

## **MÓDULO DE PROYECTO**

### **CHAT ONLINE EN TIEMPO REAL**

**Autor: César Gómez Arroyo**

**Tutor: Marlene Alicia Arangú Lobig**

CFGS Desarrollo de Aplicaciones Web  
Curso 2023-2024

CENTRO FORMACIÓN PROFESIONAL THE CORE  
TRES CANTOS UNIE-UNIVERSIDAD  
Convocatoria de Presentación: Junio 2024

## Contenido

Índice de imágenes.....	3
1. Resumen.....	4
2. Justificación del proyecto.....	5
2.1. Contexto y Motivación .....	5
2.2. Objetivos y Alcance del Proyecto .....	5
2.3. Metodología y Planificación .....	6
3. Desarrollo .....	8
3.1. Estado del Arte y Tecnologías Utilizadas.....	8
3.1.1. Estado Actual del Área que Trata el Proyecto.....	8
3.1.2. Revisión de Aplicaciones Similares.....	9
3.2. Tecnologías Utilizadas en el Desarrollo.....	13
3.3. Diseño e Implementación de la Aplicación .....	14
3.3.1. Análisis de Requisitos .....	14
3.3.2. Diseño.....	15
3.3.3. Desarrollo de las Funcionalidades Principales .....	24
3.4. Resultados y Evaluación del Proyecto.....	30
3.4.1. Testing .....	30
3.4.2. Análisis y Evaluación de la Eficacia y Eficiencia de la Aplicación.....	32
3.4.3. Comparativa con Otras Soluciones Existentes .....	33
4. Conclusiones.....	35
4.1. Conclusiones principales .....	35
4.2. Limitaciones y posibles mejoras .....	36
5. Bibliografía – Estilo APA .....	37
6. Anexo – Ejecutar el proyecto en local.....	38
7. Agradecimientos .....	39

## Índice de imágenes

Ilustración 1 – Planificación.....	7
Ilustración 2 – Imágen WhatsApp .....	9
Ilustración 3 – Imágen Telegram.....	10
Ilustración 4 – Imágen Signal .....	11
Ilustración 5 – Imágen Slack.....	12
Ilustración 6 - Diseño de la arquitectura de autenticación .....	16
Ilustración 7 - Diseño de la arquitectura de WebSockets .....	16
Ilustración 8 - Estructura de directorios del cliente .....	17
Ilustración 9 - Estructura de directorios del servidor.....	18
Ilustración 10 - Boceto de wireframes .....	20
Ilustración 11 - Vista de autenticación en móvil.....	21
Ilustración 12 - Vista de autenticación en escritorio.....	22
Ilustración 13 - Vista de chat en móvil.....	23
Ilustración 14 - Vista de chat en escritorio.....	24
Ilustración 15 - Implementación de JWT.....	25
Ilustración 16 - Modelo de usuarios BBDD .....	26
Ilustración 17 - Colección de BBDD de MongoDB.....	27
Ilustración 18 - Uso de WebSockets.....	28
Ilustración 19 - Funcionalidad de envío de archivos .....	29
Ilustración 20 - Ejemplo test cliente.....	30
Ilustración 21 - Ejemplo test servidor .....	31
Ilustración 22 - Resultados tests .....	31
Ilustración 23 - Resultado PageSpeed Insights.....	33
Ilustración 24 - Script para ejecución del proyecto en local .....	38

## 1. Resumen

Este informe detalla el desarrollo de un proyecto de Trabajo de Fin de Grado (TFG) para el ciclo formativo de Desarrollo de Aplicaciones Web (DAW). El proyecto consiste en la creación de un chat online con funcionalidades tanto útiles como de seguridad y privacidad.

La aplicación está diseñada como una Single Page Application (SPA) utilizando Vue.js para el frontend, lo que proporciona una experiencia de usuario fluida y atractiva.

La comunicación en tiempo real es una característica clave de este proyecto, facilitada por el uso de WebSockets. Esta tecnología permite una comunicación bidireccional instantánea entre el cliente y el servidor, lo que es esencial para la funcionalidad de chat en tiempo real. El backend de la aplicación está desarrollado en Node.js, y se ha utilizado MongoDB para el almacenamiento seguro de datos.

El desarrollo del proyecto se ha llevado a cabo utilizando un enfoque de desarrollo ágil, lo que ha permitido una adaptación flexible a los cambios y requerimientos a lo largo del proceso. Además, se han empleado herramientas modernas de desarrollo como pnpm para la gestión del monorepositorio, asegurando una configuración común y eficiente para el cliente y el servidor.

El informe abarca todas las etapas del desarrollo del proyecto, desde la justificación y el análisis inicial, pasando por el diseño y la implementación, hasta la evaluación final de los resultados. Se destacan las funcionalidades principales del sistema, como la autenticación de usuarios mediante JWT, la creación de salas de chat privadas, y la capacidad de enviar archivos de manera segura. También se incluye una comparativa con otras soluciones existentes en el mercado, evaluando la eficacia y eficiencia de la aplicación desarrollada. Finalmente, se exponen las conclusiones obtenidas a lo largo del desarrollo de este proyecto.

**Palabras clave:** TFG (Trabajo de Fin de Grado), Desarrollo de Aplicaciones Web, Chat online, Single Page Application (SPA), Vue.js, WebSockets, Node.js, MongoDB, JWT (JSON Web Token), Desarrollo ágil, Testing.

## 2. Justificación del proyecto

### 2.1. Contexto y Motivación

La elección de este proyecto surge de la necesidad imperante de abordar dos aspectos cruciales en el ámbito de las comunicaciones digitales. En primer lugar, el creciente énfasis en la seguridad de las comunicaciones online y la protección de los datos de los usuarios se ha convertido en una prioridad en el panorama actual de la tecnología. Con la proliferación de plataformas de mensajería y el intercambio constante de información sensible a través de internet, es esencial garantizar la privacidad y confidencialidad de las conversaciones. Los incidentes de seguridad y las fugas de datos son cada vez más frecuentes, lo que ha generado una creciente preocupación entre los usuarios por la protección de su información personal. En este contexto, el desarrollo de este proyecto de chat online con funcionalidades avanzadas de seguridad, como la encriptación de usuarios y la volatilidad de mensajes, se presenta como una respuesta directa a esta necesidad urgente.

Por otro lado, la elección de este proyecto también está motivada por un interés genuino en explorar y aplicar tecnologías innovadoras y modernas en el ámbito del desarrollo web. En particular, el uso de WebSockets representa una oportunidad emocionante para crear aplicaciones web altamente interactivas y en tiempo real. Los WebSockets permiten establecer una conexión bidireccional entre el cliente y el servidor, lo que facilita la comunicación instantánea y la actualización en tiempo real de los datos. Esta tecnología ha revolucionado la forma en que se construyen las aplicaciones web, abriendo nuevas posibilidades en términos de interactividad y experiencia de usuario. La curiosidad por explorar y dominar estas tecnologías innovadoras ha sido un factor determinante en la elección de este proyecto, ya que representa una oportunidad única para adquirir habilidades técnicas y experiencia práctica en un campo en constante evolución.

### 2.2. Objetivos y Alcance del Proyecto

El objetivo principal de este proyecto es desarrollar un chat online seguro y eficiente que cumpla con estándares de seguridad en la transmisión y almacenamiento de datos. Esto incluye la implementación de medidas de encriptación para proteger la privacidad de los usuarios y garantizar la confidencialidad de las conversaciones. Además, se busca crear una

experiencia de usuario fluida y atractiva mediante el uso de tecnologías modernas como WebSockets y Vue.js.

El alcance del proyecto abarca desde el diseño y la planificación hasta la implementación y la prueba del sistema. Se espera que el chat online desarrollado sea capaz de manejar múltiples usuarios simultáneamente, permitiendo la creación de salas de chat privadas y el intercambio de mensajes y archivos de manera segura.

Los objetivos específicos incluyen:

- **Seguridad y Privacidad:** Implementar encriptación de usuarios y mensajes para garantizar la confidencialidad.
- **Interactividad en Tiempo Real:** Utilizar WebSockets para proporcionar una comunicación fluida y en tiempo real.
- **Experiencia de Usuario:** Diseñar una interfaz intuitiva y responsiva utilizando Vue.js y Tailwind CSS.
- **Gestión de Usuarios:** Implementar autenticación JWT para asegurar que solo usuarios autorizados accedan al sistema.
- **Manejo de Archivos:** Permitir el envío y recepción de archivos dentro de las salas de chat.

## 2.3. Metodología y Planificación

Para llevar a cabo este proyecto, se han seguido varias técnicas de la metodología ágil: diseño, desarrollo y testing, lo cual ha permitido adaptarse a los cambios y requerimientos a lo largo del proceso de desarrollo. Además, se ha establecido un plan detallado que incluye la definición de tareas, la asignación de recursos y la estimación de tiempos, con el fin de garantizar la entrega oportuna y exitosa del proyecto.

La planificación se ha estructurado en las siguientes fases:

### 1. Fase de Análisis:

- Recolección de requisitos funcionales y no funcionales.
- Investigación de tecnologías y herramientas adecuadas.

### 2. Fase de Diseño:

- Diseño de la arquitectura de la aplicación.
- Creación de prototipos de la interfaz de usuario.

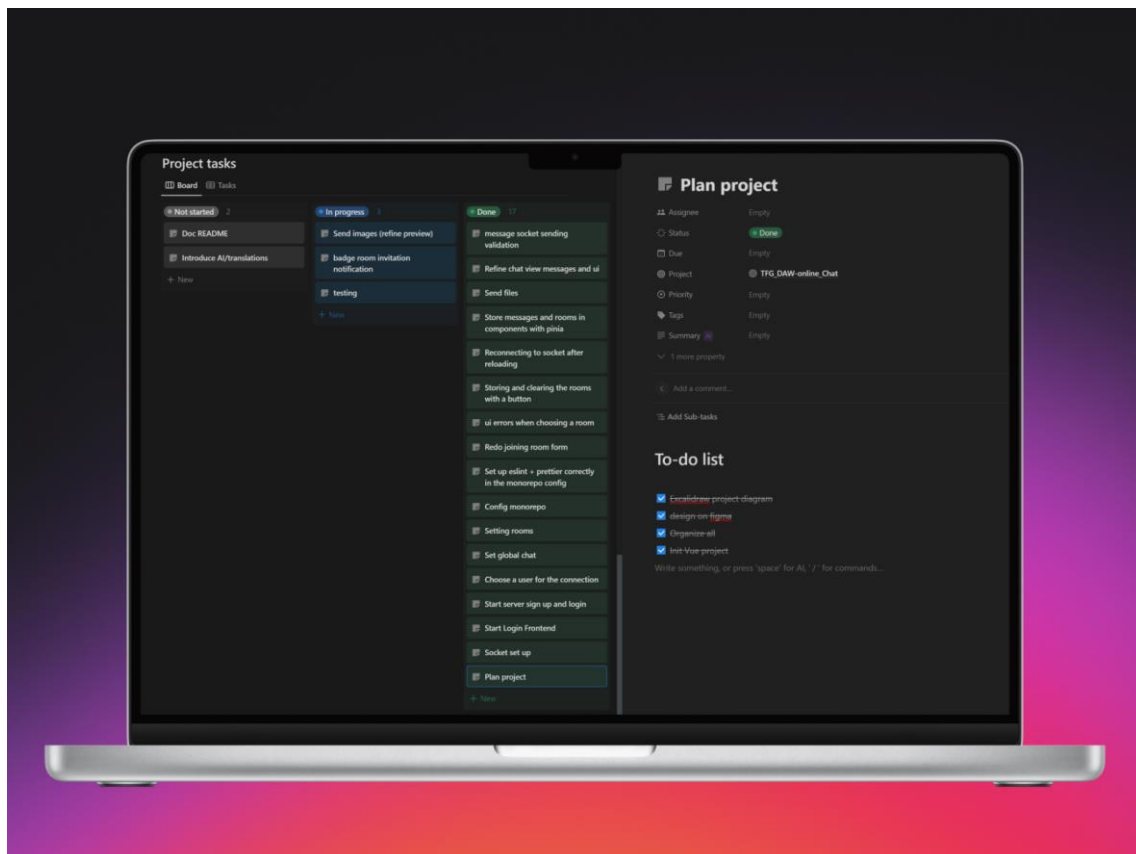
### 3. Fase de Implementación:

- Desarrollo del frontend utilizando Vue.js y Tailwind CSS.
- Desarrollo del backend con Node.js y configuración de WebSockets.
- Integración de la autenticación JWT y la encriptación de usuarios.

