volumes` y son los siguientes:

- `rschat-db`: contiene los datos y script inicial de la base de datos.
- `rschat-logs`: contiene los ficheros de registro de la aplicación.
- `grafana-storage`: contiene la configuración de Grafana.

A continuación, se mencionan todos los contenedores utilizados para la aplicación junto con una breve descripción de cada uno de ellos:

Nota: los marcados con el símbolo * se describirán en detalle más adelante.

- `rschat`: Contenedor que ejecuta el backend de la aplicación.
- `rschat-db`: Contenedor que ejecuta la base de datos MySQL.
- `prometheus`*: Contenedor que ejecuta el servidor de métricas de Prometheus.
- `grafana`*: Contenedor que ejecuta el panel de observabilidad de Grafana.
- `loki`*: Contenedor que ejecuta el servidor de logs Loki.
- `promtail`*: Contenedor que ejecuta el agente de logs Promtail.

4.4.1. PROMETHEUS (MÉTRICAS)

Prometheus es un conjunto de herramientas de código abierto para la **monitorización** de sistemas y **alertas** que recoge y almacena las métricas con la marca de tiempo cuando se producen, incluyendo etiquetas (clave-valor) de manera opcional [7]. Estas métricas se utilizan para determinar el funcionamiento y estado de la aplicación en tiempo real, permitiendo diagnosticar problemas de rendimiento o recursos de una manera rápida y sencilla. Para habilitar la exportación de las métricas (de manera automática) por parte de la aplicación hay que realizar ciertas configuraciones, que se detallan a continuación:

- Añadir 2 librerías en el fichero `pom.xml` [8]:
  - spring-boot-starter-actuator
  - micrometer-registry-prometheus
- Para exponer el 'endpoint' que permite ver las métricas, hay que añadir una propiedad en el fichero `application.properties`:
  - `management.endpoints.web.exposure.include=prometheus`
- Permitir las peticiones de tipo GET a `/actuator/prometheus`: esto se configura de manera programática estableciendo la ruta como pública (permitiendo peticiones GET sin necesidad de estar autenticado).

Las métricas que más se utilizarán son las relacionadas con el rendimiento de la aplicación y el uso de recursos del sistema, aunque se han añadido algunas personalizadas para determinar el uso que se hace de determinadas partes de la aplicación. La lista con las métricas más usadas es la siguiente:

- jvm_memory_used_bytes: bytes usados por la JVM.
- jvm_memory_committed_bytes: bytes que se han reservado para su uso por la JVM.
- system_cpu_usage: uso de CPU del sistema donde se ejecuta la aplicación.
- process_cpu_usage: uso de CPU del proceso de la JVM.
- system_cpu_count: número de procesadores disponibles para la JVM.
- logback_events_total: número de eventos de log de un determinado nivel (info, debug, warn, error).
- http_server_requests_seconds_count: número de peticiones HTTP realizadas a la aplicación.
- jvm_threads_live_threads: número total de hilos en ejecución.
- jvm_threads_daemon_threads: número de hilos en ejecución (en segundo plano).
- jvm_threads_peak_threads: número máximo de hilos activos desde que se inició la JVM.

4.4.2. GRAFANA (PANEL DE OBSERVABILIDAD)

Grafana es una solución de código abierto para mostrar las métricas que se recogen de las aplicaciones de una manera amigable en paneles personalizables [9]. Permite estudiar, analizar y monitorizar aplicaciones a lo largo del tiempo gracias a las marcas de tiempo proporcionadas junto con los datos que se recolectan. Se puede conectar con muchas fuentes de datos, como Graphite, Prometheus, ElasticSearch, MySQL, etc. Una ventaja de Grafana es que ofrece una solución 'on-premise', que permite desplegar una instancia propia en el mismo servidor donde se ejecuta la aplicación. De esta manera,

se garantiza la seguridad y protección de los datos, ya que no se exponen a Internet de manera directa.

