C.P.R. LICEO LA PAZ

Desarrollo de aplicaciones web 2024/2025

Proyecto fin de ciclo: Easy Chat
Desarrollo de aplicaciones web

Autor: Mateo Fernández Rivera

Tutor: Jesús Ángel Pérez-Roca Fernández

Índice

Resumen 4

Abstract 5

Palabras clave 6

Motivación y Objetivos 7

Diagramas 9

- Base de datos 9
- Diagrama de Clases UML 12
- Diagramas de flujo 13
- Diagramas de casos de uso 26

Diseño 28

- Wireframes 28

Manual del administrador 30

- Software necesario 30
- Pasos para ejecutar la aplicación 30

Manual del usuario 31

- Inicio y registro 31
- Lista de chats 33
- Creación de chats 34
- Página de chat 37

Desarrollo del proyecto 39

- Backend con NestJS y TypeORM 39
- Aplicación multilenguaje 39
- Frontend con Material UI 39
- Concepto de la aplicación 40
- Rendimiento 40

Viabilidad tecno-económica 42

- Requisitos a nivel hardware 42
- Requisitos de software 42
- Plataformas de despliegue posibles 42
- Resumen de costos 44

Trabajo a futuro 45

Conclusiones 46
Recursos web utilizados 47
Agradecimientos 48

Resumen

Este proyecto consiste en el desarrollo completo de una aplicación web de mensajería instantánea denominada **Easy Chat**, inspirada en plataformas similares como WhatsApp o Telegram, integrando múltiples tecnologías modernas tanto en la parte *frontend* como en *backend*.

La aplicación permite a los usuarios registrar una cuenta, iniciar sesión y comunicarse entre ellos mediante **salas o chats grupales** o **privados** en **tiempo real**. Cada chat soporta diversas funcionalidades tales como el envío y recepción de mensajes, visualización de mensajes antiguos, gestión de participantes y la eliminación de mensajes, todo con una interfaz intuitiva y adaptable a todo tipo de tamaños de pantalla y dispositivos.

La parte del *frontend* fue desarrollada con *React* utilizando *Vite* para una carga rápida y eficiente, junto con *Material UI*, una librería de componentes estilizados y responsivos. La lógica de comunicación en tiempo real se gestiona con *Socket.IO*, permitiendo que los usuarios envíen y reciban mensajes instantáneamente a los demás miembros de la conversación.

El *backend* está construido con *NestJS*, un *framework* modular y escalable basado en *Node.js*. *TypeORM* es usado para la gestión de datos, trabajando con *MySQL* en producción y *SQLite* en desarrollo. El sistema implementa *Websockets*, control de sesiones con *express-session* y una arquitectura organizada en entidades necesarias para registrar usuarios, crear y gestionar chats, enviar y eliminar mensajes y las relaciones necesarias entre estos.

Abstract

This project consists of the development of a web-based instant messaging application called Easy Chat, inspired by similar platforms such as WhatsApp or Telegram, integrating multiple modern technologies in both the frontend and backend.

The application allows users to register an account, log in, and communicate with each other through private or group rooms in real time. Each chat supports various functionalities such as sending an receiving messages, viewing past messages, managing participants, and deleting messages, all with an intuitive interface that adapts to all type of screen sizes and devices.

The frontend was developed with React using Vite for fast and efficient loading, along with Material UI, a library of stylized and responsive components. Real-time communication logic is managed with Socket.IO, allowing users to send and receive messages instantly with other members of the conversation.

The backend is built with NestJS, a modular and scalable framework based on Node.js. TypeORM is used for data management, working with MySQL in production and SQLite in development. The system implements WebSockets, session control with express-session, and an architecture organized into the necessary entities to register users, create and manage chats, send and delete messages, and the required relationships between them.

Palabras clave

Este es un listado de palabras importantes que describen el proyecto. Pueden incluirse ideas, tecnologías, conceptos, etc, y cada una contiene una breve descripción.

- **Chat**: Intercambio de mensajes mediante un software entre dos o más personas o usuarios conectados a una red.
- **Usuario**: Una o varias personas que utilizan un producto o servicio de forma habitual.
- **Instantáneo**: Que se produce inmediatamente.
- **Social**: Perteneciente o relativo a la sociedad.
- **Mensaje**: El objeto de la comunicación que envía el emisor a través de un canal de comunicación.
- **Conexión**: Unión o enlace entre dos o más cosas.
- **Simple**: Sin complicación ni dificultad.
- **Accesible**: De fácil acceso o uso.
- **Seguro**: Que no supone ningún tipo de riesgo.
- **Escalable**: Que puede ser mejorado en el futuro
- **Rendimiento**: Proporción entre el resultado obtenido y los medios utilizados.
- **UI y UX**: Interfaz de usuario y experiencia de usuario respectivamente.
- **Responsive**: Un tipo de diseño web que tiene como objetivo hacer que un sitio web sea accesible y adaptable en todos los dispositivos.
- **Open Source**: Código fuente que se pone libremente disponible para su posible modificación y redistribución
- **Websocket**: Tecnología que permite crear una conexión bidireccional entre un cliente y un servidor.
- **JavaScript**: Lenguaje de programación de scripting que permite añadir interacciones dinámicas a páginas y aplicaciones web.
- **TypeScript**: Lenguaje de programación desarrollado por Microsoft que añade tipados estáticos y objetos basados en clases.
- **NestJS**: Framework de desarrollo web que utiliza TypeScript para crear aplicaciones del lado del servidor.
- **React**: Librería JavaScript open source utilizada para construir interfaces de usuario interactivas.
- **Material UI (MUI)**: Biblioteca de componentes de UI para React que implementa los principios de diseño de Material Design de Google.
- **Vite**: Servidor local que sirve como herramienta de desarrollo para la creación de aplicaciones web.
- **BBDD**: Siglas para base de datos.

Motivación y Objetivos

Desde muy pequeño estoy rodeado constantemente de tecnología y, algo que siempre me llamó la atención eran las redes sociales, foros, blogs, etc. ¿Cómo interactúan entre sí los usuarios de internet? ¿Qué son los requisitos para que estas interacciones sean así? Son preguntas que siempre me habían llamado la atención, y supe que este proyecto debía ser sobre eso.

La idea o concepto esencial es la de crear una aplicación web **que pudiese conectar a la gente** de forma **fácil, sencilla**, pero sobre todo **rápida** y crear un entorno en el que estos puedan relacionarse. Para cumplir con esta meta pensé en varios ejemplos de aplicaciones sociales en las que los usuarios puedan conectarse entre sí.

- **Red social estilo *microblogging***: Una red social en la que cada usuario tiene un perfil que va actualizando con entradas de texto generalmente breves y salen en las *timelines* de otros usuarios de la plataforma. Inspiración: BlueSky y X/Twitter.
- **Aplicación de mensajería instantánea**: Una aplicación en la que un usuario puede crearse una cuenta y comunicarse con otros usuarios mediante chats privados o grupales. Inspiración: Facebook Messenger, WhatsApp y Telegram.
- **Red social *artsharing***: Una red social para artistas y difusión de arte, ya sea fotográfico, musical, literario, etc. Inspiración: Pinterest, Soundcloud y DeviantArt.
- **Sitio de juegos *online***: Un sitio con diversos juegos en línea de navegador con tablas de puntuación y rankings semanales. Inspiración: Friv, Kizi y Minijuegos.

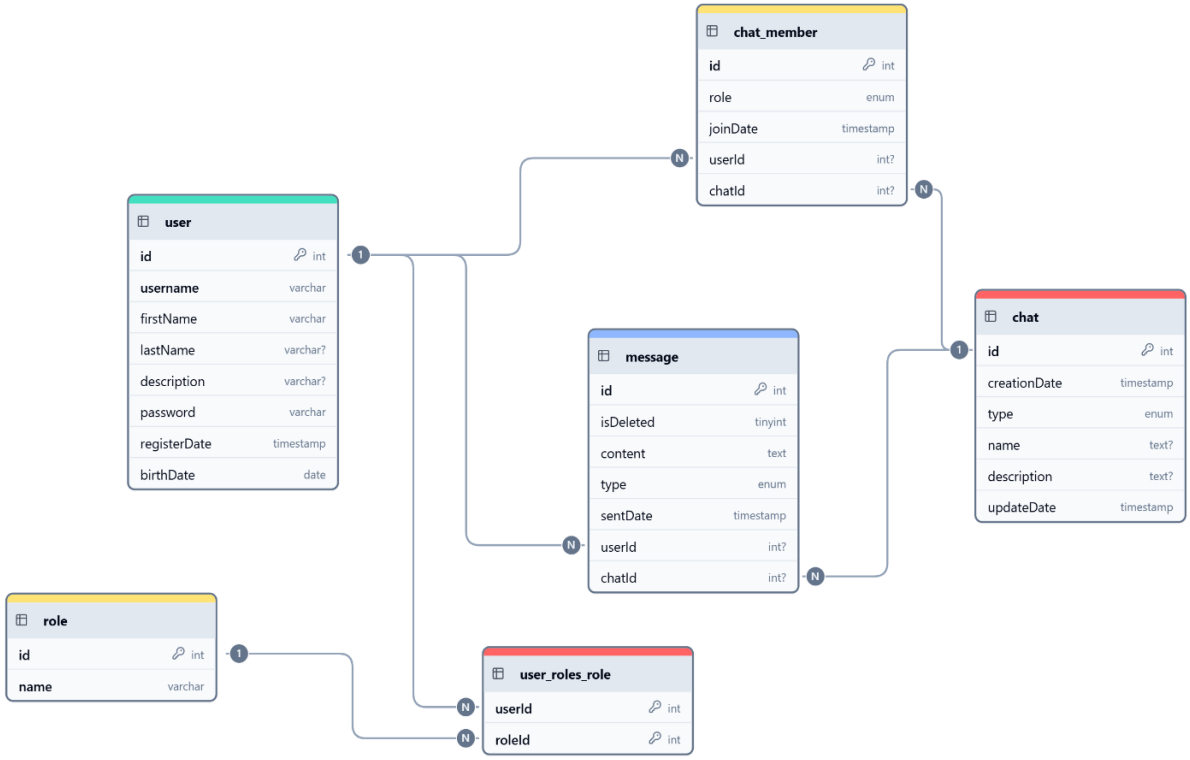
Estas son las ideas que más me llamaban en un principio, ya que en cierta manera, los usuarios podían conectar entre sí mediante distintas formas, así que empecé a **descartar ideas**. La primera que descarté fue la de los juegos, ya que en la actualidad la popularidad de los sitios de juegos de navegador es muy baja y no veía una interacción entre los jugadores tal como para crear una plataforma basada en eso. La siguiente idea descartada era la red social de difusión de arte por falta de interés e ideas. No me convenció la idea de la red social estilo *microblogging* porque la aplicación de mensajería instantánea englobaba todo lo que quería que tuviese mi aplicación web: una plataforma fácil, sencilla, rápida y segura. Esta idea fue la que más seguridad me dio sobre las otras tres.