#### 4. Fase de Pruebas:

- Realización de pruebas unitarias.
- Pruebas de usabilidad y rendimiento.

Cada fase ha sido acompañada de un ciclo iterativo de desarrollo, asegurando que cada incremento funcional del proyecto se integre, pruebe y ajuste según sea necesario. La utilización de herramientas de gestión de proyectos, como Notion, ha facilitado el seguimiento del progreso y la organización.



*Ilustración 1 – Planificación*

En resumen, la metodología ágil adoptada ha permitido una respuesta rápida a los desafíos técnicos y cambios de requisitos, asegurando la calidad y la entrega efectiva del proyecto dentro del plazo estipulado.

## 3. Desarrollo

### 3.1. Estado del Arte y Tecnologías Utilizadas

#### 3.1.1. Estado Actual del Área que Trata el Proyecto

El área de las aplicaciones de mensajería y chat online ha experimentado un crecimiento exponencial en la última década. Con la proliferación de dispositivos móviles y el aumento de la conectividad global, la demanda de plataformas de comunicación instantánea se ha disparado y aplicaciones como WhatsApp y Telegram han establecido nuevos estándares en cuanto a usabilidad, seguridad y funcionalidad.

La seguridad en las comunicaciones digitales es un tema crítico. Incidentes recientes de fugas de datos han puesto de manifiesto la necesidad de mecanismos robustos para proteger la privacidad de los usuarios. La encriptación de extremo a extremo por usuario y navegador (E2EE) ha surgido como una solución vital, garantizando que solo los participantes de una conversación puedan leer los mensajes. Este enfoque ha sido adoptado por las principales aplicaciones de mensajería, aunque la implementación varía en términos de robustez y transparencia.

Además de la seguridad, la experiencia de usuario es otro factor crucial. Las aplicaciones modernas deben ser rápidas, responsivas y capaces de manejar múltiples usuarios simultáneamente sin comprometer el rendimiento. Las tecnologías de frontend como React, Angular y Vue.js, junto con frameworks CSS como Tailwind y Bootstrap, han facilitado la creación de interfaces de usuario altamente interactivas y atractivas.

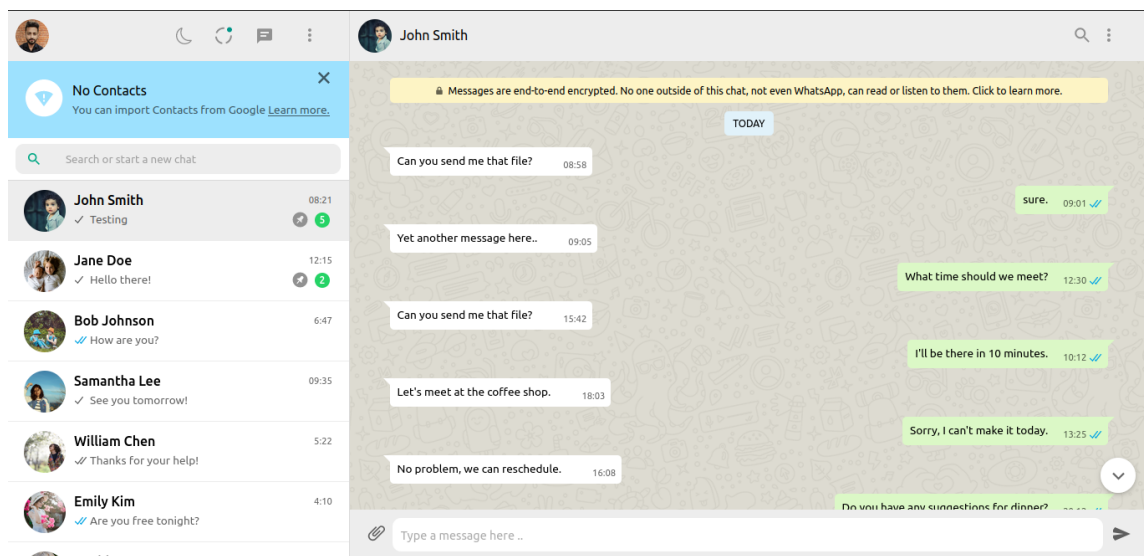
En el backend, tecnologías como Node.js, Django y PHP proporcionan la infraestructura necesaria para manejar la lógica de negocio, la gestión de usuarios y la integración con bases de datos. Bases de datos NoSQL como MongoDB y Redis son preferidas por su capacidad de manejar grandes volúmenes de datos y proporcionar respuestas rápidas, lo que es esencial para aplicaciones en tiempo real.



### 3.1.2. Revisión de Aplicaciones Similares

Para entender mejor las características y funcionalidades de las aplicaciones de chat, se realizó una revisión de varias plataformas populares:

- **WhatsApp:** Una de las aplicaciones de mensajería más utilizadas a nivel mundial. Ofrece encriptación de extremo a extremo, chats grupales, envío de archivos multimedia, llamadas de voz y video, y una interfaz de usuario intuitiva.



*Ilustración 2 – Imágen WhatsApp*

- **Telegram:** Conocida por su enfoque en la velocidad y la seguridad, Telegram permite la creación de chats secretos con autodestrucción de mensajes, uso de bots para automatización y canales públicos para difusión de mensajes a grandes audiencias.

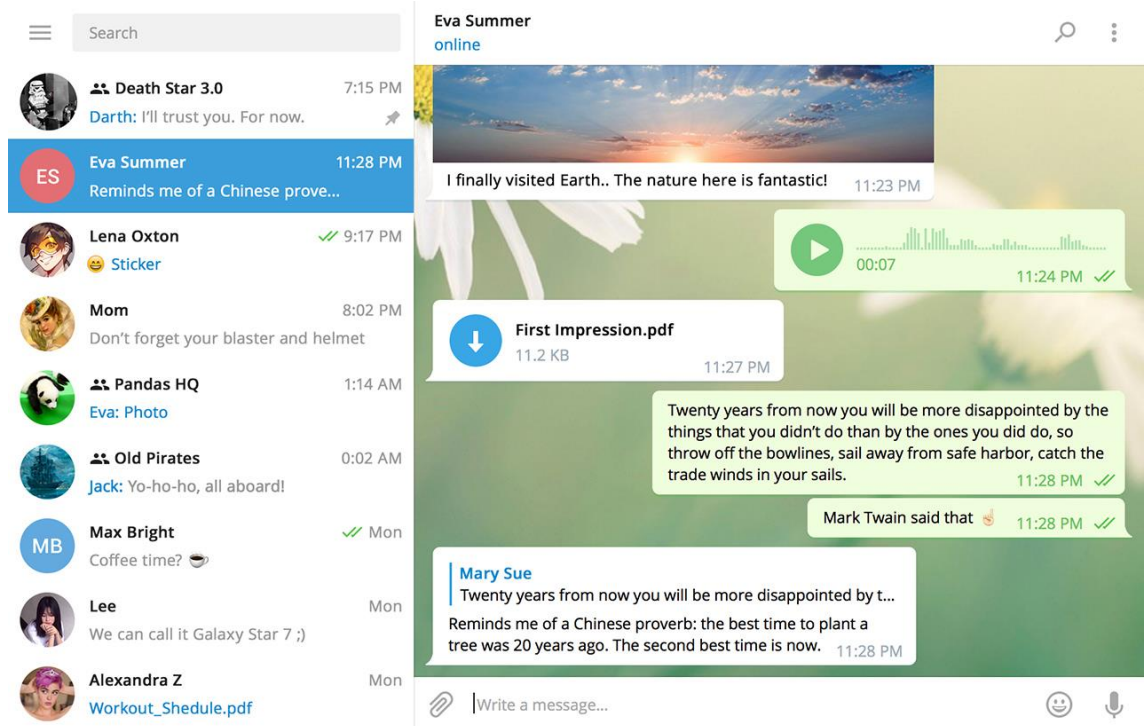


Ilustración 3 – Imágen Telegram

- **Signal:** Aclamada por su fuerte enfoque en la privacidad, Signal ofrece encriptación de extremo a extremo para todos los tipos de comunicación. Además, su código fuente es de código abierto, lo que permite una mayor transparencia y verificación por parte de la comunidad.

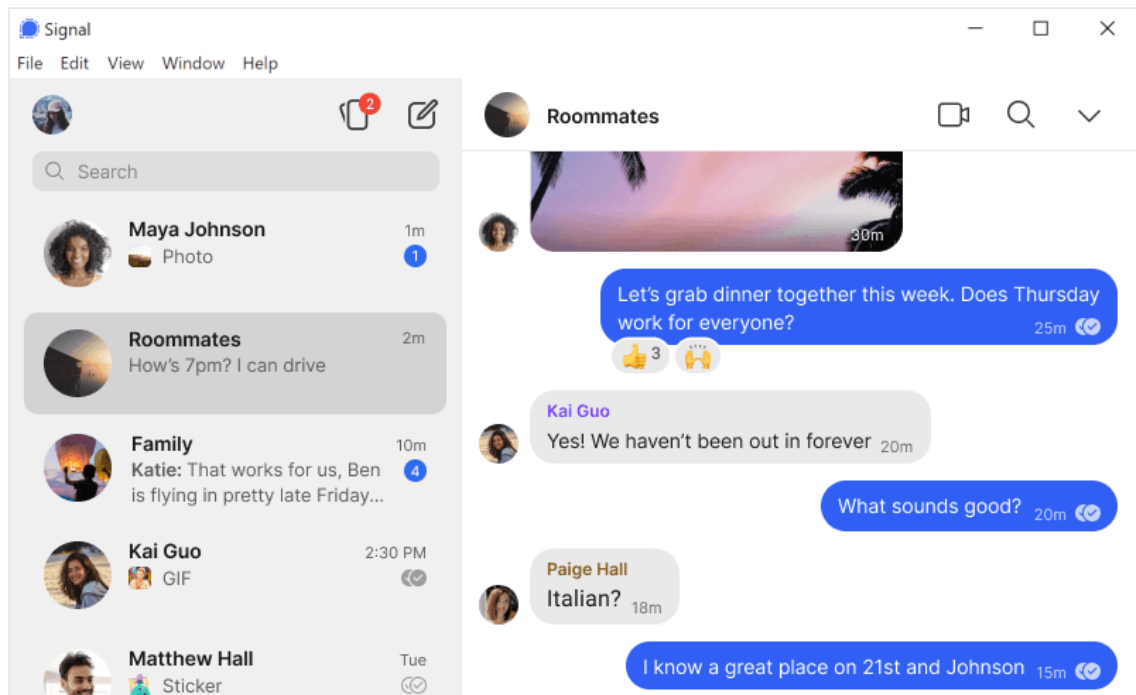


Ilustración 4 – Imágen Signal

- **Slack:** Orientada a la comunicación empresarial, Slack ofrece integración con diversas herramientas de productividad, canales organizados por temas, y un robusto sistema de búsqueda. Aunque la seguridad no es su principal punto de venta, proporciona encriptación de datos en tránsito y en reposo.