Para la configuración de Grafana en el entorno de producción, se necesita un contenedor Docker que utilice la imagen `grafana/grafana-oss` y exponga un puerto para permitir conexiones (el 4046 en este caso). A continuación, se presenta una configuración básica para ejecutar una instancia de Grafana:

```yaml
version: '3.7'
services:
  grafana:
    image: grafana/grafana-oss:9.3.1
    ports:
      - "4046:3000"  # Mapeo de puertos (HOST:CONTENEDOR)
    volumes:
      - grafana-storage:/var/lib/grafana  # Se define el volumen 'grafana-storage'
    env_file:
      - GF_SECURITY_ADMIN_USER=<usuario>
      - GF_SECURITY_ADMIN_PASSWORD=<contraseña>
    networks:
      - rschat-net  # Se asigna la red interna para comunicarse con otros servicios
```

Código 4.1: Configuración mínima para ejecutar un contenedor con Grafana

En este proyecto se ha utilizado un panel importado desde las plantillas de la comunidad de Grafana para su utilización con Spring Boot y Prometheus [10] al que se le han añadido más paneles (para los logs y otras métricas). Algunos de los tipos de paneles que se pueden añadir son de tipo numérico o texto, gráficas, histogramas, mapas de calor, tablas, entre otros [11].

Figura 4.2: Visualización de métricas relacionadas con los recursos del sistema utilizados y tiempo de actividad

Figura 4.3: Visualización de métricas relacionadas con los logs y peticiones HTTP

Para que Grafana muestre toda esta información, se deben configurar 2 fuentes de datos de las cuales extraer las métricas. Para establecer estos ajustes, accedemos a `Configuration > Data Sources` y añadimos un 'data source' para Prometheus y otro para Loki, con la URL que provee las métricas (el nombre del contenedor con el puerto interno, ya que se usa la misma red para todos los servicios):

- Prometheus: `http://prometheus:9090`
- Loki: `http://loki:3100`

4.4.3. LOKI (AGREGACIÓN DE LOGS)

Loki es una herramienta de agregación de logs, que permite almacenar, indexar y consultar logs de manera eficiente, ya que se basa en el uso de etiquetas (para el agrupamiento y filtrado de logs) dejando el mensaje original sin modificar. Para el funcionamiento de Loki es necesario disponer de un agente que se encargue de enviar los logs a la instancia de Loki, y en este proyecto se ha utilizado Promtail, explicado en la siguiente sección. La configuración necesaria para que Grafana pueda recoger los logs de Loki es sencilla, ya que se añade un fichero `yaml` con la configuración de la fuente de datos de Loki [12] y el servicio en el fichero `docker-compose.yaml`:

```yaml
loki:
  image: grafana/loki:2.7.0
  container_name: loki
  ports:
    - "4045:3100"
  volumes:
    # Habilitar la lectura del fichero de configuración desde el contenedor
    - ./config/loki.yaml:/etc/loki/loki-config.yaml
  command: -config.file=/etc/loki/loki-config.yaml
  depends_on:
    - promtail
  networks:
    - rschat-net
```

Código 4.2: Servicio de Loki para el registro de logs

4.4.4. PROMTAIL (AGENTE DE LOGS)

Promtail es un agente (o cliente) de logs, que los recoge y los convierte en flujos que puede enviar a Loki (ver imagen 4.4) a través de una API HTTP [13]. La configuración de docker para Promtail es similar a la de Loki, ya que se añade un fichero `yaml` con la configuración de la fuente de datos de Promtail y el servicio en el fichero `docker-compose.yaml`:

```yaml
promtail:
  image: grafana/promtail:2.7.0
  container_name: promtail
  restart: unless-stopped
  ports:
    - "4044:9080"
  volumes:
    - /var/log:/var/log
    # Habilitar la lectura del fichero de configuración desde el contenedor
    - ./config/promtail.yaml:/etc/promtail/promtail.yaml
    # Habilitar acceso a los logs de los contenedores
    - /var/lib/docker/containers:/var/lib/docker/containers
  command: -config.file=/etc/promtail/promtail.yaml
  depends_on:
    - rschat
  networks:
    - rschat-net
```