El concepto general de una aplicación de mensajería instantánea es, como bien dice el nombre, una **plataforma de envío y recepción de mensajes de forma inmediata**. Para ello, me propuse los siguientes objetivos.

1. Que los usuarios pudiesen **registrar una cuenta de usuario** que a su vez pudiese servir de perfil público incluyendo:
    a. Un **nombre de usuario único e irrepetible**.
    b. Una biografía o descripción del usuario.
2. Para que los usuarios puedan usar su cuenta deben de **iniciar sesión** usando:
    a. El **nombre de usuario**.
    b. Una **contraseña**.
3. La **seguridad** de los usuarios es la máxima prioridad dentro de la plataforma, así que era **vital** encontrar una forma de **encriptar la contraseña** que fuese **robusta** y que no fuese pública ni accesible.
4. La creación de **chats privados**. Un usuario puede crear un chat privado con otro usuario y este solo debería ser accesible por ambos usuarios.
5. **Chats grupales** los cuales pueden tener dos o más miembros con permisos distribuidos en una jerarquía de **roles**. Dichos chats pueden tener un nombre y descripción que pueden ser editadas por ciertos miembros del grupo.
6. El envío y recepción de mensajes debería realizarse **a tiempo real**, esta es, la característica más importante de esta aplicación, de no serlo, no tendría sentido.
7. La posibilidad de poder **eliminar mensajes** propios y que a los otros usuarios dentro del chat no les aparezca su contenido marcándolo como borrado, pero **no borrándolo del todo en la BBDD**.
8. Hacer la aplicación **lo más accesible posible** e **internacionalizarla**, para ello, es importante que la plataforma **esté disponible en distintos idiomas**.
9. Que haya usuarios que puedan moderar toda la plataforma y puedan leer **mensajes o denuncias** realizadas por los usuarios y que estos puedan tomar una decisión.
10. Que la aplicación sea **lo más sencilla posible** y que su utilización sea **cómoda y rápida** y **flexible** a todo tipo de tamaños de pantalla o dispositivos, sean ordenadores, tablets, teléfonos…

Diagramas

Base de datos



Este es el diagrama de la base de datos del proyecto. Incluye tablas para usuarios, roles de usuario, roles, chats, miembros de chats y mensajes.

USER: Guarda la información de los usuarios de la plataforma.

- Id es la clave primaria de la tabla de usuarios, un número que identifica de manera única a cada uno de ellos.
- Username es el nombre de usuario que se hace público y pueden ver todos, se usa para iniciar sesión y es **único e irrepetible**.
- Las columnas first name y last name son para el nombre y apellidos del usuario, estas tablas son datos personales del usuario y **no son públicas**.
- Description es la descripción pública que un usuario puede poner a modo de biografía en su perfil, se supone que es público para todos
- Password es la contraseña del usuario que usa para iniciar sesión en su cuenta, está **encriptada en la BBDD** y no debe ser **totalmente privada**.
- RegisterDate guarda la fecha y hora en la que el usuario creó su cuenta dentro de la plataforma.
- BirthDate es la fecha de nacimiento del usuario, el backend debe asegurarse de que los usuarios que se registran son **mayores de edad**.

ROLE: Tabla que define qué roles están disponibles para los usuarios. El backend cada vez que se ejecuta verifica si hay roles insertados, si no los hay, los inserta él mismo mediante un archivo del servidor.

- **Id:** Clave foránea de la tabla.
- **Name:** Nombre del rol.

USER_ROLES_ROLE: Tabla intermedia o *join table* en inglés. Relaciona los roles que puede tener un usuario. Cada fila indica que un usuario tiene un determinado rol, haciendo que un usuario pueda tener **múltiples** roles.

- **userId:** Este es el id de del usuario.
- **roleId:** Este es el id del role.

CHAT: Tabla que define los chats que hay creados en la plataforma. Cada chat puede tener un tipo determinado, un nombre o una descripción.

- **Id:** Clave foránea de la tabla.
- **CreationDate:** Guarda fecha y hora en la que se creó el chat
- **Type:** Define de qué tipo es el chat, pudiendo ser 'private' (privado) o 'group' (grupal).
- **Name:** Nombre del chat, utilizado únicamente para los chats grupales ya que normalmente el nombre de un chat privado es el username del otro miembro del chat.
- **Description:** Descripción del chat, utilizado únicamente para los chats grupales ya que normalmente la descripción de un chat privado es el description del otro miembro del chat.
- **UpdateDate:** Guarda la fecha y hora la cual el chat fue modificado por última vez.

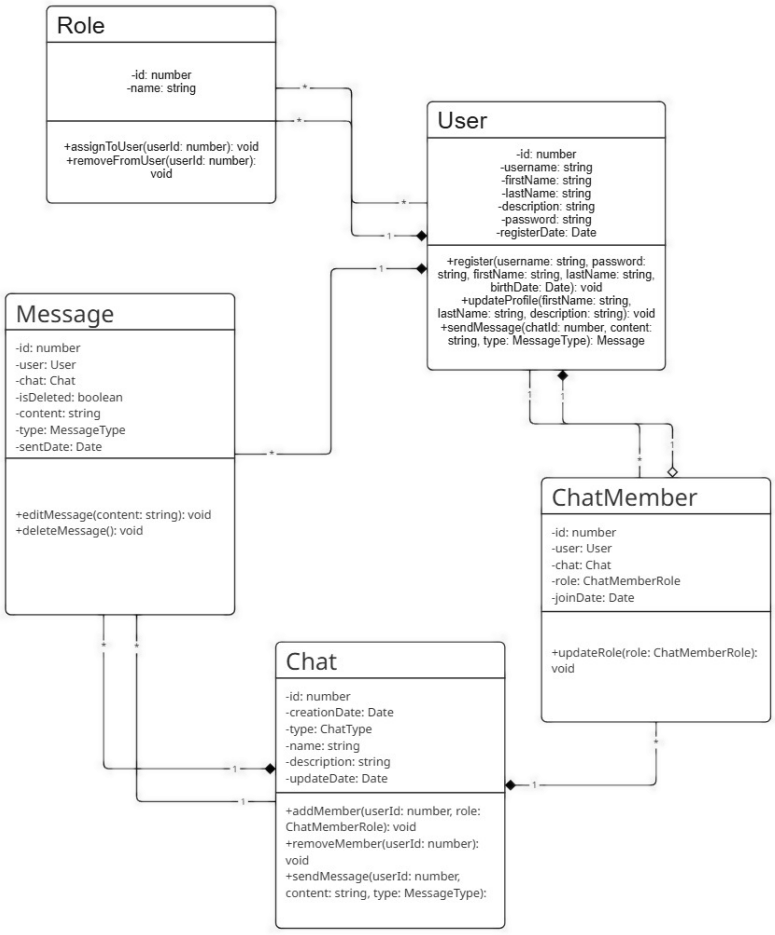
CHAT_MEMBER: Guarda los miembros de un chat. Un chat puede tener uno o varios usuarios como miembros. Tiene una **restricción** que **impide guardar** más de una fila con la **misma combinación de user id y chat id**, evitando duplicados como por ejemplo insertar varias veces a un usuario en el mismo chat.

- **Id:** Clave foránea de la tabla.
- **UserId:** Id del usuario.
- **ChatId:** Id del chat.
- **Role:** Rol del miembro dentro del chat
- **JoinDate:** Guarda la fecha y hora la cual el usuario fue añadido al chat.

MESSAGE: Los mensajes que envían los usuarios a un chat. Un mensaje necesita de un chat para existir, si el chat es eliminado, sus mensajes también. Sin embargo, si un usuario elimina su cuenta el user id del mensaje pasará a ser null.

- **Id:** Clave foránea de la tabla.
- **Content:** Contenido del mensaje de texto
- **Type:** Marca el tipo del mensaje, si es 'text' es de un usuario y si es 'system' entonces es enviado por el sistema.
- **SentDate:** Guarda la fecha y hora la cual un mensaje es enviado y guardado en la BBDD.
- **IsDeleted:** Booleano que marca si un mensaje está eliminado o no.
- **UserId:** El id del usuario que envía el mensaje. Puede estar 'null' si el usuario que envía el mensaje deja de existir o si el mensaje es tipo 'system'
- **ChatId:** El id del chat en el cual el mensaje fue enviado.

Diagrama de Clases UML

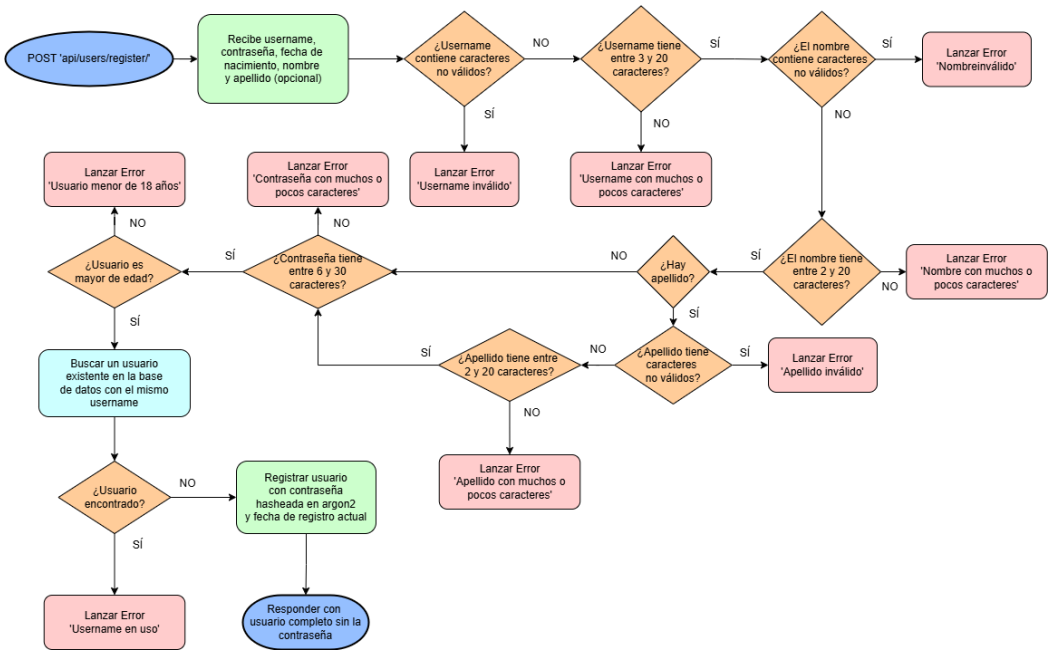


Relaciones entre clases:

- **User y Role**: Un usuario puede tener varios roles y un rol puede tener varios usuarios.
- **User y Message**: Un usuario puede enviar muchos mensajes y un mensaje es enviado por un usuario.
- **User y ChatMember**: Un usuario puede ser miembro de varios chats.
- **Chat y Message**: Un chat puede tener varios mensajes y un mensaje es enviado a un chat.
- **Chatmember y User**: Cada participación pertenece a un solo usuario.
- **Chatmember y Chat**: Cada participación pertenece a un solo chat.