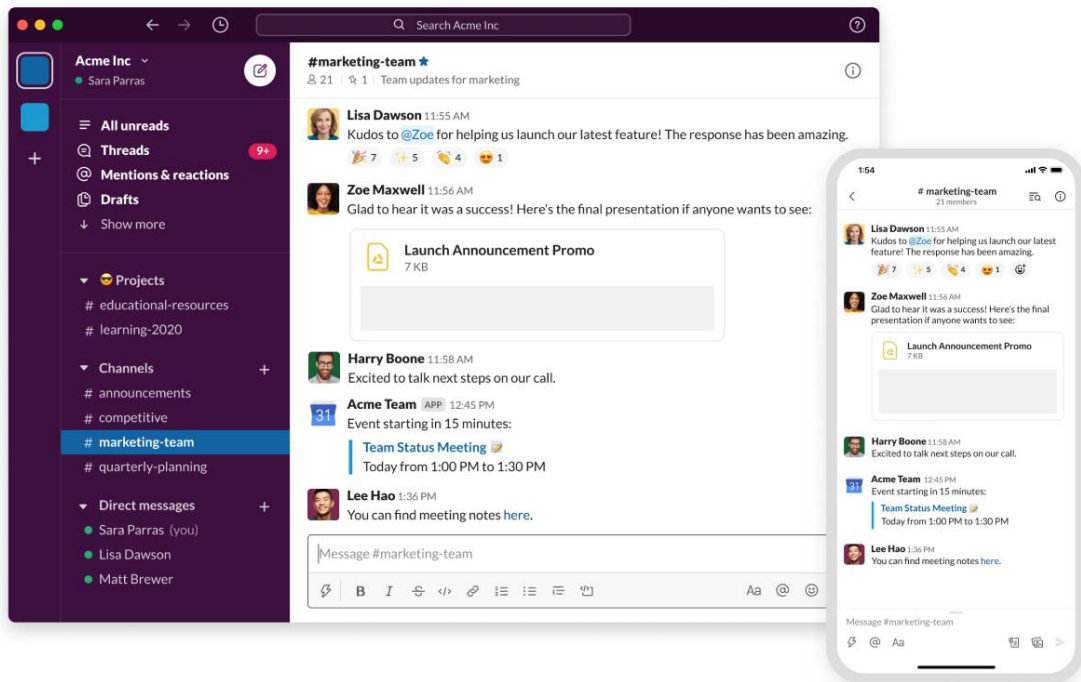


Ilustración 5 – Imágen Slack

### 3.2. Tecnologías Utilizadas en el Desarrollo

La elección de tecnologías para este proyecto se hizo con el objetivo de garantizar seguridad, eficiencia y una experiencia de usuario óptima. Las principales tecnologías utilizadas son:

- **Vue.js:** Un framework progresivo para construir interfaces de usuario. Vue.js permite la creación de componentes reutilizables y una gestión eficiente del estado, lo que es esencial para una SPA (Single Page Application). Su curva de aprendizaje relativamente baja y su capacidad de integración con otras bibliotecas lo hacen ideal para este proyecto.
- **Node.js:** Un entorno de ejecución para JavaScript en el servidor. Node.js permite manejar múltiples conexiones simultáneamente con alta eficiencia gracias a su modelo de E/S no bloqueante basado en eventos.
- **Tailwind CSS:** Un framework CSS altamente configurable que permite la creación rápida de interfaces responsivas. Tailwind facilita la estilización mediante clases utilitarias, lo que acelera el desarrollo y asegura consistencia en el diseño.
- **WebSockets:** Un protocolo web que permite la comunicación bidireccional en tiempo real entre el cliente y el servidor. WebSockets son ideales para aplicaciones de chat, ya que permiten la actualización instantánea de mensajes y la creación de experiencias interactivas sin la necesidad de refrescar la página.
- **MongoDB:** Una base de datos NoSQL que ofrece alta escalabilidad y flexibilidad. MongoDB es adecuada para almacenar grandes volúmenes de datos no estructurados, como mensajes de chat, y proporciona respuestas rápidas a las consultas.
- **Vitest:** Un framework de pruebas para asegurar la calidad del código. Vitest permite la escritura y ejecución de pruebas unitarias y de integración, lo que es crucial para mantener la robustez y la fiabilidad del sistema.
- **Figma:** Una herramienta de diseño colaborativa que permite crear prototipos de interfaces de usuario. Figma facilita la creación de diseños detallados y su validación mediante pruebas de usuario, asegurando que la interfaz sea intuitiva y efectiva.

- **Notion:** Una herramienta de organización y gestión de proyectos que permite coordinar las tareas del equipo, documentar el progreso y planificar el desarrollo de manera eficiente. Notion se ha utilizado para la planificación del proyecto y la gestión y organización de tareas.
- **PageSpeed Insights:** Una herramienta de Google que analiza el rendimiento de las páginas web. PageSpeed Insights proporciona información sobre cómo mejorar la velocidad de carga y la eficiencia de la aplicación, lo que es crucial para ofrecer una experiencia de usuario óptima. Utilizar esta herramienta ha ayudado a identificar y solucionar problemas de rendimiento, garantizando que la aplicación funcione de manera rápida y fluida en todos los dispositivos y navegadores.

### 3.3. Diseño e Implementación de la Aplicación

#### 3.3.1. Análisis de Requisitos

El análisis de requisitos es una fase que define las funcionalidades y características esenciales del sistema. Los requisitos se dividen en dos categorías: funcionales y no funcionales.

- **Requisitos Funcionales:**
  - Autenticación de usuarios mediante JWT. Json Web Token es un estándar de Internet propuesto para crear datos con firma opcional y/o cifrado opcional cuya carga útil contiene JSON que afirma un cierto número de peticiones. Los tokens se firman utilizando un secreto privado o una clave pública/privada.
  - Volatilidad de mensajes y chats ligada a la sesión de pestaña en el navegador mediante el uso de sessionStorage. El almacenamiento de sesión permite almacenar datos en el navegador en función de la memoria del sistema y los datos almacenados en el navegador hasta que se cierra el navegador o la pestaña en cuestión.
  - Comunicación en tiempo real con el servidor mediante WebSockets.
  - Creación y gestión de salas de chat privadas.
  - Envío y recepción de mensajes de texto y archivos en el chat global y las salas privadas.
- **Requisitos No Funcionales:**
  - Seguridad: Protección contra suplantación de usuarios.
  - Rendimiento: Respuesta rápida y manejo eficiente de múltiples usuarios.
  - Usabilidad: Interfaz intuitiva y fácil de usar.

- Escalabilidad: Capacidad para añadir nuevas funcionalidades y manejar un creciente número de usuarios.

### 3.3.2. Diseño

#### *3.3.2.1. Diseño de la Arquitectura de la Aplicación*

Hay dos puntos diferenciadores fundamentales que distinguen a esta aplicación de otras similares.

En primer lugar, la característica de volatilidad al cerrar la sesión en la aplicación o al cerrar la pestaña del navegador es crucial. Esto se logra mediante el uso de `sessionStorage`, que almacena datos de forma temporal y se borra automáticamente cuando se cierra la sesión o la pestaña del navegador. Esta funcionalidad garantiza la privacidad y la seguridad de los datos del usuario al eliminar cualquier rastro de la sesión.

En segundo lugar, la comodidad de tener la aplicación siempre disponible en la web sin necesidad de descargar ningún paquete de datos o aplicación adicional es un punto fuerte significativo. Esto elimina las barreras de entrada para los usuarios al proporcionarles acceso inmediato al chat desde cualquier dispositivo con un navegador web, lo que mejora la accesibilidad y la conveniencia para los usuarios en cualquier momento y lugar. En conjunto, estos dos puntos diferenciadores resaltan la atención al detalle y el compromiso con la seguridad y la experiencia del usuario en el diseño y desarrollo de esta aplicación web.

Esta aplicación web sigue el paradigma de arquitectura cliente-servidor, una estructura de diseño de software en la que las tareas se dividen entre los proveedores de recursos o servicios, conocidos como servidores, y los demandantes, llamados clientes. En este caso, la aplicación sigue un enfoque de microservicios, donde se divide en dos paquetes o componentes principales: el cliente y el servidor. Ambos componentes están gestionados dentro de un monorepositorio utilizando el gestor de paquetes `pnpm`. Esta elección de arquitectura permite compartir configuraciones comunes y facilita la implementación de integración continua, lo que mejora la eficiencia y la coherencia del desarrollo.

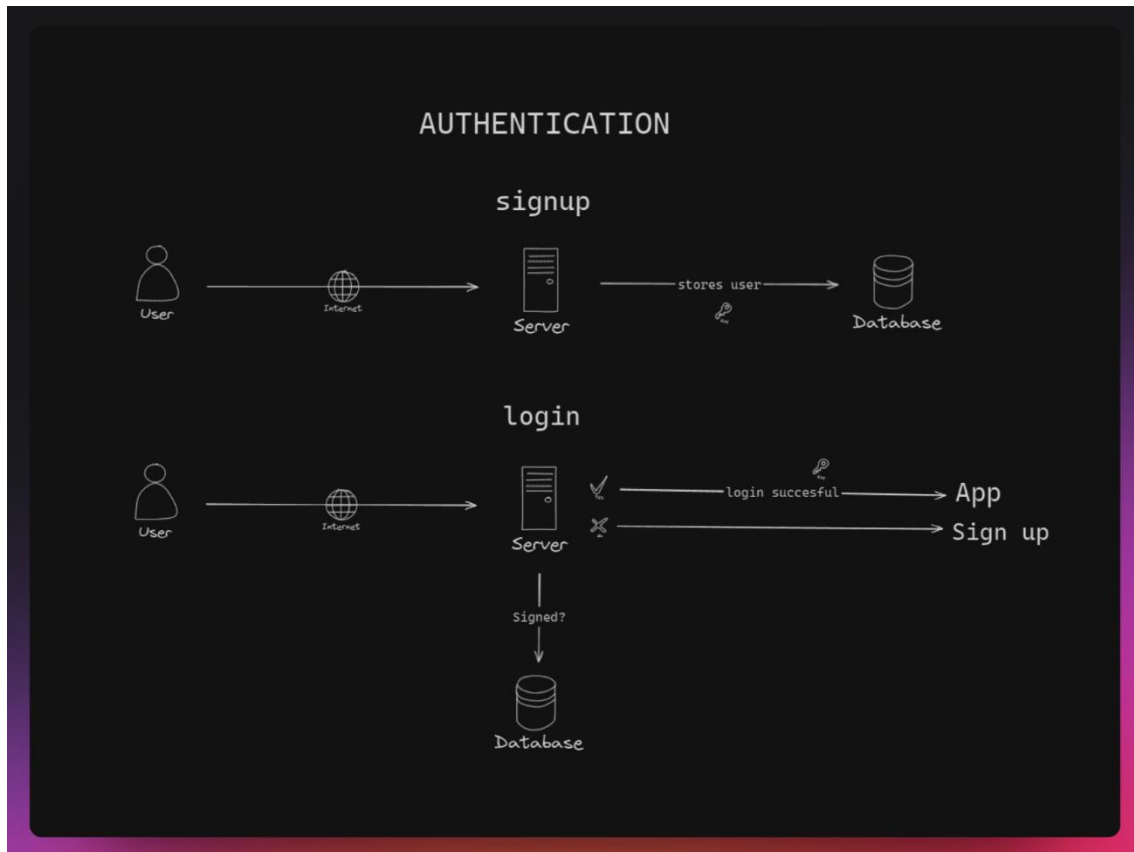


Ilustración 6 - Diseño de la arquitectura de autenticación

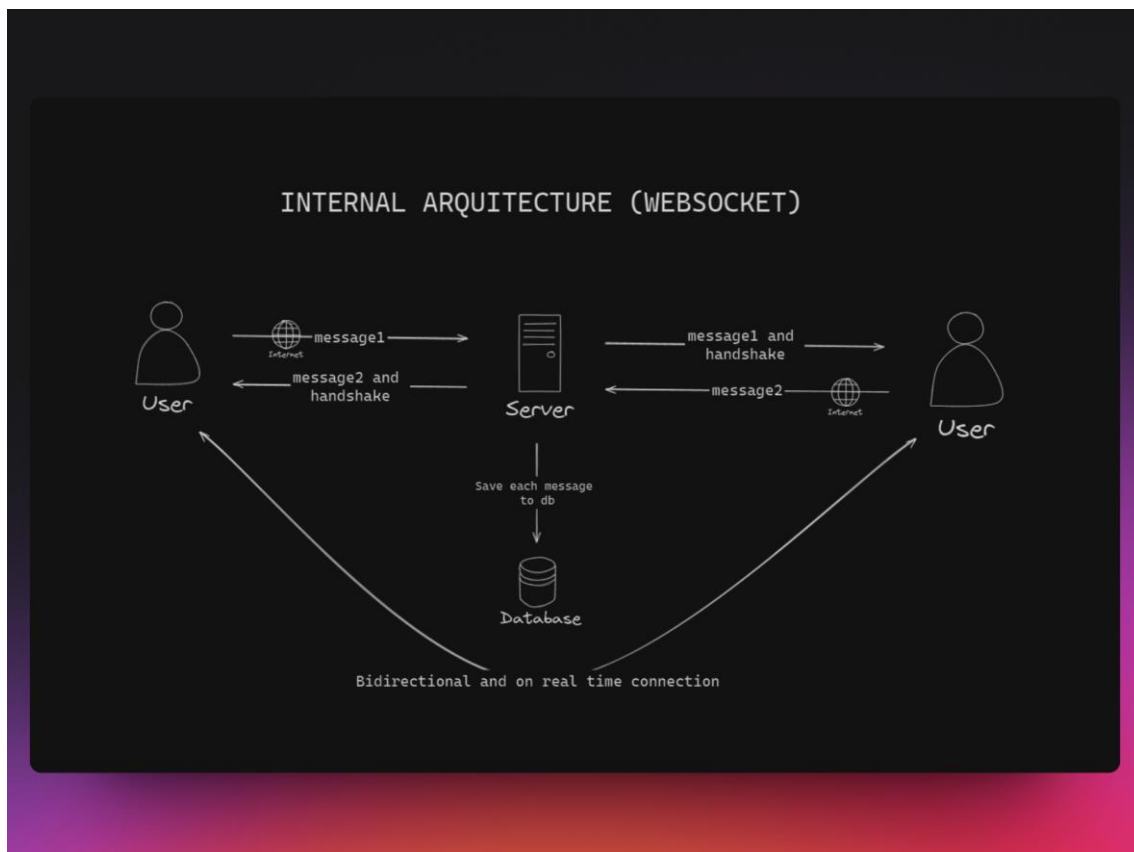


Ilustración 7 - Diseño de la arquitectura de WebSockets



A continuación, se detalla muy resumidamente la estructura de cada paquete o componente del monorepositorio:

- **Cliente (Frontend):**

- **Framework:** Vue.js
- **Estructura de Directorios:**
  - `src/main.js`: Punto de entrada de la aplicación.
  - `src/App.vue`: Componente principal.
  - `src/componentes`: Componentes reutilizables.
  - `src/vistas`: Diferentes vistas de la aplicación.
  - `src/router`: Configuración de rutas.
  - `src/store`: Gestión del almacenamiento del estado en cliente con Pinia.
  - `src/___{folder/file}__tests.js`: En varios directorios principales existen carpetas o archivos `.file.test.js`, que contiene los test unitarios de un directorio completo o un fichero en concreto.

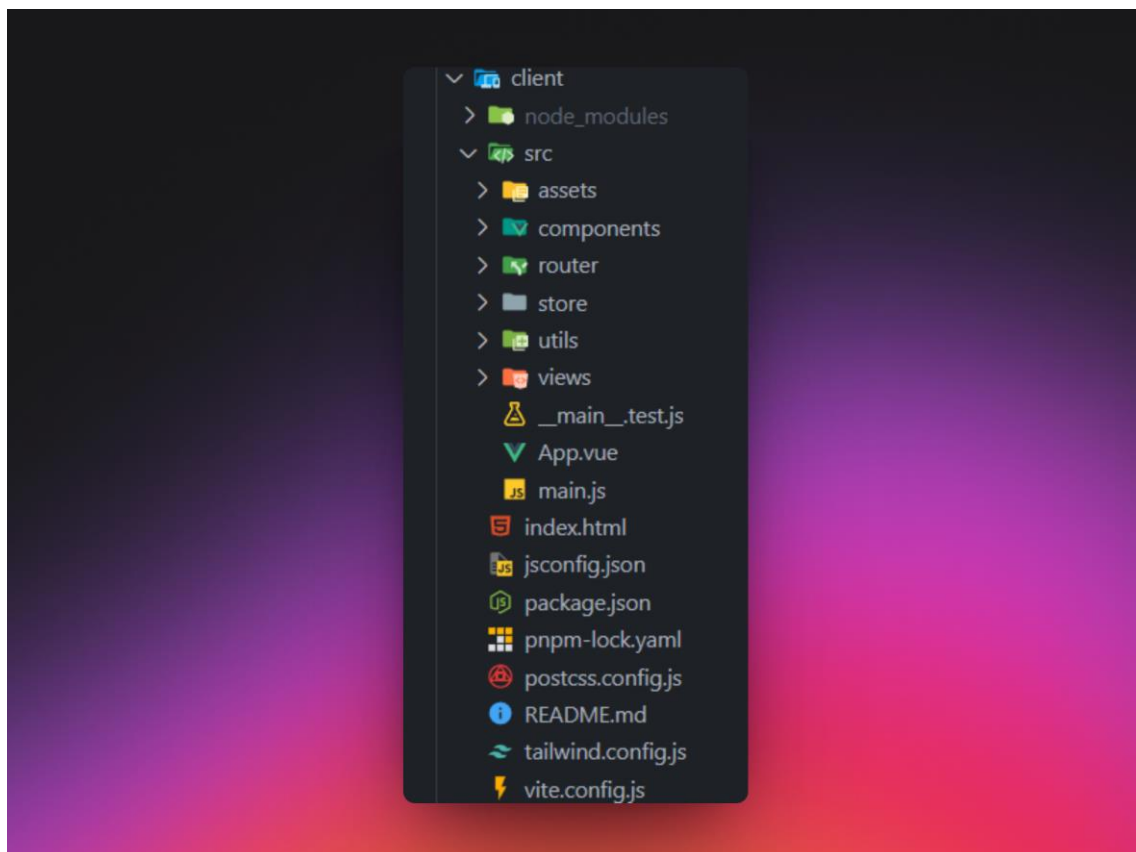
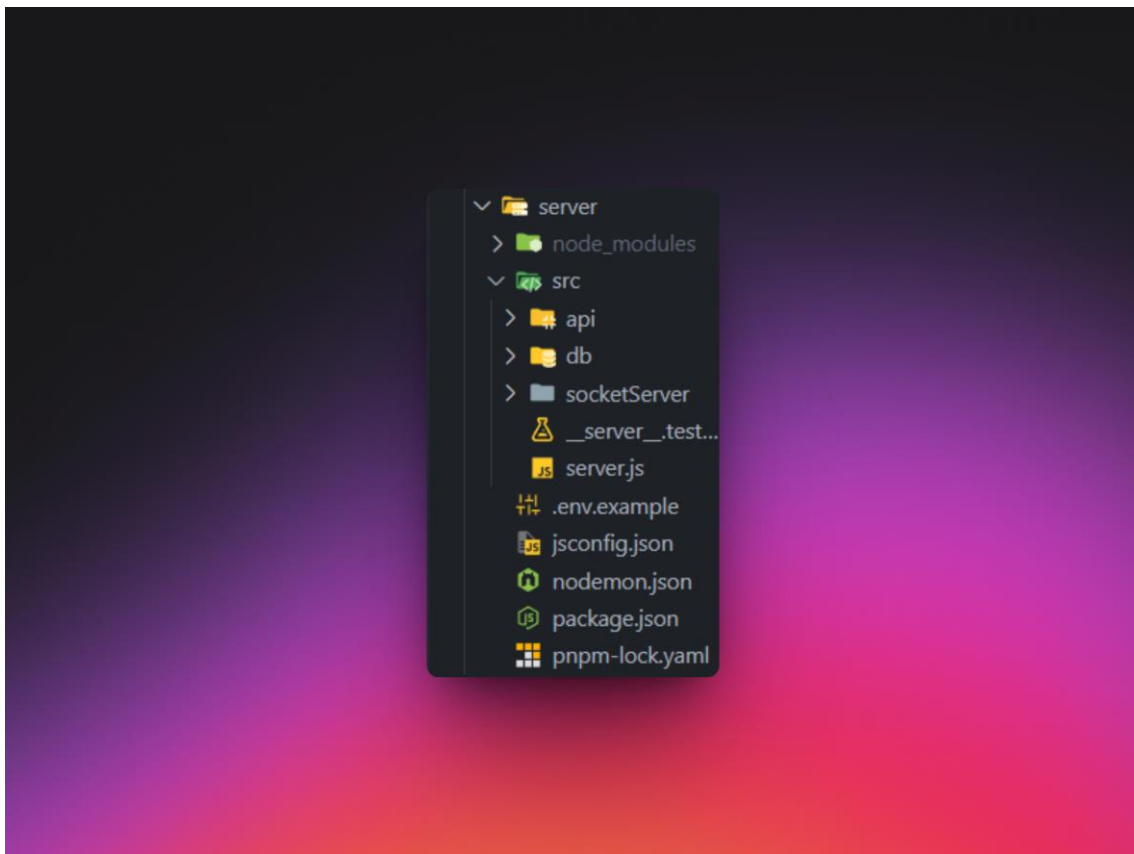


Ilustración 8 - Estructura de directorios del cliente

- **Servidor (Backend):**
  - **Framework:** Node.js
  - **Estructura de Directorios:**
    - `src/server.js`: Punto de entrada del servidor.
    - `src/api`: Endpoints de la API.
    - `src/db`: Configuración de la base de datos.
    - `src/socketServer`: Configuración de WebSockets.
    - `src/{folder/file}__tests.js`: En varios directorios principales existen carpetas o archivos `.file.test.js`, que contiene los test unitarios de un directorio completo o un fichero en concreto.



*Ilustración 9 - Estructura de directorios del servidor*

### 3.3.2.2. Diseño de la UI

El diseño de la interfaz de usuario (UI) es un componente fundamental para asegurar una experiencia de usuario atractiva e intuitiva. Para el desarrollo de la UI, se ha utilizado Figma, una herramienta colaborativa de diseño que permite la creación de prototipos de alta fidelidad y la validación de conceptos de diseño antes de su implementación.

- **Estilo de Diseño:**

El estilo de diseño adoptado para la aplicación sigue una temática tipo "matrix" o "hacker". Este estilo se caracteriza por:

- **Interfaz Oscura:** Un diseño predominante en colores oscuros que no solo es agradable a la vista durante largas sesiones de uso, sino que también resuena con la estética de la programación y la seguridad cibernética.
- **Tipografía Monoespaciada:** El uso de fuentes monoespaciadas contribuye a la sensación de un entorno de terminal o consola, reforzando la temática de "hacker".
- **Efectos de Luz Neón:** Los efectos de luz neón, especialmente en el color verde (#00ffe0), se utilizan para resaltar botones, enlaces y otros elementos interactivos, creando un ambiente futurista y tecnológico.

- **Paleta de Colores:**

La elección de la paleta de colores es crucial para transmitir la temática y el tono de la aplicación. Para este proyecto, la paleta de colores seleccionada incluye:

- **#00ffe0:** Un color verde neón vibrante que se utiliza para resaltar elementos interactivos y activos, dando un toque moderno y tecnológico.
- **#0f363e:** Un verde oscuro que proporciona un fondo contrastante y establece un ambiente técnico y profesional.
- **#000000:** El negro, utilizado como color de fondo principal, crea un contraste dramático y facilita la lectura en pantalla.
- **#ffffff:** El blanco, utilizado para texto y elementos que requieren destacar en un fondo oscuro.

- **Tipografía**

Para la tipografía, se seleccionó la fuente **"Share Tech"** de Google Fonts, una elección que complementa perfectamente el estilo "hacker". La fuente "Share Tech" es moderna y tecnológica, con un diseño limpio y legible, ideal para interfaces de usuario que requieren claridad y precisión.

- **Proceso de Diseño:**

El proceso de diseño comenzó con la creación de wireframes en Figma para definir la estructura básica de la aplicación y la disposición de los elementos en la interfaz. A continuación, se desarrollaron los prototipos de alta fidelidad que incorporan la paleta de colores y el estilo visual definitivo, que se mostrará a continuación. Estos prototipos permitieron realizar pruebas de usuario preliminares para identificar posibles problemas de usabilidad y hacer ajustes antes de la implementación. Uno de los objetivos clave del diseño fue asegurar que la aplicación sea completamente responsive, es decir, que funcione y se vea bien en dispositivos de diferentes tamaños, desde teléfonos móviles hasta tablets y computadoras de escritorio. Esto se logró mediante el uso de Figma para crear diseños que se adaptan a múltiples resoluciones de pantalla, garantizando una experiencia de usuario consistente y optimizada en todas las plataformas.