Código 4.3: Servicio de Promtail para la recolección de logs

El formato de las trazas que exporta Promtail se ha ajustado para que se permita la visualización tanto por fichero, como por el nombre del contenedor que genera el

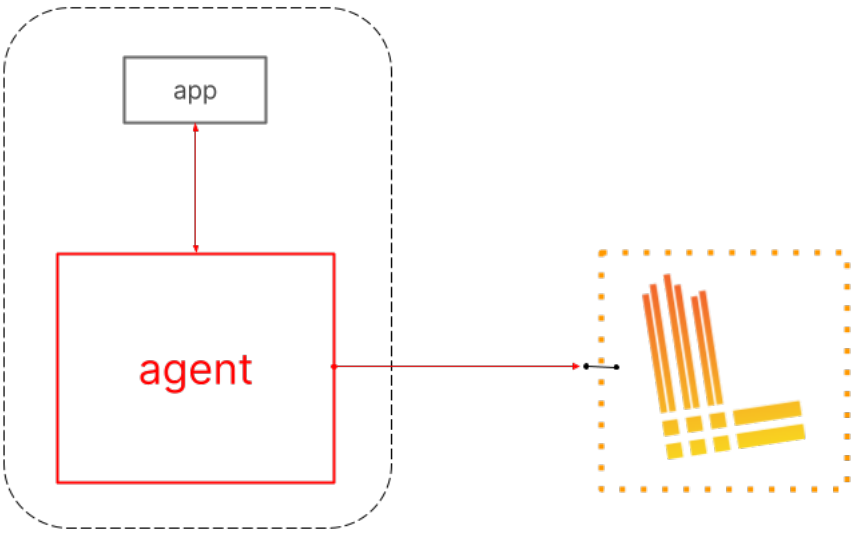

Figura 4.4: Flujo de logs entre Promtail y Loki

log. Para ello, el fichero `promtail.yaml` contiene una sección `pipeline_stages` con la configuración de los 'stages' que se aplican a las trazas antes de enviarlas a Loki. La más importante es la que se encarga de añadir las etiquetas `container_name` con una expresión regular que extrae el nombre del contenedor a partir de una etiqueta asociada a cada traza:

```
pipeline_stages:
  # ...
  - regex:  # Nombre del contenedor en el valor de "tag"
      expression: (?P<container_name>(?:[^|]*[^|]))
      source: tag
  # ...
```

Código 4.4: Stage para obtener el nombre del contenedor con una expresión regular a partir de la etiqueta `tag`

4.5. BASE DE DATOS

El sistema gestor de base de datos utilizado para la aplicación es `MySQL`. Las bases de datos de MySQL son relacionales, lo que significa que los datos se almacenan en tablas, que están formadas por filas (campos) y columnas (registros). Las tablas se relacionan entre sí mediante claves primarias y claves foráneas, que son campos que identifican a cada registro de la tabla. Este tipo de base de datos es muy empleado en aplicaciones web, por lo que es sencillo encontrar información sobre cómo usar MySQL. Como el

lenguaje de programación con el que se ha desarrollado el backend de la aplicación es Java, se ha utilizado el ORM `Hibernate`, que permite realizar un mapeado de la base de datos a objetos de Java, de forma que se puede trabajar con ella de una forma transparente y cómoda para el programador. Para realizar las operaciones de inserción, actualización, consulta y borrado de los datos, `Spring Boot Data JPA` es la mejor opción, ya que se encarga de realizar estas operaciones de forma transparente para el desarrollador, dependiendo del nombre que reciba un método. También, mediante el uso de anotaciones (expresiones precedidas por el símbolo @, como por ejemplo `@Query`), se pueden personalizar de las consultas que se ejecutarán en base de datos. El diagrama de las tablas de la base de datos de la aplicación se muestra en la siguiente figura:

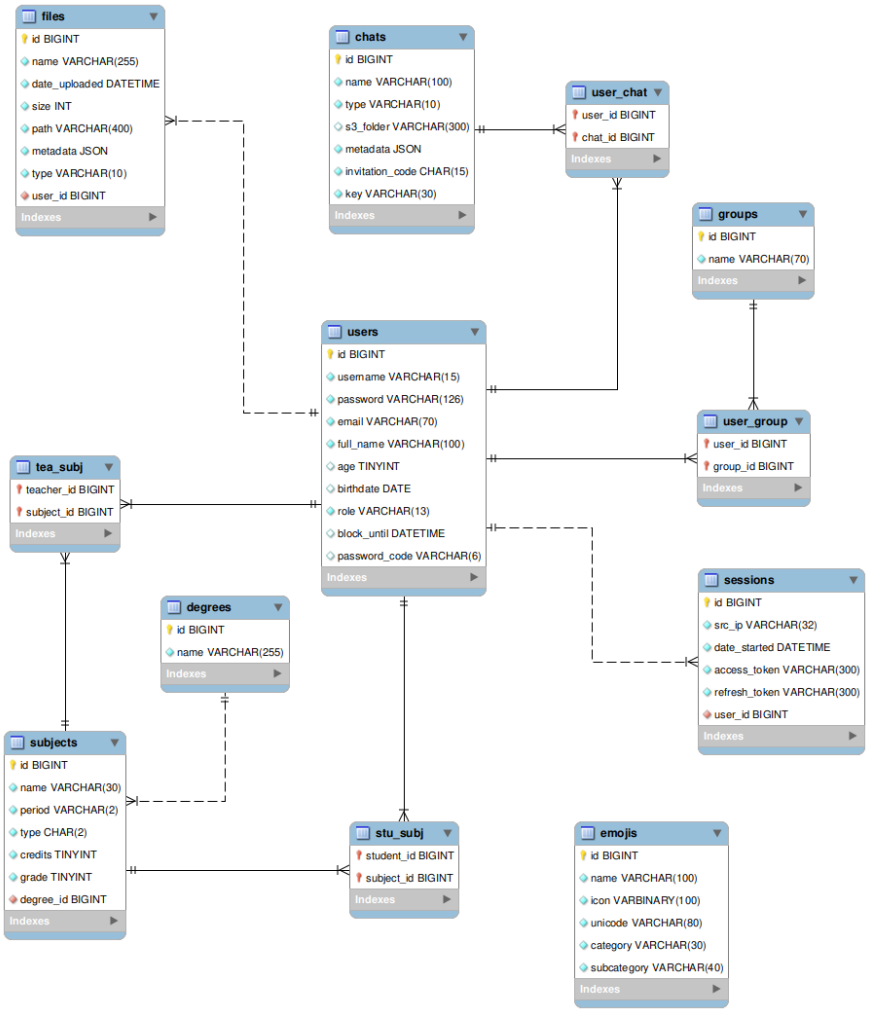

Figura 4.5: Diagrama de las tablas de la base de datos

4.6. SEGURIDAD

Para la prevención de ciberataques al servidor por parte de usuarios malintencionados, se han establecido las medidas de seguridad que se detallan a continuación:

- **Cifrado de la comunicación entre el servidor y los clientes**: Para evitar que los datos de los usuarios puedan ser interceptados por terceros, se ha establecido un cifrado de las comunicaciones. Para ello, se ha empleado el protocolo TLS, que permite cifrar la comunicación utilizando el certificado SSL que proporciona Let's Encrypt. Este certificado se ha generado automáticamente mediante su servicio gratuito de certificados y, utilizando la aplicación `Certbot`, se renueva automáticamente cada 90 días. El certificado se almacena en el servidor en el directorio `/etc/letsencrypt/live/`.