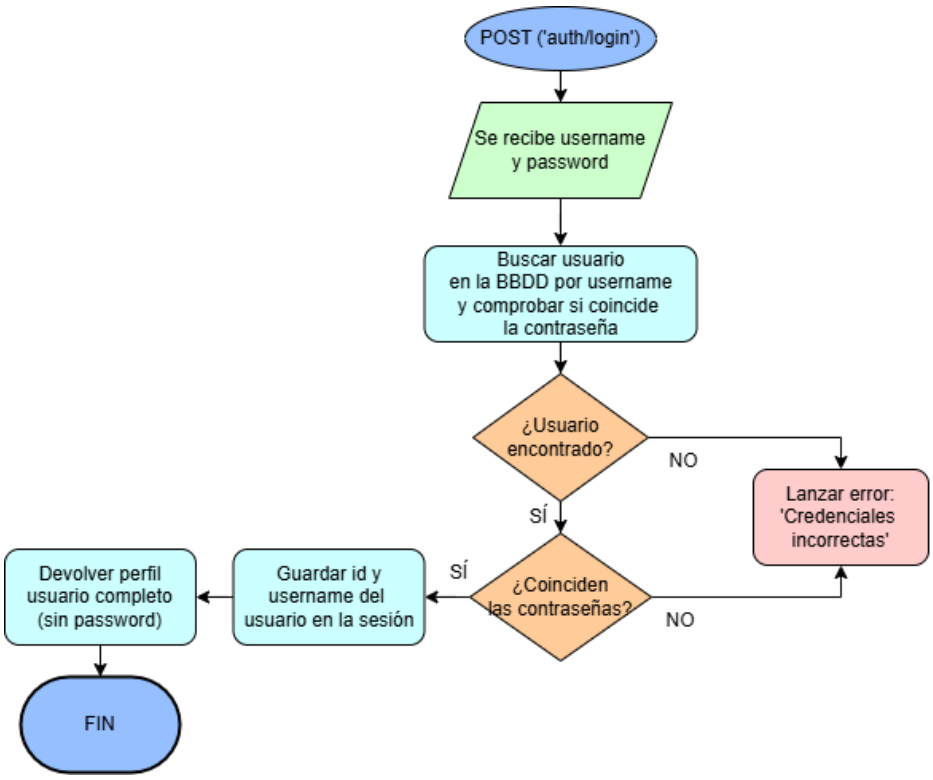
Diagramas de flujo

Registro de usuarios:



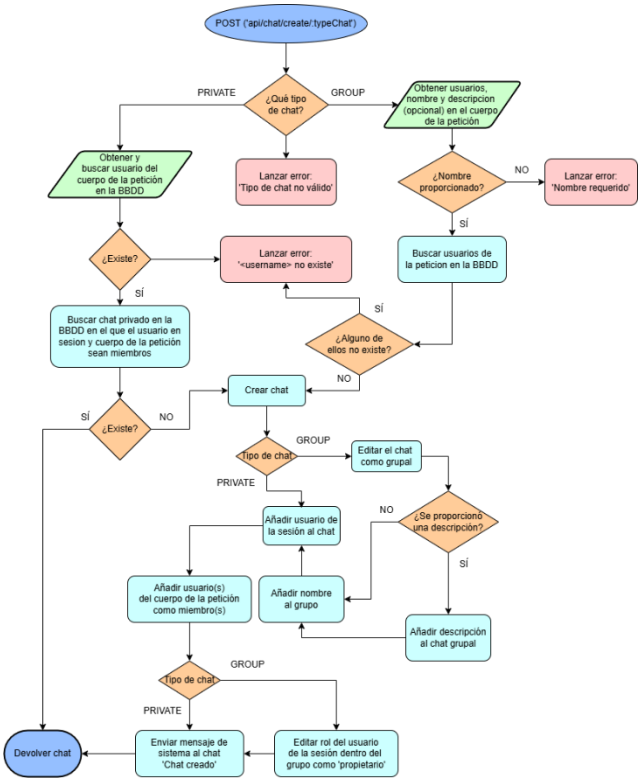
1. Recibir username, contraseña, fecha de nacimiento, nombre y apellido (opcional)
2. Si el username contiene caracteres no permitidos, lanzar error.
3. Si no, lanzar error si el nombre de usuario contiene menos de 3 o más de 20 caracteres.
4. Si no, lanzar error si el nombre contiene caracteres no permitidos.
5. Si no los tiene, lanzar error si el nombre no contiene entre 20 y 2 caracteres.
6. Si el apellido no ha sido proporcionado, saltar al paso 10.
7. Si es proporcionado, comprobar si contiene caracteres no permitidos
8. Si los tiene, lanzar error.
9. Si no, lanzar error si no contiene entre 20 y 2 caracteres.
10. Comprobar si la contraseña tiene entre 6 y 30 caracteres.
11. Si excede o es menor a ese rango, lanzar error.
12. Si tiene ese número de caracteres, revisar edad del usuario.
13. Si la fecha de nacimiento proporcionada es menor a 18 años, lanzar error.
14. Si es mayor de edad, buscar un usuario guardado en la BBDD con el mismo username
15. Si ese usuario existe, lanzar error.
16. Si no existe, entonces guardar usuario en la BBDD hasheando la contraseña en argon2 y guardando la fecha de registro.
17. Devolver perfil completo al cliente sin incluir la contraseña.

Inicio de sesión:



1. Obtener en el cuerpo de la respuesta el username y la contraseña.
2. Buscar usuario en la base de datos con el mismo username y comprobar que las contraseñas coinciden.
3. Si el usuario no existe, lanzar error de credenciales inválidas.
4. Si existe, comprobar contraseñas.
5. Si las contraseñas no coinciden, lanzar error de credenciales inválidas.
6. Si la contraseña es correcta, guardar el id y username del usuario en la sesión.
7. Devolver el perfil del usuario completo sin la contraseña.

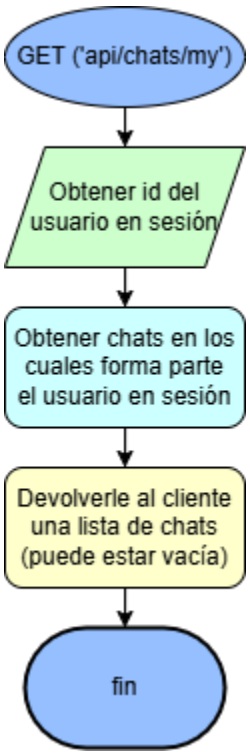
Crear chat:



1. Leer el tipo de chat a crear.
2. Si el tipo no es ‘private’ ni ‘group’ lanzar error.
3. Si el tipo de chat a crear es **private**: obtener usuario guardado en la sesión y usuario en el cuerpo de la petición (para **group**, saltar al paso 13)
4. Buscar el usuario del cuerpo de la petición en la base de datos.
5. Si el usuario no es encontrado, lanzar error.
6. Si el usuario sí existe, buscar un chat privado ya existente en el que el usuario de la sesión y este sean miembros.
7. Si existe, devolverlo al cliente y finalizar flujo.
8. Si no existe, crear chat en la BBDD.
9. Añadir usuario de la sesión al chat.
10. Añadir usuario del cuerpo de la petición al chat.
11. Enviar mensaje del sistema sobre creación del chat.
12. Finalizar flujo para private
13. Si el tipo de chat a crear es **group**: obtener usuario guardado en la sesión, usuarios en el cuerpo de la petición, nombre del chat y descripción (opcional).
14. Si el nombre no es proporcionado en el cuerpo de la petición, lanzar error.
15. En cambio, si es proporcionado, proporcionados en el cuerpo de la petición en la base de datos.
16. Si uno de ellos no existe, lanzar error.
17. Si todos existen crear chat.

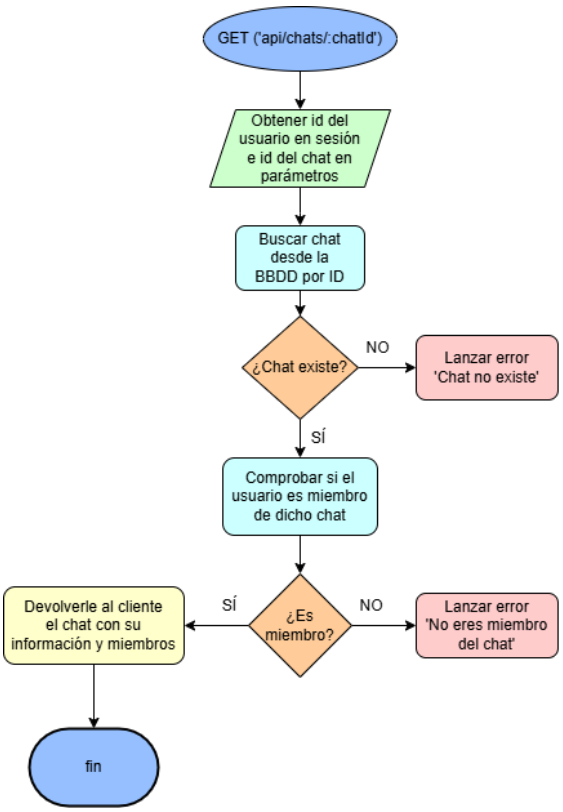
18. Cambiar tipo de chat a grupal.
19. Si se ha proporcionado una descripción, añadirla al chat grupal.
20. Si no es así, dejarla vacía.
21. Añadir nombre proporcionado como nombre de chat.
22. Añadir usuario de la sesión y los de la petición al chat como miembros.
23. Asignarle al usuario de la sesión el rol de propietario.
24. Enviar mensaje del sistema sobre creación del chat.
25. Finalizar flujo para group.