A continuación se muestra el boceto de los wireframes:

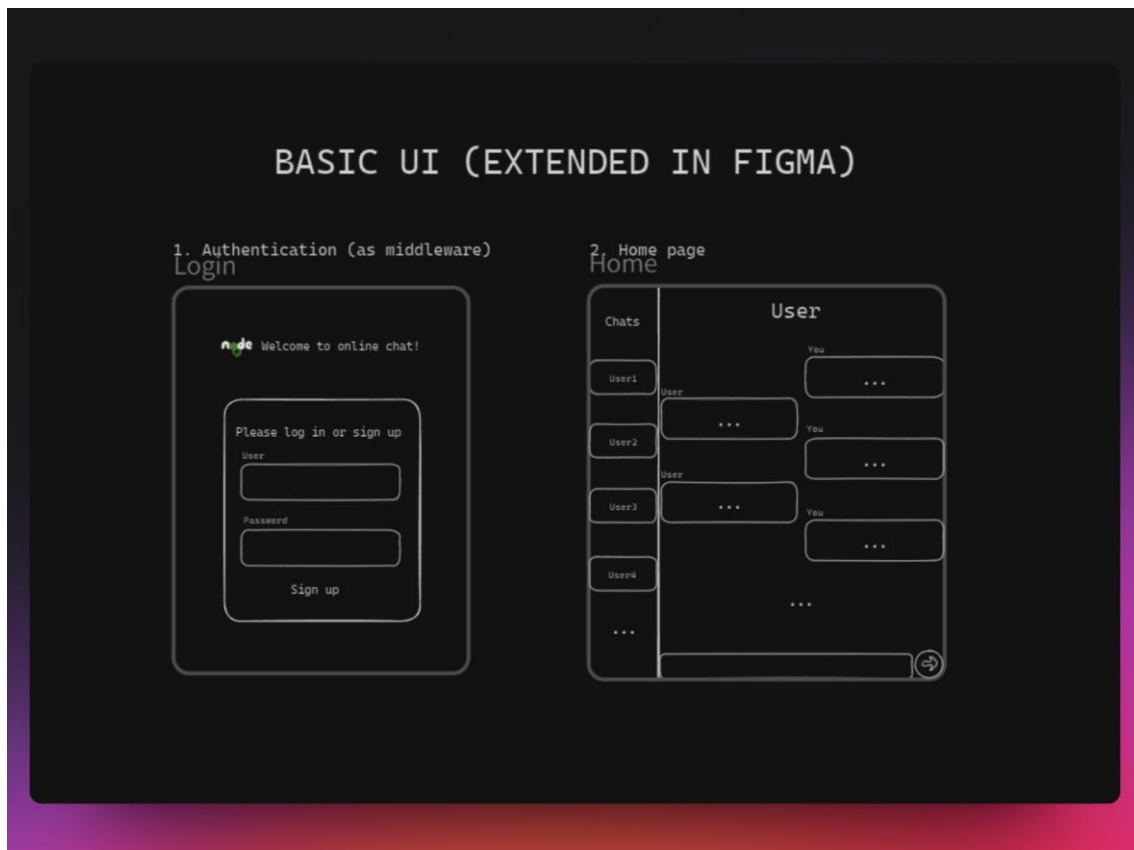
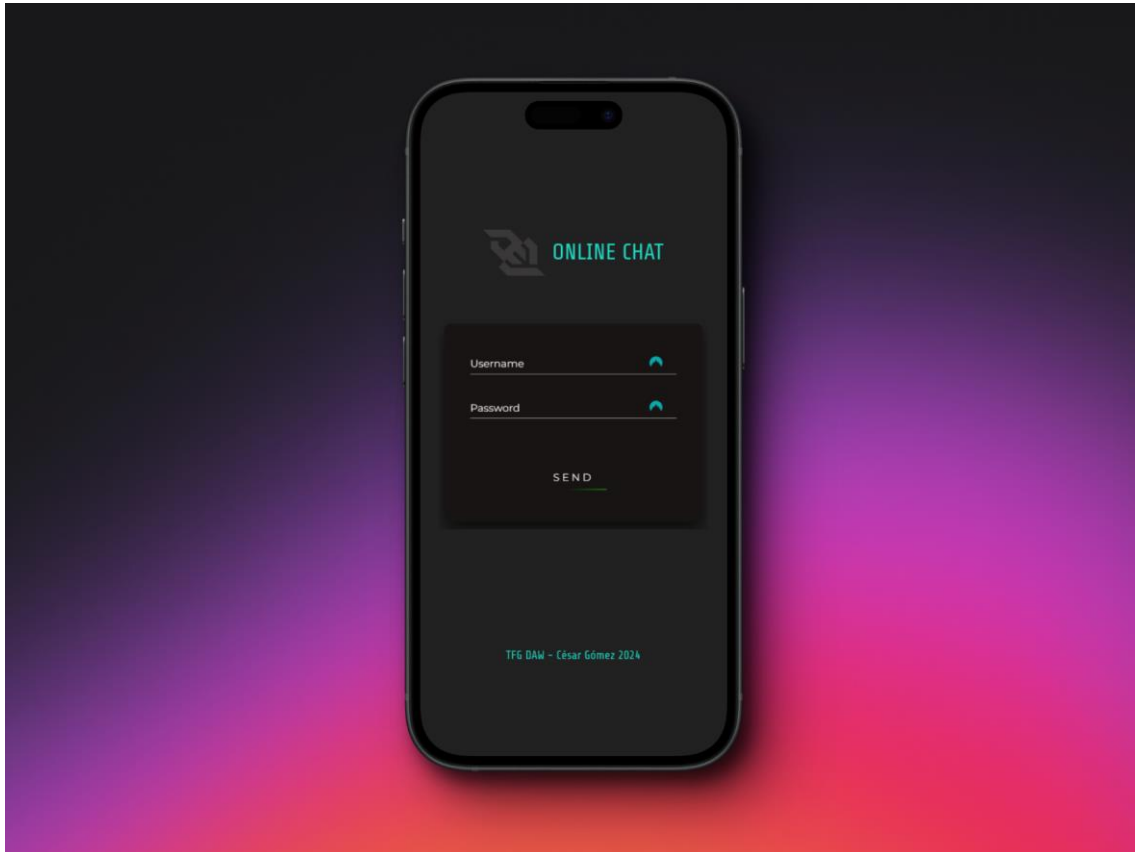
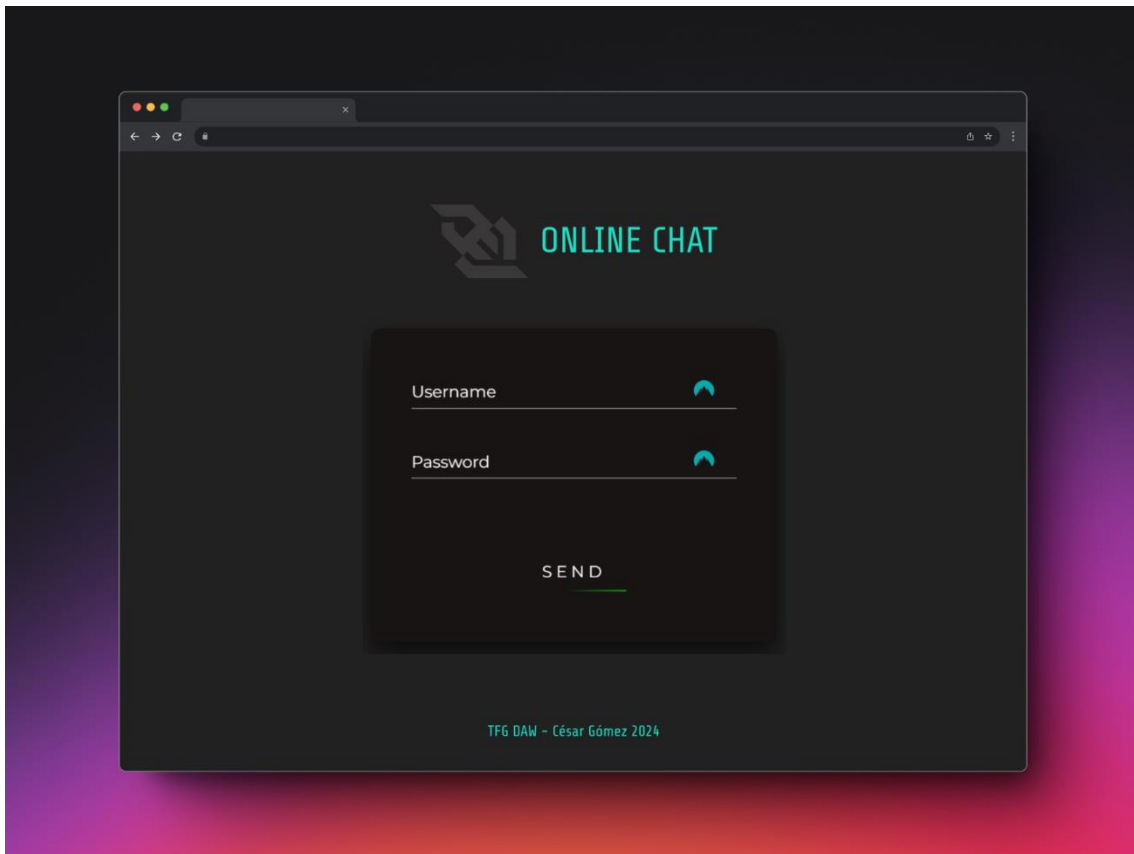


Ilustración 10 - Boceto de wireframes

- **Componentes Clave de la UI:**
  - **Vista de Inicio de Sesión:** Incluye campos para el nombre de usuario y la contraseña, con botones destacados para iniciar sesión y registrarse.

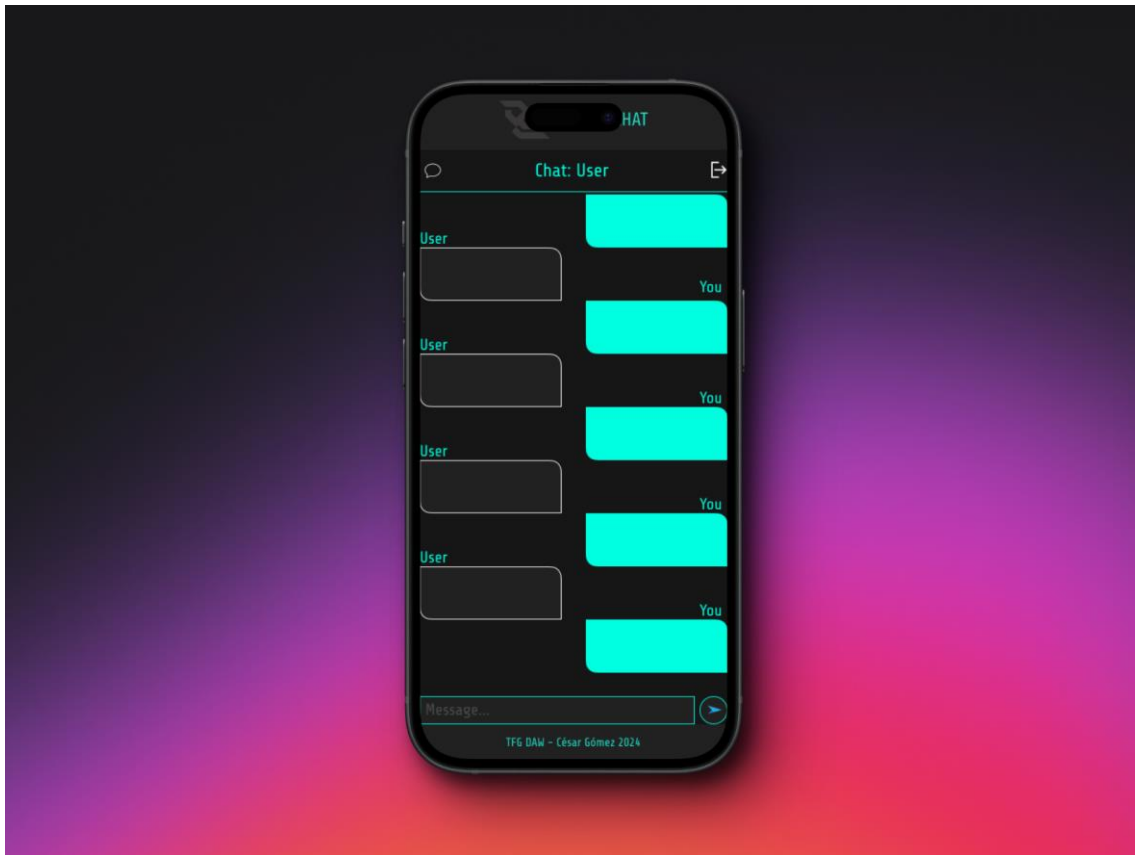


*Ilustración 11 - Vista de autenticación en móvil*



*Ilustración 12 - Vista de autenticación en escritorio*

- **Vista de Chat:** Diseñada para mostrar mensajes en tiempo real con burbujas de texto estilizadas y opciones para enviar archivos. Ésta incluye la sub-vista de las salas.