- **Configuración del servidor ssh**: Para evitar que los usuarios puedan acceder al servidor mediante ssh, se ha configurado para que únicamente se pueda acceder mediante claves ssh (no permitiendo usuario y contraseña). Estas claves se obtienen mediante el comando `ssh-keygen`, que genera un par de claves (pública y privada), y se ha copiado la clave pública del cliente en el fichero `/home/user/.ssh/authorized_keys` del servidor. De esta forma, únicamente se puede acceder al servidor mediante la clave privada del cliente.

- **Configuración del servidor web**: Se ha limitado la redirección de puertos desde el router hasta el servidor web, de forma que únicamente se puede acceder al servidor desde el puerto 22 (ssh), 4040 (aplicación en entorno de producción). Además, se ha configurado el servidor web para que únicamente se pueda acceder mediante el protocolo https, y no mediante el protocolo http, haciendo que los usuarios sean redirigidos automáticamente a https.

- **Configuración de la base de datos**: La base de datos solo es accesible desde el propio servidor web, por lo que el acceso está restringido a la red interna.

4.7. PRUEBAS

4.8. PROBLEMAS EN EL DESARROLLO

Uno de los problemas más significativos que han surgido en el desarrollo de la aplicación se ha dado en la parte del despliegue en producción de la misma. En un comienzo, la aplicación se había desplegado en dos clusters con el plan gratuito de Heroku (uno para el frontend y otro para el backend), pero en el momento en que esta plataforma publicó que se dejaría de dar soporte a las aplicaciones gratuitas, se empezaron a buscar alternativas para alojar la aplicación.

4.8.1. ALTERNATIVAS A HEROKU

Las opciones que se consideraron fueron las que se describen a continuación, ordenadas según la fecha de consideración.

- Para la parte de **frontend**, la elección fue muy sencilla. **Vercel** fue la primera opción a considerar, siendo una plataforma muy conocida por su facilidad para desplegar aplicaciones web. Cada vez que se realiza un cambio en el código fuente de la aplicación, se inicia un despliegue automático en la plataforma (esto ocurre si se tiene enlazado el repositorio de Git).

- Para la parte de **backend**, se consideraron las siguientes opciones:

  - **Alojamiento en la nube**: se consideró la posibilidad de alojar la aplicación en la nube utilizando otros servicios. Se llegó a concretar una reunión con un *Cloud Consultant* de **Platform.sh**, que fue la primera opción a considerar, pero no se llegó a ningún acuerdo para conseguir un plan gratuito para desplegar la aplicación, por tanto, se descartó esta opción.

  - **Raspberry Pi 4**: es un ordenador de tamaño muy reducido, que se puede configurar para que actúe como un servidor. Es una buena opción en caso de se desee un consumo de energía bajo. Sin embargo, no es una opción viable para el proyecto a largo plazo, ya que el almacenamiento se realiza en una tarjeta de memoria, cuya velocidad de lectura/escritura es más baja en comparación con los discos duros sólidos de los ordenadores convencionales.

- **Servidor propio**: es la opción más viable, ya que se puede aprovechar para realizar otras tareas como almacenamiento de gran cantidad datos o desplegar otras aplicaciones más potentes que con la Raspberry. Esta ha sido la **opción elegida** finalmente, ya que se ha encontrado un pequeño ordenador que cumple con los requisitos necesarios para alojar la aplicación.

4.9. PREPARACIÓN DEL SERVIDOR

Cuando se ha recibido el ordenador, se han seguido una serie de pasos para configurarlo de la manera más segura posible, ya que se va a utilizar para alojar el backend una aplicación web. Estos pasos se describen a continuación.