Obtener chats:



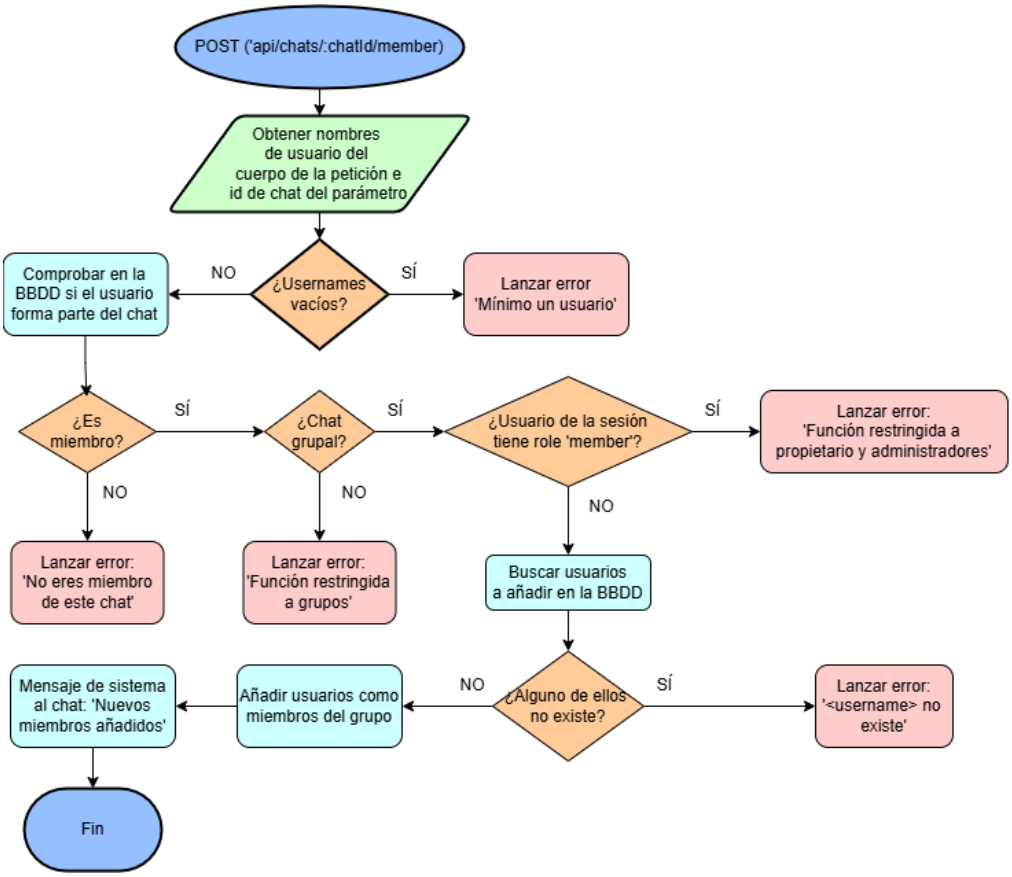
1. Obtener id del usuario guardado en la sesión.
2. Obtener chats en los cuales es miembro.
3. Devolver al cliente la lista de chats.

Obtener chat:



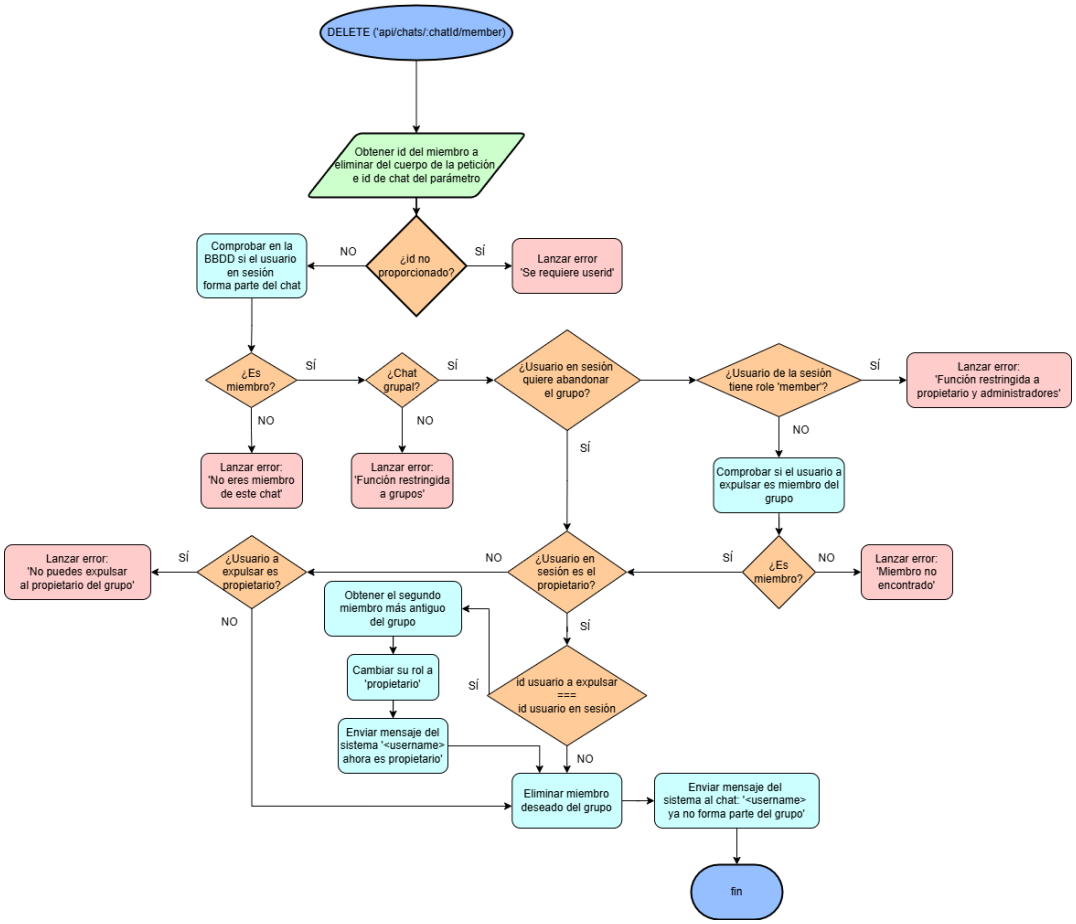
1. Obtener id del usuario guardado en la sesión e id del chat en los parámetros.
2. Buscar chat desde la base de datos por id.
3. Sí el chat no existe, lanzar error.
4. Si existe, comprobar que el usuario en la sesión es miembro de este.
5. Si no es miembro lanzar error.
6. Si es miembro devolverle al cliente el chat con su información y miembros.

Añadir miembros:



1. Obtener nombres de usuario del cuerpo de la petición, id del usuario guardado en la sesión e id del chat del parámetro.
2. Si los usernames en el cuerpo de la petición están vacíos, lanzar error.
3. Si no lo están, comprobar si el usuario de la sesión forma parte del chat.
4. Si no es un miembro, lanzar error.
5. Si lo es, comprobar si es un chat grupal.
6. Si no es grupal, lanzar error.
7. En caso de ser grupal, comprobar si el usuario de la sesión tiene el role de ‘member’
8. En ese caso, lanzar error, ya que los miembros con rol ‘member’ no tienen permitido añadir usuarios nuevos al chat grupal.
9. Si su rol no es ‘member’, comprobar si ya son miembros del chat.
10. Si alguno de ellos ya lo es, lanzar error.
11. Si ninguno de ellos no es miembro del chat, agregarlos como miembros.
12. Enviar mensaje de sistema de ‘nuevos miembros añadidos’.

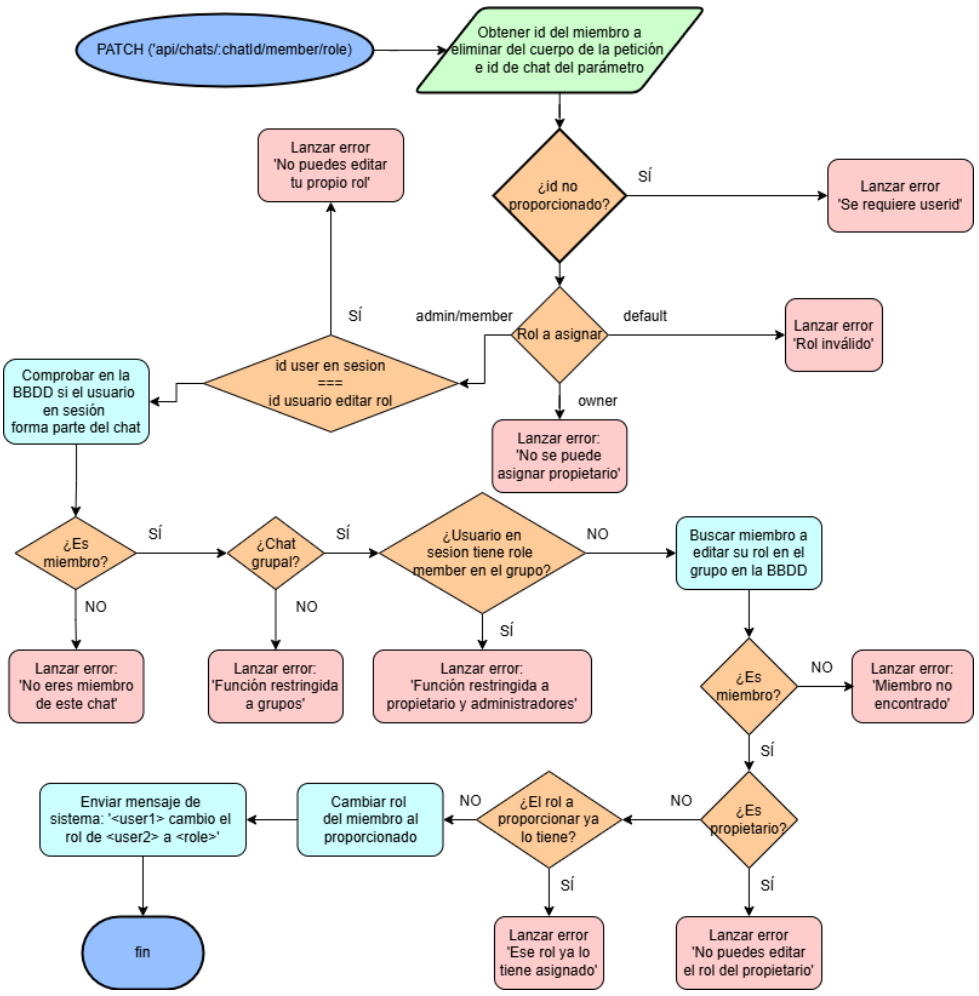
Eliminar miembro:



1. Obtener ID del miembro eliminar del cuerpo de la petición e ID de chat del parámetro.
2. Si el id no ha sido proporcionado, lanzar error.
3. Si fue proporcionado, comprobar en la base de datos si el usuario guardado en la sesión forma parte del chat.
4. Si no forma parte del chat, lanzar error.
5. Si forma parte del chat, comprobar si este es grupal.
6. Si no es grupal lanzar error.
7. En caso de serlo es comprobar si el usuario en la sesión quiere abandonar el grupo.
8. Si quiere abandonarlo, comprobar si es el propietario del grupo. (para la expulsión de otros usuarios, saltar al paso 14)
9. En caso de serlo, obtener el miembro más antiguo del grupo.
10. Cambiar su rol a propietario.
11. Enviar mensaje del sistema notificando sobre el nuevo propietario del grupo.
12. Eliminar el usuario de la sesión como miembro del grupo.
13. Enviar mensaje del sistema al grupo notificando el abandono.