*Ilustración 13 - Vista de chat en móvil*

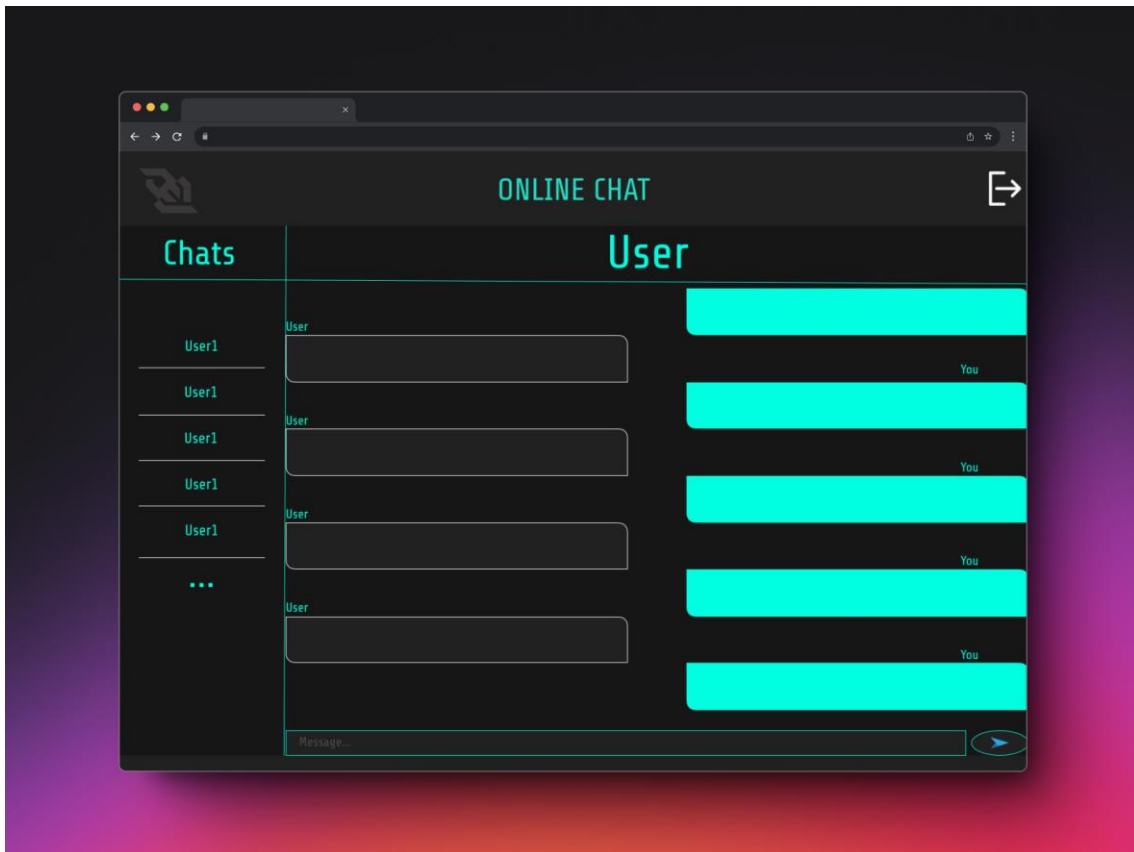


Ilustración 14 - Vista de chat en escritorio

El uso de Figma ha permitido una colaboración efectiva y una iteración rápida sobre el diseño, asegurando que la interfaz final no solo sea visualmente atractiva, sino también funcional y alineada con las expectativas de los usuarios.

### 3.3.3. Desarrollo de las Funcionalidades Principales

- **Autenticación y Autorización:**
  - Implementación de JWT para la autenticación segura de usuarios.
  - Gestión de sesiones y permisos de acceso basados en sessionStorage. Ejemplo del uso del JWT en el proyecto: Cuando un usuario intenta iniciar sesión enviando su nombre de usuario y contraseña a través de la ruta `/login`, el servidor verifica la existencia del usuario en la base de datos. Si se encuentra un usuario registrado con las credenciales proporcionadas, se genera un token JWT que contiene el nombre de usuario y se firma con una clave secreta única (**línea 12**). Este token se devuelve al cliente junto con la confirmación de inicio de sesión. El cliente puede luego incluir este token en las solicitudes posteriores al servidor para autenticarse y acceder a recursos protegidos. El uso de JWT proporciona una forma segura y eficiente de manejar la autenticación y autorización en la aplicación.

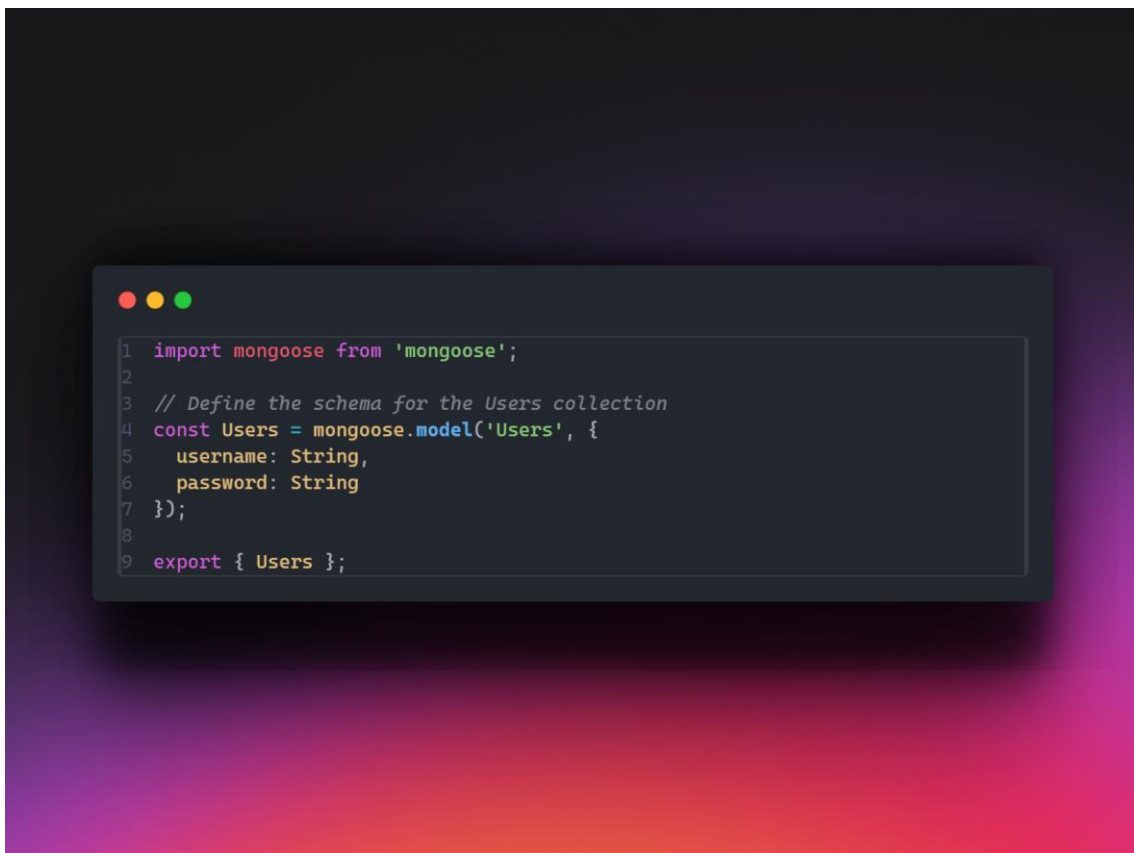


```
1 app.post('/login', async (req, res) => {
2   if (!req.body.username || !req.body.password) {
3     res.status(500).json({ error: 'Username and password are required!' });
4     return;
5   }
6   try {
7     const userRegistered = await Users.findOne({
8       username: req.body.username,
9       password: req.body.password
10    });
11    if (userRegistered) {
12      const token = jwt.sign({ username: req.body.username }, process.env.JWT_SECRET_KEY);
13      const userWithoutPassword = {
14        username: userRegistered.username
15      };
16      res.json({
17        message: 'User logged in successfully!',
18        userWithoutPassword,
19        token
20      });
21    } else {
22      res.json({ error: 'Invalid username or password' });
23    }
24  } catch (e) {
25    res.json({ error: e });
26  }
27 });
```

Ilustración 15 - Implementación de JWT

- **Almacenamiento en BBDD:**

- Implementación de una base de datos con MongoDB para el almacenamiento seguro de usuarios.
- Ejemplo de código de creación de la colección con el modelo de los usuarios. Primeramente, se importa el conector de mongoose, el cual permite conectar fácilmente el entorno de Node.js con MongoDB. Después se define el modelo de usuarios (**líneas 4 a 7**) y se exporta (**línea 9**). También se adjunta una imagen que muestra la estructura de la colección.



*Ilustración 16 - Modelo de usuarios BBDD*

```
_id: ObjectId('664cc25f1173cc49a78768f9')
username: "test"
password: "123"
__v: 0
```

```
_id: ObjectId('66547468f1a1a6ffd0038ff6')
username: "test1"
password: "123"
__v: 0
```

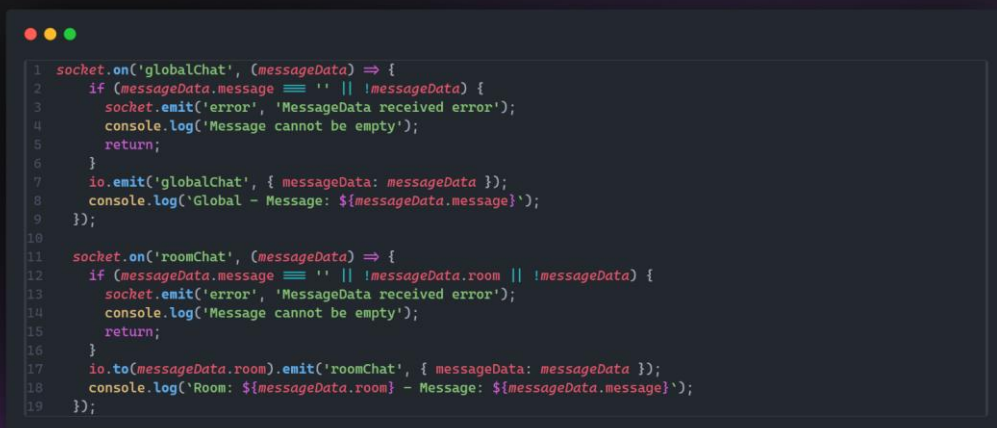
```
_id: ObjectId('66547474f1a1a6ffd0038ff4')
username: "test2"
password: "123"
__v: 0
```

```
_id: ObjectId('66547480f1a1a6ffd0038ff8')
username: "test3"
password: "123"
__v: 0
```