4.9.1. INSTALACIÓN DEL SISTEMA OPERATIVO

El sistema operativo que se ha instalado en el ordenador es **Ubuntu Server 20.04 LTS**. Se ha escogido esta distribución de Linux debido a que es una de las más populares y con una gran comunidad de usuarios, lo que hace que sea fácil encontrar información sobre cómo configurar el sistema. Además, se ha elegido la versión LTS para tener soporte durante un periodo de tiempo más largo (en el caso de esta versión, hasta 2025 y con actualizaciones de seguridad hasta 2030). Se han instalado todos los paquetes necesarios para administrarlo de una manera más cómoda y para garantizar la seguridad e integridad del sistema. Algunos de estos paquetes son los siguientes:

- **OpenSSH**: el protocolo **ssh** permite establecer conexiones seguras entre dos ordenadores. Se ha configurado esta herramienta para que se pueda acceder a él mediante un par de claves (pública/privada), eliminando la posibilidad de acceder mediante una contraseña. Esto último se realiza para evitar que se pueda establecer una conexión con él de forma no autorizada. De esta manera, los únicos usuarios que pueden acceder al servidor son aquellos que tienen la clave privada asociada a la clave pública registrada en el servidor. Para generar estas claves, se ha utilizado el comando **ssh-keygen** (en el cliente) y se ha guardado la clave pública en el archivo **authorized_keys** del servidor.

- **certbot**: es un servicio que permite obtener certificados SSL y configurarlos automáticamente en el servidor web.

4.9.2. CONFIGURACIÓN DE LA RED

El ordenador se ha conectado a la red mediante un cable Ethernet para una mejor comunicación entre sistemas. En la configuración del router, se ha asignado una dirección IP estática (dentro de la red local) al ordenador para que siempre tenga la misma dirección IP. Se ha contactado con el proveedor de Internet para consultar si sería posible obtener una dirección IP estática para el ordenador, pero no se ha podido conseguir, ya que utilizan la tecnología CG-NAT. Esto no es un problema ya que existe un servicio llamado **DuckDNS** que permite asociar una dirección IP de un ordenador a un dominio. Para configurar este servicio, se ha seguido la guía de instalación que se encuentra en la página web de **DuckDNS** [14]. Este servicio requiere de una tarea CRON para que se actualice la dirección IP cada 5 minutos. Esto se ha realizado con un pequeño script en Bash, que envía al servidor de **DuckDNS** la dirección IP pública del ordenador.

4.9.3. INSTALACIÓN Y EJECUCIÓN DE LA APLICACIÓN

La aplicación se ha instalado en el ordenador descargando el repositorio con el código fuente. Se ha programado un script en Bash que realiza todas las tareas necesarias antes de ejecutar la aplicación en los entornos de producción y desarrollo. Este script se ejecuta de manera manual cada vez que se quiere actualizar la aplicación, y realiza las siguientes tareas:

- Actualiza el código fuente de la aplicación.
- Instala las dependencias y compila el código fuente.
- Exporta las variables de entorno.
- Ejecuta la aplicación: para que se ejecute la nueva versión de la aplicación, se manda la señal `TERM` a todos los procesos que estén en escucha por los puertos que se necesitan para la aplicación (**4040** - Producción), configurados en ficheros `.env`. Esto provoca la terminación los procesos que estén escuchando por esos puertos.

4.9.4. CONFIGURACIÓN DE LA APLICACIÓN

Como se ha mencionado anteriormente, la aplicación cuenta con dos entornos: producción y desarrollo. Para configurar cada uno de ellos, se ha creado un fichero `.env` en la raíz del proyecto. Estos ficheros contienen las variables de entorno que se necesitan para ejecutar la aplicación. De esta manera, se asegura que la aplicación no exponga ninguna información sensible. Las variables más importantes que se han configurado en estos ficheros son las siguientes:

- Usuario y contraseña de la base de datos.
- Una cadena de texto que se utiliza para firmar los tokens de autenticación.
- Claves de acceso a los servicios de AWS.
- Puerto en el que se ejecuta la aplicación.
- Cuenta y contraseña (de aplicación) para enviar correos electrónicos.
- Nombre y contraseña de los ficheros que habilitan SSL en el servidor web.