14. Si el usuario de la sesión quiere abandonar el grupo, pero no es propietario, eliminarlo del grupo sin cambiar el rol de los demás miembros.
15. Enviar mensaje del sistema al grupo notificando el abandono.
16. Si el usuario de la sesión quiere eliminar a otro miembro, comprobar que este no tenga el role ‘member’
17. Si lo tiene, lanzar error debido a que no tiene permiso para expulsar usuarios.
18. Si tiene otro rol, comprobar si dicho usuario es miembro del grupo.
19. En caso de no serlo, lanzar error.
20. Si lo es, entonces verificar si el usuario en la sesión es propietario.
21. Si no lo es, comprobar si el usuario que está tratando de expulsar sí lo es.
22. En caso de serlo lanzar error.
23. Si no lo es, expulsarlo del grupo.
24. Enviar mensaje del sistema al chat notificando la expulsión.

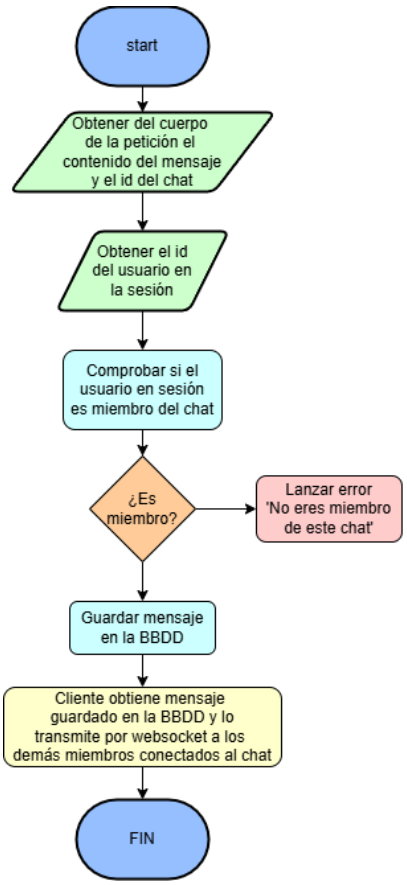
Editar el rol de un miembro en un grupo.



1. Obtener id del miembro a editar su rol del cuerpo de la petición e id de chat del parámetro.
2. Si el ID no está proporcionado lanzar error.
3. Si es proporcionado, verificar qué rol se asigna.
4. Si se asigna uno que no es válido, lanzar error.
5. Si se quiere asignar owner (propietario) lanzar error.
6. Si se quiere asignar member u owner, comprobar que el usuario en la sesión no está queriendo cambiar su propio rol.
7. Si el propio usuario está tratando de cambiar su rol, lanzar error.
8. Si no es así, comprobar que el usuario de la sesión forma parte del chat.
9. Si no es miembro lanzar error.
10. Si lo es, comprobar si es grupal.
11. Si no es grupal lanzar error.
12. Si es grupal, asegurarnos que el usuario de la sesión no tiene el rol ‘member’.
13. En caso de serlo, lanzar error.
14. Si tiene otro rol, buscar al miembro a editar su rol en el grupo.
15. Si no es miembro lanzar error.

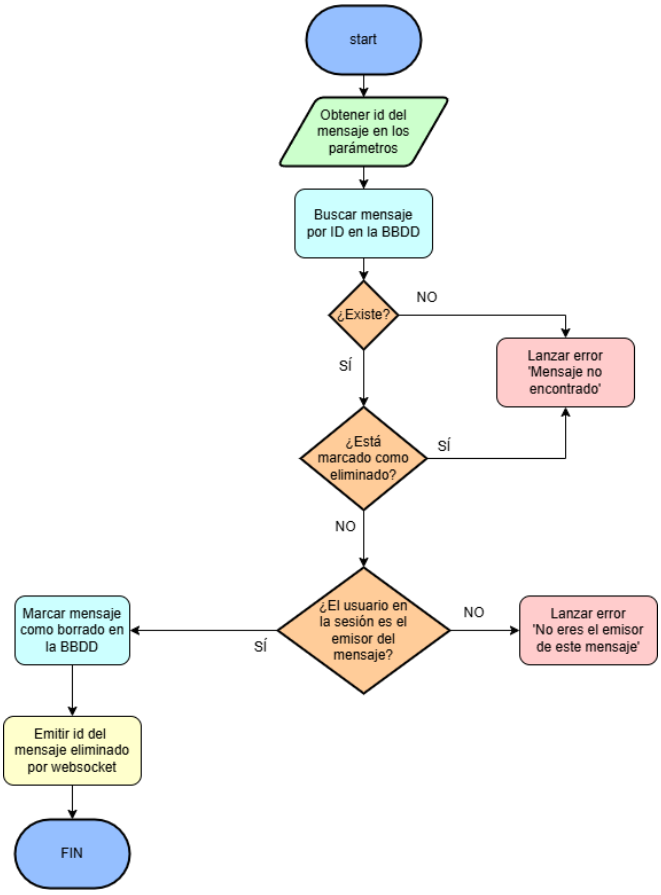
16. Si lo es, comprobar si es propietario.
17. Si es propietario, lanzar error.
18. En caso de no serlo, comprobar si el rol que se quiere asignar ya lo tiene ese miembro.
19. Si ya lo tiene, lanzar error.
20. Cambiar rol de dicho miembro al deseado.
21. Enviar mensaje de sistema al chat notificando sobre el cambio de rol.

Enviar mensaje:



1. Obtener del cuerpo de la petición el contenido del mensaje y el id del chat al que se va a enviar.
2. Obtener el ID del usuario en la sesión.
3. Comprobar si el usuario guardado en la sesión es miembro del chat.
4. Si no lo es, lanzar error.
5. Si lo es, guardar mensaje en la base de datos.
6. Obtener mensaje guardado en la base de datos y transmitirlo por websocket a tiempo real a los demás miembros conectados al chat.

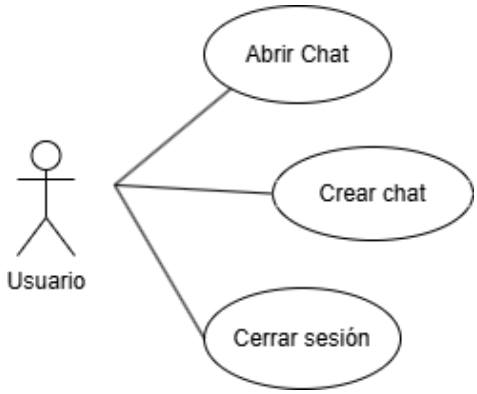
Eliminar mensaje:



1. Obtener de los parámetros el ID del mensaje que se quiere eliminar.
2. Buscar mensaje en la base de datos.
3. Si no existe, lanzar error.
4. Si existe, lanzar error si ya está marcado como eliminado.
5. En caso de no estarlo, verificar si el usuario en la sesión es el emisor del mensaje.
6. Si no lo es, lanzar error.
7. Si lo es, marcar mensaje como borrado en la base de datos.
8. Emitir ID del mensaje eliminado por websocket para que los clientes en el chat puedan actualizarlo a tiempo real.

Diagramas de casos de uso

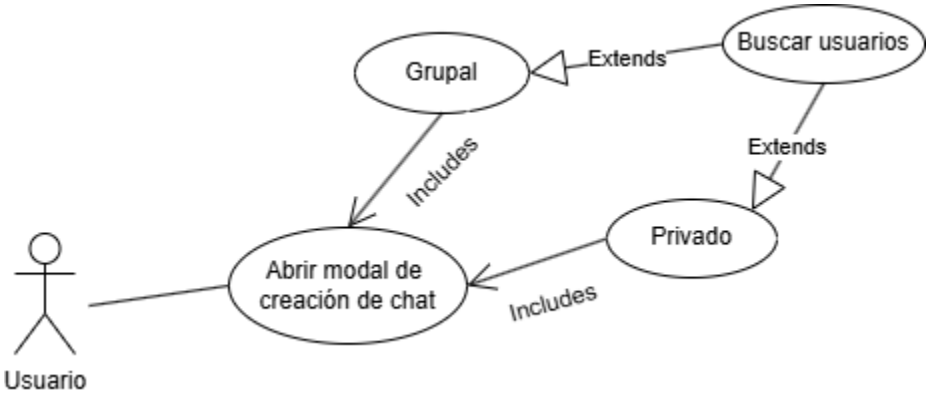
Página principal:



Un usuario desde la página principal con el listado de los chats puede:

1. Abrir un chat.
2. Crear un chat.
3. Cerrar la sesión.

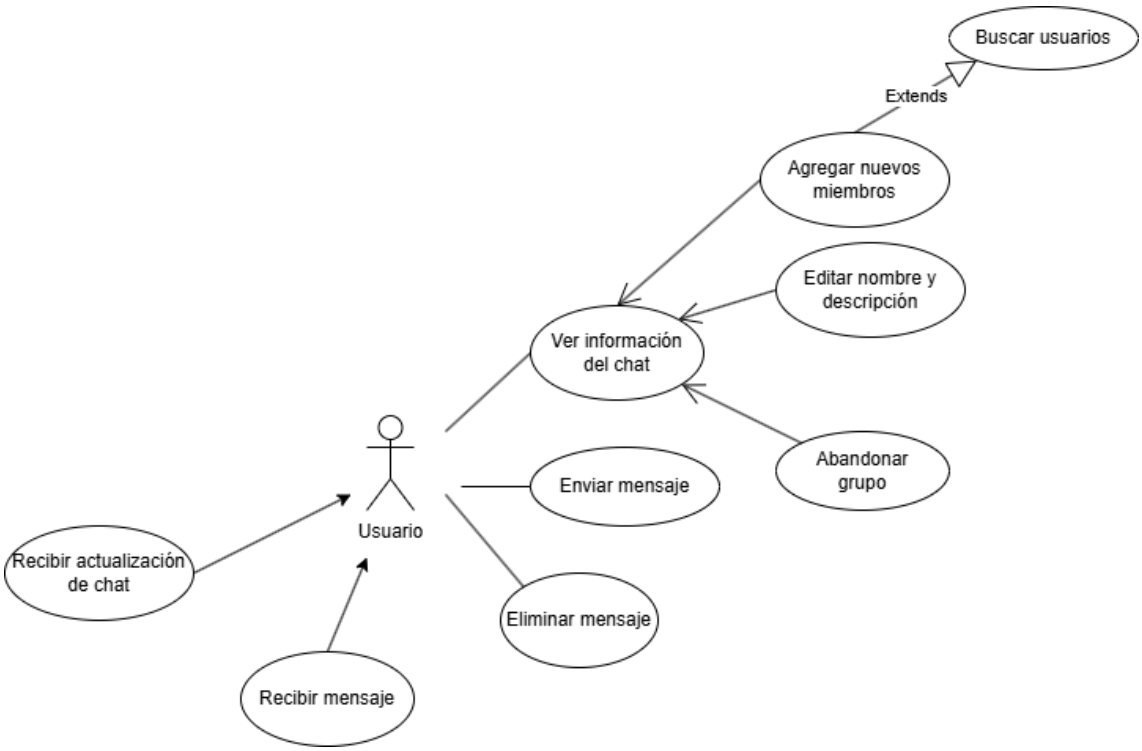
Modal de creación de chat:



Al abrir el modal de creación de chat, el usuario puede

1. Elegir si lo quiere crear de tipo privado o grupal.
2. En cualquier caso, deberá buscar el usuario o usuarios con los que quiera crear la conversación.