Ilustración 17 - Colección de BBDD de MongoDB

- **Comunicación en Tiempo Real:**

- Configuración de WebSockets para la comunicación bidireccional instantánea.
- Gestión de salas de chat y envío de mensajes en tiempo real.
- Ejemplo de uso de WebSockets en la aplicación: Las funciones `socket.on` escuchan eventos específicos del cliente, como `globalChat` y `roomChat`. Cuando se recibe un mensaje del cliente a través de `globalChat`, el servidor verifica que el mensaje no esté vacío y luego emite ese mensaje a todos los clientes conectados utilizando `io.emit()` (**línea 7**), lo que permite que todos los usuarios conectados al servidor reciban el mensaje al mismo tiempo. Del mismo modo, cuando se recibe un mensaje del cliente a través de `roomChat`, el servidor verifica que el mensaje no esté vacío y que esté asociado a una sala específica. Luego, emite el mensaje solo a los clientes que se encuentran en esa sala específica utilizando `io.to(room).emit()` (**línea 17**). Este enfoque permite una comunicación bidireccional instantánea entre los usuarios en salas globales y salas específicas, proporcionando una experiencia de chat en tiempo real.



```
1 socket.on('globalChat', (messageData) => {
2   if (messageData.message === '' || !messageData) {
3     socket.emit('error', 'MessageData received error');
4     console.log('Message cannot be empty');
5     return;
6   }
7   io.emit('globalChat', { messageData: messageData });
8   console.log('Global - Message: ${messageData.message}');
9 });
10
11 socket.on('roomChat', (messageData) => {
12   if (messageData.message === '' || !messageData.room || !messageData) {
13     socket.emit('error', 'MessageData received error');
14     console.log('Message cannot be empty');
15     return;
16   }
17   io.to(messageData.room).emit('roomChat', { messageData: messageData });
18   console.log('Room: ${messageData.room} - Message: ${messageData.message}');
19 });
```

Ilustración 18 - Uso de WebSockets

- Envío de Archivos:
  - Implementación de funcionalidad para el envío y recepción de archivos.
  - La funcionalidad de envío de archivos permite a los usuarios seleccionar y cargar archivos desde su dispositivo para compartirlos en el chat. Cuando un usuario selecciona un archivo mediante un evento de entrada de archivos, como se muestra en la función `handleFileUpload`, se verifica si se seleccionó un archivo válido. Si se selecciona un archivo, se crea un objeto `FileReader` (línea 5) para leer el contenido del archivo como una matriz de bytes. Una vez que se ha leído el archivo, se crea un objeto URL de Blob para representar el archivo como una URL que puede ser utilizada para mostrar o descargar el archivo (línea 7). Esta URL se almacena junto con el tipo de archivo en una referencia para ser utilizada más tarde al enviar el mensaje. De esta manera, el usuario puede cargar archivos de forma segura y eficiente en el chat, mejorando la experiencia de comunicación y compartiendo información de manera efectiva.

```
1 const handleFileUpload = (e) => {  
2   const file = e.target.files[0];  
3   if (file) {  
4     messageRequired.value = false;  
5     const reader = new FileReader();  
6     reader.onload = (e) => {  
7       const blobUrl = URL.createObjectURL(new Blob([e.target.result]));  
8       // Store the blobUrl and file type in a ref to be used when sending the message  
9       fileUrl.value = { url: blobUrl, type: file.type };  
10      // Disable the text input  
11    };  
12    reader.readAsArrayBuffer(file);  
13  } else {  
14    messageRequired.value = true;  
15  }  
16 };
```

*Ilustración 19 - Funcionalidad de envío de archivos*

## 3.4. Resultados y Evaluación del Proyecto

### 3.4.1. Testing

Un aspecto destacado en el desarrollo de esta aplicación es la implementación de pruebas unitarias exhaustivas utilizando el framework para tests de vitest, junto con otras bibliotecas de ayuda. Estas pruebas unitarias garantizan la estabilidad y la fiabilidad del código e informando si cada funcionalidad testada funciona como se espera, identificando y corrigiendo posibles errores y problemas de rendimiento antes de que lleguen a producción.

A continuación, ejemplos de test de los puntos de entrada del cliente (primera imagen) y el servidor (segunda imagen) y sus resultados donde se puede ver que ambos tests pasan correctamente.

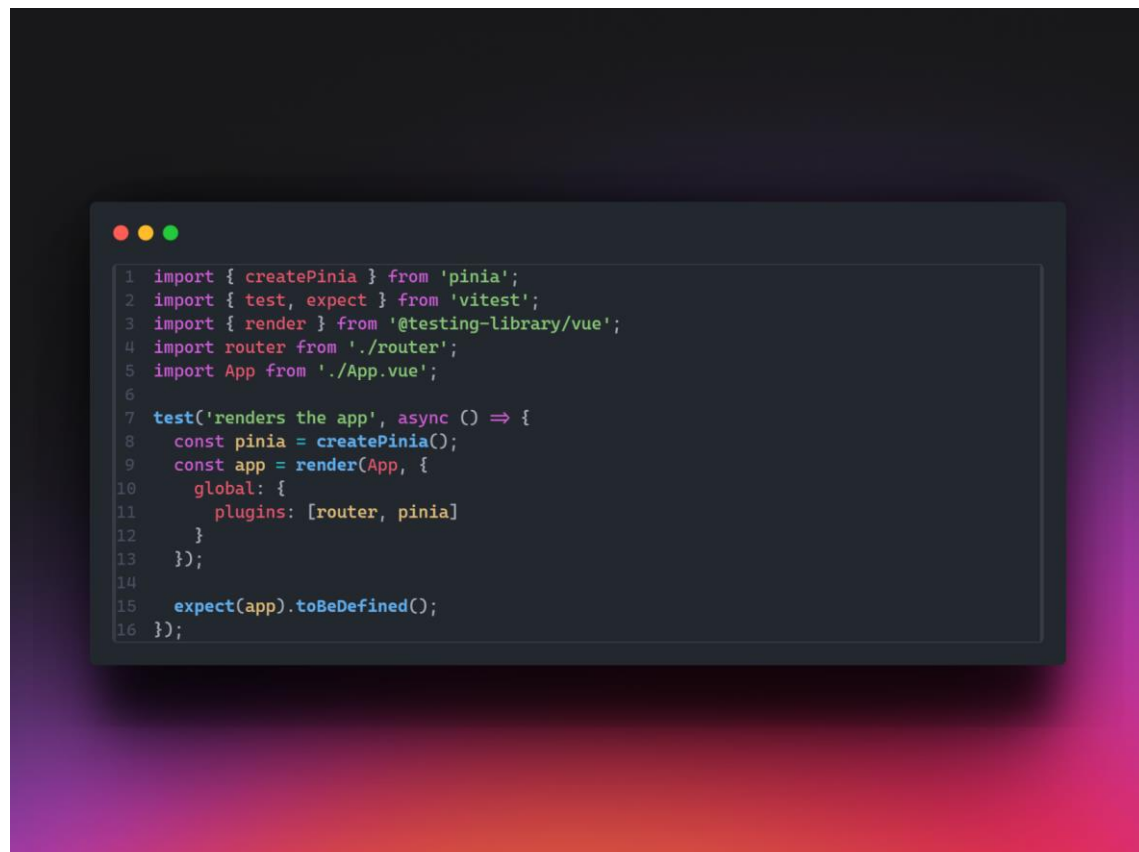


Ilustración 20 - Ejemplo test cliente

```
1 import { test, expect } from 'vitest';
2 import { exec } from 'child_process';
3
4 test('server starts without errors', async () => {
5   const server = exec('pnpm dev');
6
7   let res;
8   const maxAttempts = 20;
9   for (let attempt = 0; attempt < maxAttempts; attempt++) {
10     try {
11       res = await fetch('http://localhost:3000');
12       break;
13     } catch (error) {
14       // Wait for 1 second before the next attempt
15       await new Promise((resolve) => setTimeout(resolve, 1000));
16     }
17   }
18
19   expect(res).not.toBe(undefined);
20   server.kill();
21 });
```

Ilustración 21 - Ejemplo test servidor

```
Windows PowerShell x PowerShell x + v
online-chat on 1.0.0 main is 1.0.0 via 20.11.1 and - 10.2.4
> pnpm run test src/__main__.test.js src/__server__.test.js
> online-chat@1.0.0 test S:\myProjects\online-chat
> vitest --ui "src/__main__.test.js" "src/__server__.test.js"

DEV v1.6.0 S:/myProjects/online-chat
UI started at http://localhost:51204/__vitest__/

✓ [server] src/__server__.test.js (1)
✓ [client] src/__main__.test.js (1)

Test Files 2 passed (2)
Tests 2 passed (2)
Start at 11:27:41
Duration 1.26s (transform 122ms, setup 0ms, collect 363ms, tests 107ms, environment 327ms, prepare 605ms)

PASS Waiting for file changes...
press h to show help, press q to quit
```

Ilustración 22 - Resultados tests

Por ejemplo, en el archivo `__main__.test.js`, el cual es el punto de entrada del paquete del cliente, se realiza una prueba que verifica si la aplicación se renderiza correctamente. Se crea una instancia de Pinia, un gestor de estado para Vue.js, y se renderiza la aplicación utilizando la función `render` de `@testing-library/vue`. Luego, se asegura que la aplicación renderizada esté definida, lo que confirma que la aplicación se carga correctamente y está lista para su uso.

En el archivo `__server__.test.js`, el cual es el punto de entrada del paquete del servidor, se lleva a cabo una prueba para verificar que el servidor se inicie sin errores. Se utiliza la función `exec` de la biblioteca `child_process` para ejecutar el comando `pnpm dev`, que inicia el servidor local. Luego, se realiza un bucle que intenta realizar una solicitud a la URL del servidor local varias veces, esperando un segundo entre cada intento. Una vez que se realiza con éxito la solicitud, se verifica que la respuesta no sea indefinida, lo que indica que el servidor se inició correctamente y está listo para recibir solicitudes. Finalmente, se detiene el servidor utilizando el `kill()` método.

Estas pruebas unitarias son solo ejemplos de las numerosas pruebas realizadas en la aplicación, que abarcan diferentes aspectos del código, desde la interfaz de usuario hasta la lógica del servidor. El uso de pruebas unitarias garantiza la estabilidad y la calidad del código, lo que contribuye significativamente a una experiencia de usuario sólida y libre de errores.

### 3.4.2. Análisis y Evaluación de la Eficacia y Eficiencia de la Aplicación

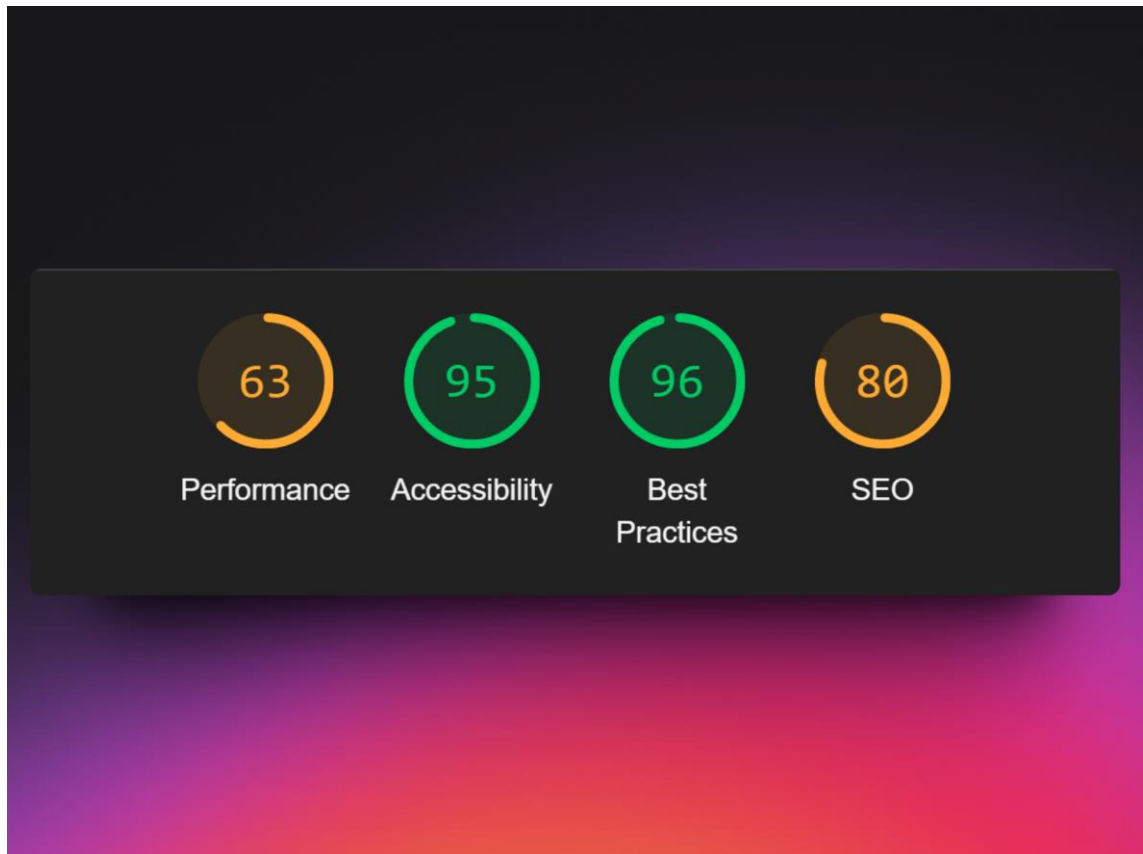
La evaluación de la aplicación se ha llevado a cabo considerando varios criterios clave:

- **Rendimiento:** Se realizaron pruebas de carga para asegurar que la aplicación puede manejar múltiples usuarios simultáneamente sin degradar el rendimiento. La implementación de WebSockets permitió una comunicación en tiempo real eficiente, mientras que la utilización de MongoDB junto con Node.js garantizó una respuesta rápida a las consultas de la base de datos.
- **Seguridad y privacidad:** La encriptación de usuarios y mensajes mediante JWT y el uso de `sessionStorage` garantizan buenos niveles de seguridad y privacidad en la aplicación.