4.9.5. CONFIGURACIÓN DE LA BASE DE DATOS

La base de datos se ha configurado por defecto con un usuario nuevo para el uso con la aplicación. Este usuario tiene permisos de lectura y escritura sobre la base de datos, por lo que no es necesario crear un usuario para cada aplicación.

4.9.6. SEGURIDAD DEL SERVIDOR

Con el objetivo de prevenir ataques a los dispositivos de la red local, se han configurado diferentes medidas de seguridad en el ordenador y el router. Estas medidas son las siguientes:

- **fail2ban**: es un servicio que permite bloquear las conexiones a un ordenador cuando se detecta un ataque de fuerza bruta a la IP de este servidor.
- Se han desactivado los puertos inactivos del ordenador, para evitar que se puedan utilizar para atacar a otros ordenadores.
- En el router, solo se ha habilitado la redirección de puertos a los que se necesitan para ejecutar la aplicación.

5. Anexos

5.1. ANEXO A: TAREAS DEL DESARROLLO

Durante todo el desarrollo del proyecto se han realizado una serie de tareas, para las cuales se ha utilizado la herramienta **Notion**. Esta herramienta proporciona una interfaz muy sencilla para la creación de listas de tareas, con la posibilidad de añadir etiquetas, asignar responsables, añadir comentarios, entre otras funcionalidades. A continuación se incluye el fichero pdf generado por Notion con las tareas realizadas durante el desarrollo del proyecto.

Bibliografía

[1] *Sueldos | Indeed.com*. https://es.indeed.com/career/salaries?from=gnav-homepage. (Accedido el 02/05/2023).

[2] Refactoring Guru. *Classification of patterns*. URL: https://refactoring.guru/design-patterns/classification.

[3] V. Sarcar. *Java Design Patterns: A Hands-On Experience with Real-World Examples*. Apress, 2018. Cap. 1. ISBN: 9781484240786. URL: https://books.google.es/books?id=vPt9DwAAQBAJ.

[4] I. Fette y A. Melnikov. «The WebSocket Protocol». En: (dic. de 2011). ISSN: 2070-1721. DOI: 10.17487/RFC6455. URL: https://www.rfc-editor.org/info/rfc6455.

[5] *Docker overview | Docker Documentation*. (Accedido el 21/12/2022). URL: https://docs.docker.com/get-started/overview/.

[6] *Volumes | Docker Documentation*. (Accedido el 21/12/2022). URL: https://docs.docker.com/storage/volumes/.

[7] *Overview | Prometheus*. (Accedido el 21/12/2022). URL: https://prometheus.io/docs/introduction/overview/.

[8] *Spring Boot app metrics - with Prometheus and Micrometer - Tutorial Works*. (Accedido el 22/12/2022). URL: https://www.tutorialworks.com/spring-boot-prometheus-micrometer/.

[9] *What Is Grafana? Why Use It? Everything You Should Know About It - Scaleyourapp*. (Accedido el 22/12/2022). URL: https://scaleyourapp.com/what-is-grafana-why-use-it-everything-you-should-know-about-it/.

[10] *Spring Boot 2.1 System Monitor | Grafana Labs*. URL: https://grafana.com/grafana/dashboards/11378-justai-system-monitor/.

[11] *Visualizations | Grafana documentation*. URL: https://grafana.com/docs/grafana/latest/panels-visualizations/visualizations/.

[12] *Promtail-Loki-Grafana-using-Docker-Compose/loki-config.yaml*. URL: https://github.com/shazforiot/Promtail-Loki-Grafana-using-Docker-Compose/blob/main/loki-config.yaml.

[13] *Overview | Grafana Loki documentation*. URL: https://grafana.com/docs/loki/latest/fundamentals/overview/.

[14] *Duck DNS - install*. (Accedido el 17/10/2022). URL: https://www.duckdns.org/install.jsp.