Página de chat:

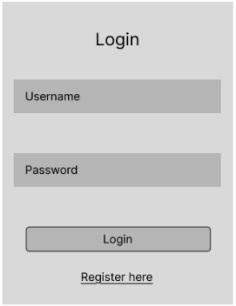


En la página de un chat, un usuario podrá.

1. Enviar un mensaje al chat.
2. Eliminar un mensaje suyo.
3. Ver la información del chat, y ahí también podrá:
    a. Abandonarlo (en caso de ser un grupo).
    b. Editar nombre y la descripción (en caso de ser administrador o propietario y si es un grupo).
    c. Agregar nuevos miembros, que incluye además el tener que buscarlos en la BBDD (en caso de ser administrador o propietario y si es un grupo).
4. Recibir mensajes de otros usuarios a tiempo real y de forma inmediata.
5. Recibir actualizaciones de chat, ya sea cuando un nuevo miembro es añadido o cuando su nombre y/o descripción han sido editados.

Diseño

Wireframes




EASY CHAT
Chat 1
User: last message goes here...
12:09
Chat 2
User: last message goes here...
14:02
Chat 3
User: last message goes here...
Yesterday
+
<
Chat 1
Today
Blah Blah Blah Blah Blah...
Blah Blah Blah Blah Blah...
Send message...
SEND

Manual del administrador

Este es el manual con todos los requerimientos y pasos a seguir para ejecutar Easy Chat en cualquier máquina.

Software necesario

1. Docker compose o Docker Desktop para levantar el servidor y el backend.
2. Node.js, imprescindible para ejecutar el frontend (recomiendo la versión número 20)
3. Git, para poder clonar el repositorio de GitHub en el cual se haya el código.
4. Tener los puertos 3000, 3306, 8080 y 5173 liberados.

Pasos para ejecutar la aplicación

1. Debes clonar el repositorio en tu máquina haciendo uso de git.
   ```
   git clone https://github.com/mateonation/ProxectoFinalDeCiclo-MFR.git
   ```

2. Ve a la raíz del directorio y ejecuta el siguiente comando para crear los contenedores de Docker.
   ```
   docker-compose up --build -d
   ```

3. Después de que los contenedores se inicializaran de forma correcta, ve a la raíz de *frontend.*
   ```
   Cd frontend
   ```

4. Instala los módulos de node del proyecto.
   ```
   npm install
   ```

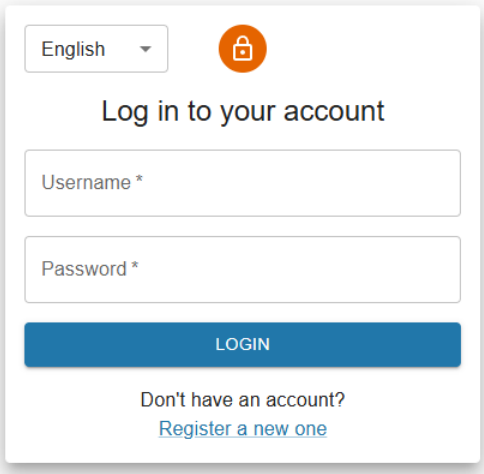
5. Ejecuta el proyecto en un servidor de Vite.
   ```
   npm run vite
   ```

6. ¡Listo! Ahora puedes acceder a Easy Chat desde tu máquina local.
   ```
   http://localhost:5173/register
   ```

Manual del usuario

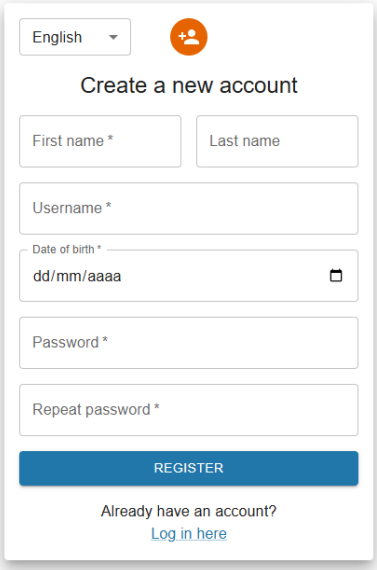
Inicio y registro

/EasyChat/login



Esta es la página de login o inicio de sesión. Para iniciar sesión es necesario tener una cuenta de usuario ya creada en el sitio e iniciar sesión con el nombre de usuario y contraseña

/EasyChat/register



Esta es la página de registro, aquí puedes crear una cuenta con tu nombre, apellido (opcional), un nombre de usuario único, una fecha de nacimiento y una contraseña. Datos como el nombre, los apellidos, la fecha de nacimiento son privados y en no son públicos para el resto de los usuarios. Para validar la contraseña, te pedimos que la introduzcas dos veces y que sea de 6 a 30 caracteres.

El nombre de usuario solo podrá contener letras, números y guiones bajos, cualquier otro tipo de carácter es inválido. Dicho nombre de usuario es único e irrepetible, así que te pedimos que uses uno que no use nadie más.

Lista de chats

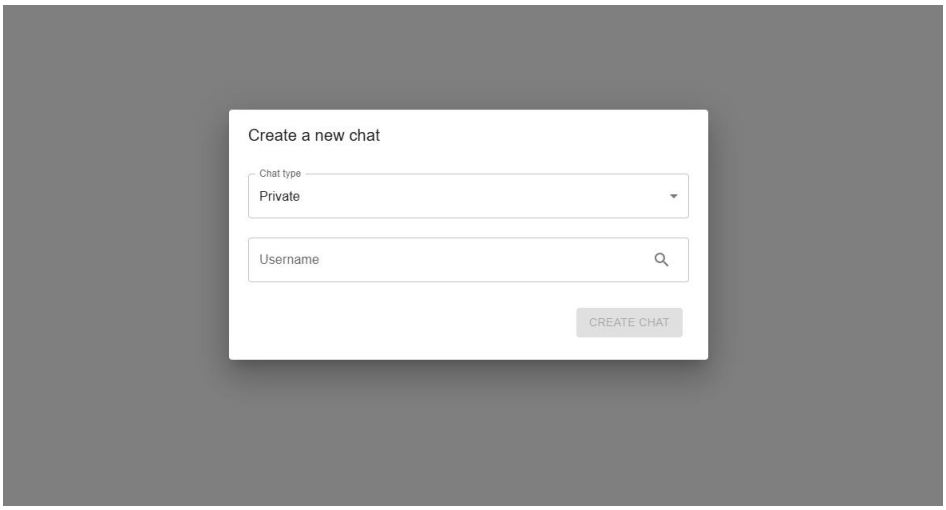
/EasyChat/chats

En esta página podrás visualizar los chats, mostrándose primero los últimos en los que se ha enviado un mensaje. Aquí podrás ver chats tanto grupales como individuales.

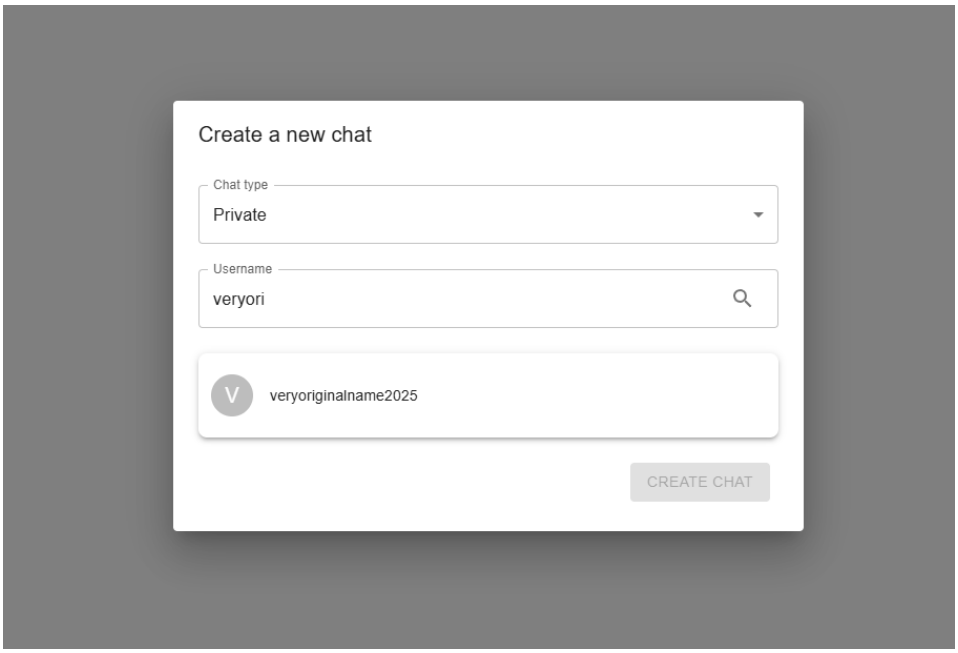
Arriba se muestra un header con el nombre de la aplicación, un selector de idioma y tu avatar, que al pincharlo te saldrá un menú con la opción de cerrar la sesión.

Si quieres crear un chat, tendrás que presionar el botón con el símbolo “+” que se encuentra abajo a la derecha. Al pincharlo, se te abrirá una ventana con las opciones de creación de chat.

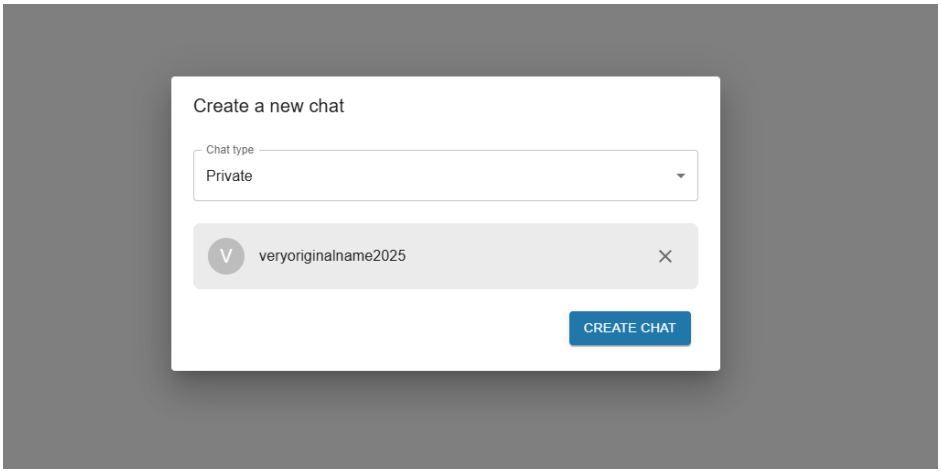
Creación de chats



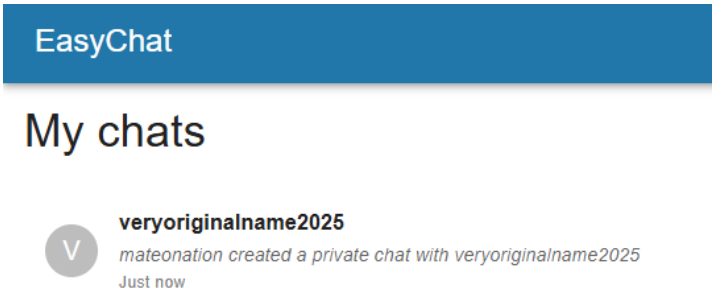
En este modal podrás crear un nuevo chat, ya sea privado (solo tú y otro usuario) o grupal (tú y varios usuarios). Para crear un chat privado solo tienes que elegir ‘Private’ y buscar un usuario por su username.