- **Usabilidad:** La interfaz diseñada con Vue.js y Tailwind CSS fue evaluada mediante pruebas de usuario, confirmando que es intuitiva y fácil de usar. Los usuarios reportaron una experiencia fluida y satisfactoria.

Cada uno de éstos puntos se han testeado con la herramienta web PageSpeed Insights.



*Ilustración 23 - Resultado PageSpeed Insights*

Este test indica que la aplicación web tiene que mejorar el rendimiento. Esto se debe a que la aplicación se encuentra en desarrollo y hay ciertos ajustes y configuraciones que están orientados a la parte de desarrollo.

### 3.4.3. Comparativa con Otras Soluciones Existentes

Se realizó una comparativa exhaustiva con otras aplicaciones de mensajería populares para evaluar la eficacia y eficiencia de la aplicación desarrollada:

- **WhatsApp:** Aunque WhatsApp presenta una encriptación sólida y una amplia gama de funcionalidades, la aplicación desarrollada se destaca por su enfoque centrado en la seguridad y la comodidad del usuario. Si bien WhatsApp ofrece una experiencia móvil sólida, la aplicación web desarrollada proporciona una alternativa conveniente al permitir a los usuarios acceder al chat directamente desde el navegador, sin necesidad de

descargar una aplicación adicional. Además, la característica de volatilidad de los mensajes y grupos en la aplicación desarrollada garantiza una mayor privacidad y seguridad al eliminar automáticamente los mensajes después de cerrar la pestaña o cerrar la sesión en la aplicación web. Esto proporciona a los usuarios un mayor control sobre su privacidad y la confidencialidad de sus conversaciones, lo que puede ser especialmente útil en situaciones donde la seguridad es una preocupación prioritaria.

- **Telegram:** A pesar de las opciones de chats secretos y autodestrucción de mensajes ofrecidos por Telegram, la aplicación desarrollada se destaca por su enfoque en la seguridad y la accesibilidad. Al igual que Telegram, la aplicación web permite a los usuarios acceder al chat de forma rápida y conveniente desde el navegador, eliminando la necesidad de descargar una aplicación independiente. Sin embargo, lo que diferencia a la aplicación desarrollada es su característica de volatilidad de mensajes y grupos, que garantiza una mayor privacidad y confidencialidad al eliminar automáticamente los mensajes una vez que se cierra la pestaña o se sale de la sesión en el navegador. Esta funcionalidad única brinda a los usuarios un mayor control sobre sus datos y conversaciones, asegurando que la información sensible se mantenga protegida en todo momento.

## 4. Conclusiones

### 4.1. Conclusiones principales

El desarrollo de este proyecto de Trabajo de Fin de Grado (TFG) para el ciclo formativo de Desarrollo de Aplicaciones Web (DAW) ha sido una experiencia enriquecedora que ha permitido aplicar los conocimientos teóricos adquiridos durante los dos cursos a un proyecto práctico y significativo. Las principales conclusiones extraídas son las siguientes:

- Se ha logrado implementar con éxito un chat online con funcionalidades avanzadas de seguridad, incluyendo encriptación de usuarios y volatilidad de mensajes, que aborda de manera efectiva las necesidades de privacidad y confidencialidad de los usuarios en el entorno digital actual.
- La elección de tecnologías modernas y herramientas de desarrollo colaborativas, como Vue.js, Node.js, WebSockets, y Figma, ha demostrado ser acertada, permitiendo una implementación eficiente y una experiencia de usuario mejorada.
- La aplicación sigue un enfoque de microservicios, gestionados en un monorepositorio utilizando pnpm, lo que ha facilitado la integración continua y ha mejorado la coherencia del desarrollo.

## 4.2. Limitaciones y posibles mejoras

A pesar de los logros alcanzados, existen algunas limitaciones y áreas de mejora identificadas durante el desarrollo del proyecto:

- **Escalabilidad:** A medida que la aplicación crezca en usuarios y funcionalidades, podría surgir la necesidad de optimizar su arquitectura para garantizar la escalabilidad y el rendimiento óptimo del sistema.
- **Mejora de la interfaz de usuario:** Aunque se ha puesto un énfasis significativo en el diseño de la interfaz de usuario, siempre hay margen para mejorar la usabilidad y la estética de la aplicación para ofrecer una experiencia aún más atractiva y fácil de usar.
- **Pruebas exhaustivas:** Aunque se han realizado pruebas unitarias, se podría profundizar en las pruebas de integración y end-to-end para garantizar una cobertura completa del código y una mayor confiabilidad del sistema en diferentes escenarios.
- **Mejora continua:** La naturaleza dinámica del desarrollo web requiere un enfoque de mejora continua. Se deben establecer procesos para recopilar comentarios de los usuarios y realizar actualizaciones y mejoras regulares en la aplicación para mantenerla relevante y competitiva en el mercado.

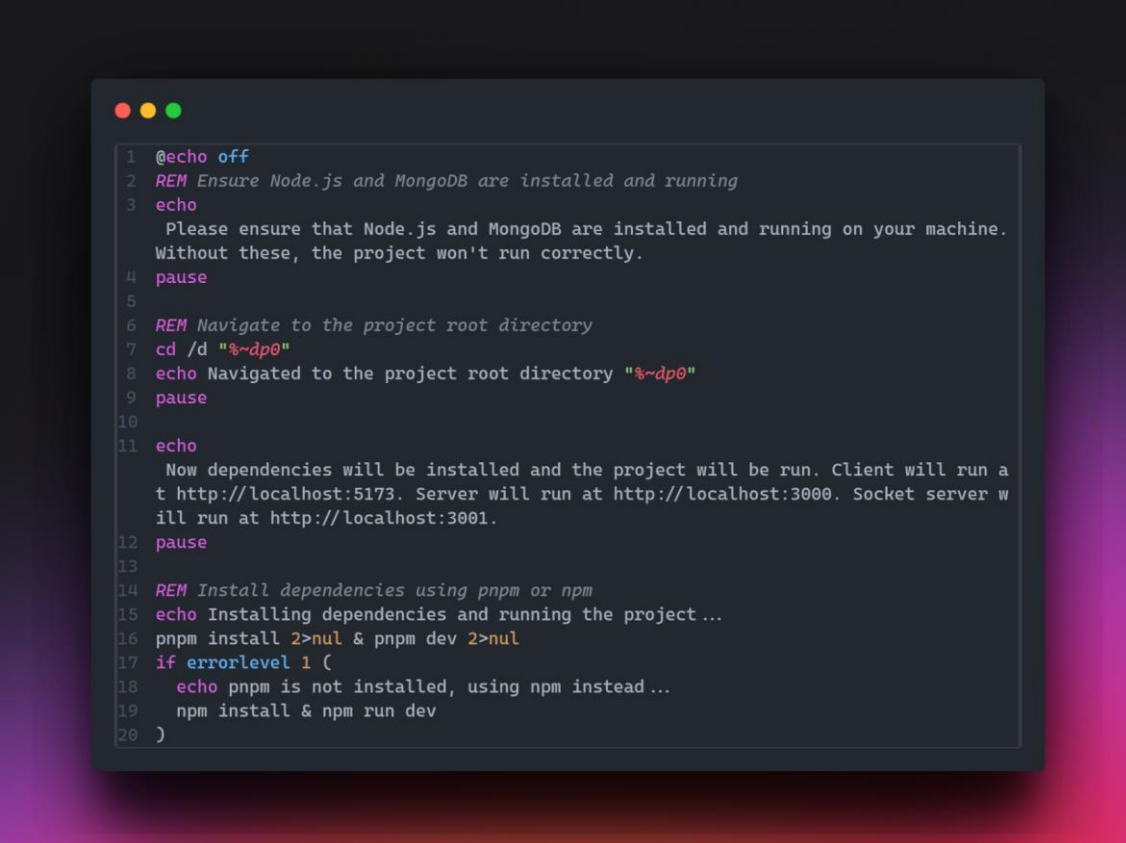
En resumen, el proyecto ha sido un éxito en términos de cumplir con los objetivos establecidos y proporcionar una experiencia de aprendizaje valiosa.

## 5. Bibliografía – Formato APA

- *Vue.js*. (s. f.). The Progressive JavaScript Framework | <https://vuejs.org/>
- *Node.js — Run JavaScript everywhere*. (s. f.). | <https://nodejs.org/>
- *Vitest*. Next Generation Testing Framework. | <https://vitest.dev/>
- MongoDB. (s. f.). *MongoDB: the Developer Data platform*. | <https://mongodb.com/>
- *Tailwind CSS - Rapidly build modern websites without ever leaving your HTML*. (s. f.). Tailwind CSS. | <https://tailwindcss.com/>
- *Figma: The Collaborative Interface Design Tool*. (s. f.). Figma. | <https://figma.com/>
- *Your connected workspace for wiki, docs & projects / Notion*. (s. f.). Notion. | <https://notion.so/>

## 6. Anexo – Ejecutar el proyecto en local

A continuación, se muestra el script `dev_up.bat`. Este script se encuentra en la raíz del monorepositorio y al ejecutarlo normalmente se ejecutará e iniciará el proyecto en la máquina local automáticamente ejecutando todos los comandos necesarios para ello. Si se tiene el gestor de paquetes `pnpm` se utilizará para la ejecución del script, y si no se tiene se utilizará `npm`.



```
1 @echo off
2 REM Ensure Node.js and MongoDB are installed and running
3 echo
4   Please ensure that Node.js and MongoDB are installed and running on your machine.
   Without these, the project won't run correctly.
5 pause
6 REM Navigate to the project root directory
7 cd /d "%~dp0"
8 echo Navigated to the project root directory "%~dp0"
9 pause
10
11 echo
12   Now dependencies will be installed and the project will be run. Client will run a
   t http://localhost:5173. Server will run at http://localhost:3000. Socket server w
   ill run at http://localhost:3001.
13 pause
14 REM Install dependencies using pnpm or npm
15 echo Installing dependencies and running the project...
16 pnpm install 2>nul & pnpm dev 2>nul
17 if errorlevel 1 (
18   echo pnpm is not installed, using npm instead...
19   npm install & npm run dev
20 )
```

Ilustración 24 - Script para ejecución del proyecto en local

Además, se incluyen una serie de pasos para ejecutar la aplicación manualmente:

- Asegurarse que se tiene el servidor local de MongoDB instalado en la máquina local y corriendo.
- Ejecutar una terminal y navegar a la raíz de la carpeta del proyecto.
- Ejecutar el comando `npm install`.
- Ejecutar el comando `npm run dev`.

## 7. Agradecimientos

Quisiera expresar mi más profundo agradecimiento a todos mis profesores por estos dos años de formación en el ciclo superior de Desarrollo de Aplicaciones Web. En particular, me gustaría destacar a Marlene, Dani y Garbiñe, quienes han sido fundamentales en mi proceso de aprendizaje. Su dedicación, paciencia y profundo conocimiento han sido clave para mi desarrollo académico y profesional. Gracias a ellos, no solo he adquirido una sólida base de conocimientos técnicos, sino que también he encontrado la claridad y la confianza necesarias para afirmar que mi verdadera vocación es el desarrollo de software. Su apoyo constante y su disposición para resolver mis dudas han sido invaluable, y por ello, les estaré eternamente agradecido.