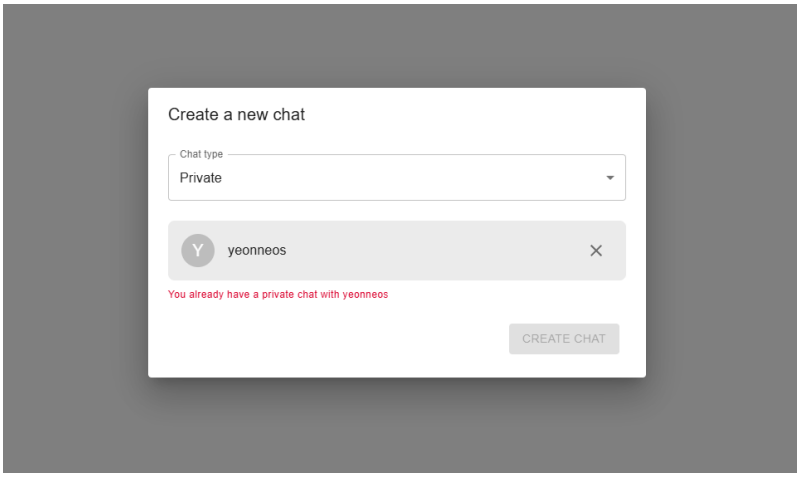
En cuanto veas el usuario con el que quieres crear tu chat privado, selecciónalo haciendo click en él.



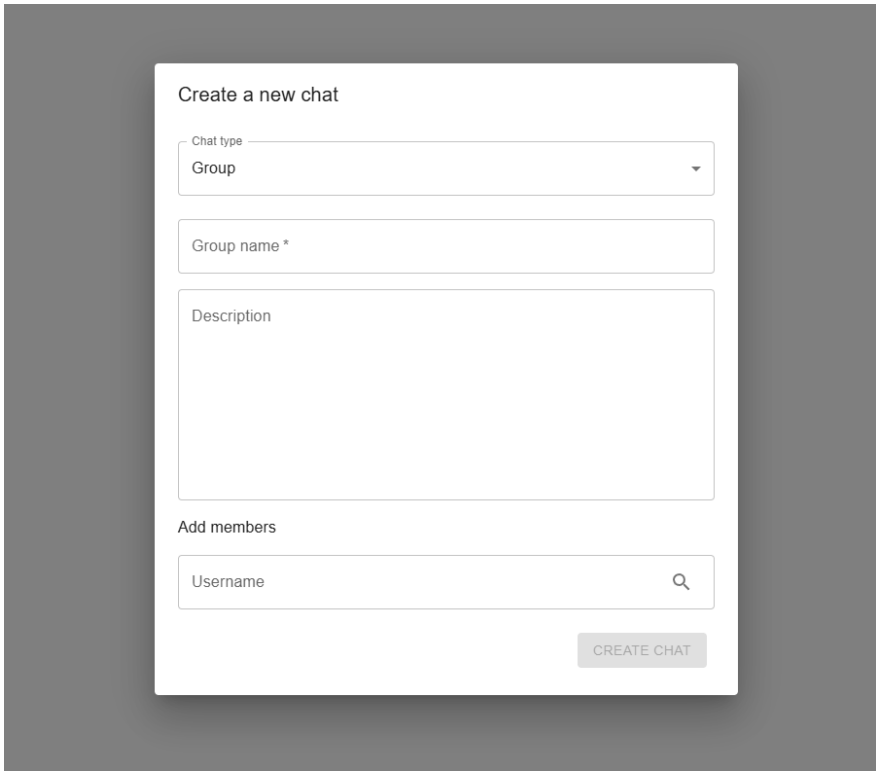
¡Y listo! Acabas de crear tu primer chat privado. Ya puedes enviarle mensajes a ese usuario y él los recibirá.



De todas formas, solo puedes tener un chat privado por usuario, por lo que, si quieres crear un chat privado con un usuario con el que ya tenías uno, te saldrá una advertencia.

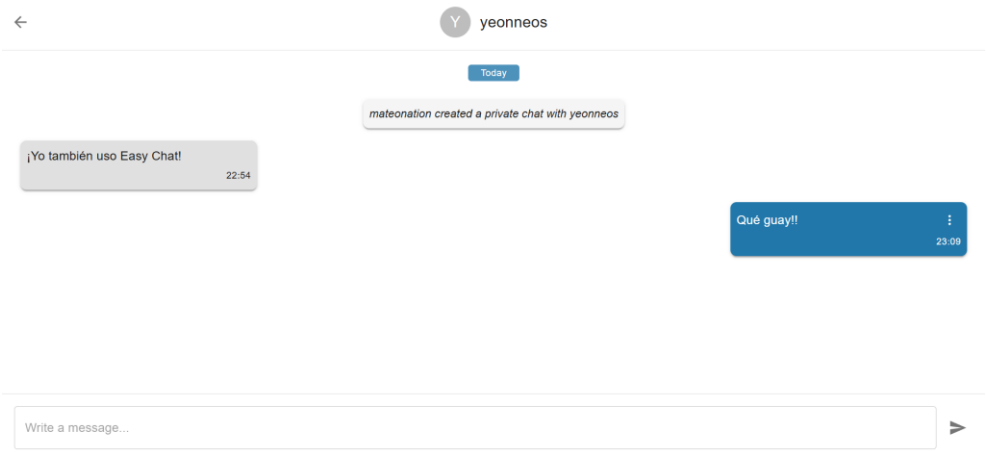


Si eliges crear un chat grupal, puedes elegir un nombre y una descripción para dicho grupo, además de poder añadir más miembros. Las posibilidades son infinitas.



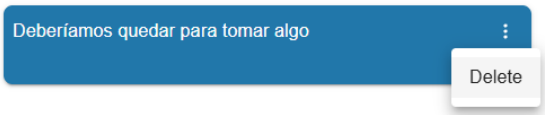
Página de chat

/EasyChat/chats/:id

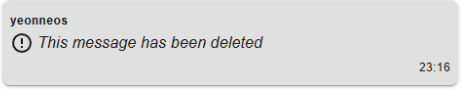


Esta es la página de conversación con una persona o en un grupo, los mensajes se ordenan hacia arriba los más antiguos y hacia abajo los más nuevos.

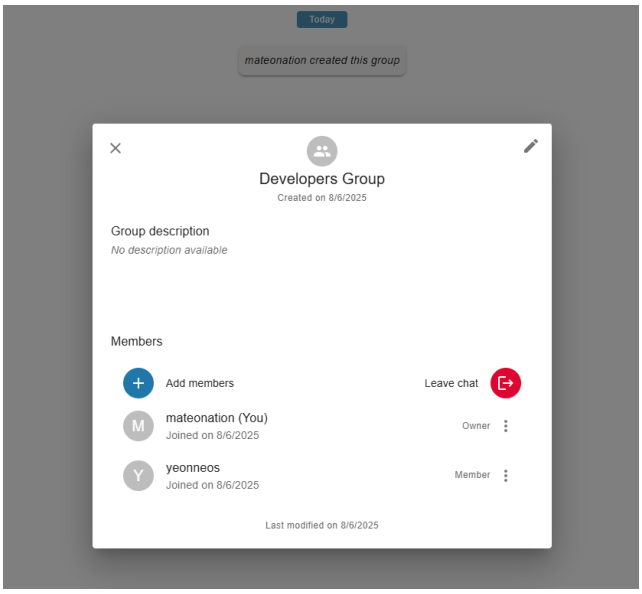
Para que la experiencia del usuario sea lo más fluida posible, los mensajes del chat se cargan de 100 en 100, es decir, al entrar a un chat se cargarán los últimos 100 mensajes enviados a este. Si quieres ver mensajes más antiguos, desliza o haz scroll hacia arriba y la página detectará que querrás cargar mensajes más viejos, así que te cargará los siguientes 100 mensajes.



Al lado de cada mensaje que envíes habrá un botón con el que te saldrá la opción de borrar tu mensaje. Si lo haces, el contenido del mensaje original seguirá guardado en la base de datos, pero se marcará como borrado y ya no estará disponible para los demás usuarios.



Podrás ver la información de los chats pulsando en el nombre del banner arriba, te saldrá un modal con los miembros, nombre y descripción, además de otros datos como fechas de creación y última edición.



Desde aquí puedes modificar los datos del grupo, como añadir usuarios nuevos, administrar los ya existentes o editar el nombre y descripción del grupo.

Desarrollo del proyecto

En esta sección de la memoria se tratarán las decisiones que he tomado a lo largo del proyecto que sean más importantes, por qué las hice y el resultado de estas.

Backend con NestJS y TypeORM

Al principio del proyecto no tenía mucha idea de qué framework usar para el backend. Que tuviera una implementación fácil y cómoda con Websockets para el envío y actualización de los datos a tiempo real era la mayor prioridad.

De entre varias opciones que tenía, la que mejor pinta tenía era NestJS. Este es un framework que me permitía organizar el código en módulos, controladores y servicios, lo que nos permite separar responsabilidades, reutilizar lógica (lo más importante), pero, sobre todo, facilitar la escalabilidad para que la implementación de futuras mejoras no fuera caótica. Además, el soporte 100% en TypeScript ofrece una detección temprana de errores, un tipado totalmente estricto y una mejor mantenibilidad en un proyecto como el mío.

La integración con bases de datos es sencilla. Con TypeOrm puedo crear entidades como clases, manejar relaciones entre ellas y realizar las operaciones CRUD usando repositorios. A parte de ser un framework bastante moderno, tiene un software integrado para Websockets Que permite implementar comunicación en tiempo real, lo cual es una característica indispensable.

También pensé en usar Spring Boot y Spring data JPA, pero el tiempo de desarrollo en este framework, al ser JAVA, requiere de mucho más tiempo y es más complejo. Por lo tanto, NestJS se convirtió en una muy buena opción para mi proyecto, encajaba muy bien con lo que requería.

Aplicación multilenguaje

Uno de mis objetivos para que la aplicación sea lo más accesible posible era que estuviera disponible en distintos idiomas. Para llevar a cabo este objetivo, Comencé a usar la librería de i18n de React que nos permite crear una aplicación con una interfaz que podía desarrollarse en varios idiomas y que estos pudieran ser cambiados y editados con facilidad.

Frontend con Material UI

No es ningún secreto que las aplicaciones de mensajería actualmente existen diseñadas principalmente para dispositivos móviles. Así que, uno de mis requerimientos para este proyecto era la realización de una interfaz responsive que se pudiera adaptar fácilmente a cualquier tipo de pantalla y tamaño de esta.

Material UI ofrece un abanico de ventajas tanto visuales como funcionales que hicieron que mi aplicación fuese fácil de usar. En primer lugar, este framework implementa componentes ya estilizados y listos para usar, están basados en las guías de estilo de Google, lo que garantizaba que mi aplicación tuviera una interfaz limpia consistente y familiar para todos los usuarios, haciendo que mi proyecto se viera profesional sin diseñar nada desde cero.

Dichos componentes están diseñados para adaptarse a cualquier tamaño de pantalla así que la implementación de este framework en mi aplicación tiene esa ventaja de ser responsive desde el minuto 1 sin que esto supusiera un esfuerzo extra.

Concepto de la aplicación

Quería que Easy Chat fuera eso, una aplicación de mensajería fácil de usar cómoda para los usuarios familiar para ellos, además accesible, práctica e intuitiva. El concepto del diseño y de la estructura de la aplicación también es bastante básico y esencial. El elemento esencial de esta aplicación es que es básica, simple, minimalista, sin ningún tipo de complicación.

El objetivo es hacer que la experiencia del usuario fuese lineal y que este no se perdiera entre las distintas funciones que tiene la aplicación. Es algo así como un lienzo en blanco para que los usuarios pudiesen conectar entre ellos y hacer de Easy Chat un lugar con personalidad.

Rendimiento

Una de mis mayores preocupaciones en el desarrollo era rendimiento de la aplicación tanto en frontend como en backend. Una desventaja de NestJS es que el ser un framework tan grande compilar y arrancar la aplicación puede ser bastante lento, cuantas más librerías externas tenga el proyecto peor rendimiento tendrá.

Para evitar que el backend procesara peticiones innecesarias decidí implementar el uso de un middleware. ¿Cómo funciona? Cuando mandas una petición a un endpoint que requiere de una sesión iniciada antes de entrar al controlador entra en una clase que implementa NestMiddleware. Esta clase se encarga de verificar si la petición entrante contiene una sesión y un ID de usuario, si la tiene, deja pasar la petición al controlador. Esto nos puede ahorrar el procesamiento de peticiones innecesarias, dándole un poco más de oxígeno al servicio.

Para evitar saturar el cliente he decidido que los mensajes de cada chat se cargarán de 100 en 100. Esto es, cuando abres un chat se te cargarán los últimos 100 mensajes enviados, ordenándose desde abajo los más recientes hasta arriba los más antiguos. Si el usuario desea cargar mensajes más antiguos solo tiene que hacer scroll hacia arriba y si hay más el cliente mandará la petición de obtener los siguientes 100

mensajes. Esto evita cargar todos los mensajes de un chat cada vez que se abre, ahorrándonos tiempo de procesamiento en el servidor y que el cliente vaya más liviano.

Viabilidad tecno-económica

Para determinar la viabilidad tecno-económica del proyecto se ha realizado un análisis de los recursos necesarios tanto a nivel hardware como software así como una estimación de costes y rentabilidad.

Requisitos a nivel hardware

Este proyecto puede desplegarse en una máquina con requisitos bastante modestos

1. CPU de 2 núcleos
2. Una memoria de 4 GB
3. Un almacenamiento de por lo menos 20 GB
4. Conectividad con una red de soporte HTTPS y WebSocket

Requisitos de software

Como todos los componentes utilizados en la creación y elaboración de Easy Chat son de código abierto, los costos se reducen drásticamente a 0€ en esta parte.

Plataformas de despliegue posibles

Para esta parte, he buscado diferentes servicios de hosting y he hecho una lista con lo que podría costar un despliegue pequeño de Easy Chat.

Railway:

Ideal para prototipos y MVPs pequeños

Ventajas:

- Fácil de usar.
- Despliegue automático desde GitHub
- Tiene soporte no solo para MySQL sino para PostgreSQL y Redis

Coste:

- Desde $5 dólares al mes. Es gratuito por 500 horas al mes y 512mb de RAM.

Render:

Ideal para proyectos con backend y bases de datos.

Ventajas:

- Despliegue simple y escalado automático básico.

Coste:

- Web services: $7/mes por 512mb RAM.
- Sitios estáticos: gratis.
- DB externa (PlanetScale o Neon) desde $29/mes.

Vercel:

Ideal para: arquitectura JAMStack.

Ventajas:

- Frontend súper rápido.
- Funciona perfecto con Vite + React

Coste:

- Vercel: Gratis (hasta 100GB de ancho de banda)
- Railway o PlanetScale para backend: desde $5 a $10 al mes

DigitalOcean

Ideal para: pequeños equipos o despliegue continuo.

- Fácil integración con GitHub.
- Permite apps backend y frontend

Coste:

- Basic: $5/mes por container (512mb RAM)
- DB MySQL gestionada: desde $15 al mes

Heroku

Ideal para estudiantes o apps con poco tráfico.

Ventajas:

- Gran documentación.
- Addons fáciles de usar.

Coste:

- Plan Hobby (durmiente) Gratis e ilimitado.
- Básico (no durmiente) $7 al mes.
- MySQL externo: usar PlanetScale o ClearDB.

Scaleway / Hetzner Cloud

Más control con un VPS barato

Ventajas:

- Precios muy competitivos.
- Se puede instalar Docker.

Coste:

- VPS de 2 vCPU y 4GB de RAM desde 6€ a 10€ al mes

Resumen de costos

Tras analizar varias plataformas de despliegue en la nube, se estima que el coste mensual aproximado para mantener una versión funcional de **Easy Chat** en producción, a pequeña escala, rondaría entre **10 y 20 euros al mes**, utilizando servicios como Railway, Render o una combinación de Vercel para el frontend y PlanetScale para la base de datos. Este coste incluye el servidor backend, el alojamiento del frontend y una base de datos gestionada con capacidad suficiente para los primeros usuarios. A lo largo de un año, esto supondría un gasto total estimado de **120 a 240 euros anuales**.

Teniendo en cuenta las funcionalidades que ofrece Easy Chat y su potencial escalabilidad, este coste resulta **bastante asequible y rentable**, especialmente para proyectos personales, startups en fase inicial o demostraciones académicas. Además, el sistema está preparado para escalar con facilidad si se incrementa el número de usuarios o la carga del servicio.

Trabajo a futuro

Easy Chat es una aplicación con mucho potencial de escalabilidad y a la que se le pueden implantar muchas mejoras y funciones nuevas en el futuro. Aquí hay una pequeña lista de las potenciales actualizaciones que se podrían desarrollar después.

- **Sistema de notificaciones.** Actualmente, Easy Chat carece de un sistema apropiado de notificaciones que avise al usuario del número de mensajes sin leer que hay en un chat. Esta es una característica que debería ser priorizada.
- **Lista de bloqueos.** Los usuarios pueden bloquear usuarios con los que no quieran interactuar ni recibir mensajes.
- **Canales**. Canales de difusión en los que un usuario podría suscribirse y poder recibir noticias y comunicación que haga el administrador de dicho canal.
- **Mensajes favoritos.** Posibilidad de poder marcar mensajes como favoritos y luego tener un registro de éstos a los que el usuario pueda acceder.
- **Sistema de reportes.** Un sistema en el que los usuarios puedan reportar usuarios que rompan las reglas de comportamiento o sean irrespetuosos hacia otros usuarios.
- **Archivo.** Un chat que sirva para que el usuario pueda enviarse mensajes a sí mismo y pueda guardarse cosas para acceder a ellas más tarde.
- **Mensajes multimedia.** Posibilidad de poder enviar distintos elementos multimedia, ya sean imágenes, vídeos, audios, etcétera.
- **Blog o canal de actualizaciones.** Un blog que se actualice con una nueva entrada cada vez que haya un acontecimiento o una actualización en la aplicación.
- **Personalizar interfaz.** Posibilidad de elegir los colores de la interfaz para permitir que los usuarios puedan tener colores con los que se puedan sentir identificados.
- **Landpage.** Una landpage que defina y presente el producto y sus características fundamentales. Es un buen escaparate virtual para dar a conocer Easy Chat.
- **Videollamadas.** Llamadas de vídeo y audio.
- **Avatares.** Posibilitar a los usuarios la función de elegir una foto de perfil para que sus usuarios tengan más personalización.

Conclusiones

Easy Chat es un proyecto que, si bien es cierto que ha cumplido con gran parte de mis expectativas, hay varios puntos que deberían mejorarse.

Creo que es una aplicación en la que no tenía mucho conocimientos sobre creación de web sockets y he perdido mucho tiempo en esa implementación. Tenía pensado al principio la implementación de un landpage que hiciera de presentación a la aplicación con un blog de noticias o actualizaciones. La implementación de dichas noticias me resultaba muy compleja, ya que quería que estás pudiesen tener distintos idiomas disponibles.

Echo en falta el hecho de que la lista de chats no se vaya actualizando a medida que se van enviando mensajes, o sea, los mensajes dentro del chat se envían y reciben a tiempo real, pero no tienes manera de ver eso desde la lista de chats a no ser que reinicies la página o entres a uno de ellos.

A la parte visual habría que darle una vuelta y pulirla más, pero aún así creo que cumple con lo mínimo que requería la aplicación, visualización de mensajes, chats, información, etc. Inicie la parte frontal bastante tarde en el desarrollo y he tenido bastantes problemas a la hora de integrar los Websockets debido a que es una característica bastante nueva la cual no estoy muy familiarizado.

Aún así, me siento satisfecho porque dentro de un chat puedes enviar recibir y eliminar mensajes y que estos los reciban los demás usuarios en tiempo real, creo que ése era el único requisito que necesitaba una aplicación de mensajería para que fuera instantánea.

Para darle más dinamismo a la aplicación estaría bien que en un futuro se pudiesen crear usuarios automatizados o bots que tengan algún tipo de función especial, como por ejemplo administrar grupos, dar información sobre un tema, etc.

Recursos web utilizados

Documentación de Nest JS: https://docs.nestjs.com/

Documentación de React: https://react.dev/

Documentación de TypeScript: https://www.typescriptlang.org/docs/

Documentación de Vite https://vite.dev/guide/

Documentación de Node https://vite.dev/guide/

Documentación de Material UI https://mui.com/

Agradecimientos

A Raquel Rodríguez

Por ayudarme a traducir la aplicación al portugués.

A Sabrina Husanu

Por ayudarme a traducir la aplicación al rumano.

A Alberto Paz

Por probar la aplicación conmigo y darme un feedback honesto.