Web Chattify

Bruno García Trípoli I.E.S HLanz

INTRODUCCIÓN

Web Chattify es el Trabajo de Fin de Grado (TFG) desarrollado para I.E.S. HLanz, este trabajo integra diferentes técnologias y metodologías de desarrollo para desarrollar un chat online web que permite a sus usuarios enviar mensajes en tiempo real.

INDICE

REQUISITOS

REQUISITO DE DATOS

RD1■ Datos de Clientes
- **RD1.1■** Correo electrónico
- **RD1.2■** Password
- **RD1.3■** Display name, Nickname
- **RD1.4■** Roles

RD2■ Datos de la sala chat (Chatroom)
- **RD2.1■** Titulo
- **RD2.2■** Fecha de creación

RD3■ Datos del participante
- **RD3.1■** Fecha de inscripción

RD4■ Datos del mensaje
- **RD4.1■** Fecha de creación
- **RD4.2■** Fecha de edición
- **RD4.3■** Contenido

REQUISITOS FUNCIONALES

Los requisitos funcionales se dividen en Datos de Entrada (E), Datos de Almacenamiento o Modificación (A/M), Datos de Salida (S) y Requisitos No Funcionales (RNF).

RF1■ Añadir nuevos usuarios (Sign Up)

E	**A/M**	**S**
RD1.1, RD1.2, RD1.3	RD1	RD1
RNF	RF1, RNF6	

RF2■ Autenticarse como usuario

E	**A/M**	**S**
RD1.1, RD1.2	RD1	RD1
RNF	RF2, RNF6	

RF3■ Crear una sala chat

E	**A/M**	**S**
RD2.1	RD2	RD2
RNF	RF3, RNF6	

RF4■ Consultar salas chats

E	A/M	S
RD1	RD1, RD2	RD2
RNF	RNF6	

RF5■ Invitar usarios a salas chats

E	A/M	S
RD1, RD2	RD1, RD2	RD3
RNF	RNF4, RNF6	

RF6■ Crear un mensaje

E	A/M	S
RD1, RD4.3	RD1, RD4	RD4
RNF	RNF5.1, RNF5.2, RNF6	

RF7■ Editar mensajes

E	A/M	S
RD1, RD4	RD1, RD4	RD4
RNF	RNF5.1, RNF5.2, RNF5.3, RNF6	

RF8■ Consultar mensajes de una sala chat

E	A/M	S
RD1, RD2	RD1, RD2	RD4
RNF	RNF6	

RF9■ Consultar registros de edicion de un mensaje

E	A/M	S
RD1, RD4	RD1, RD4	RD4
RNF	RNF5.4, RNF6	

RF10■ Desarrollar una API

RNF	RNF6, RNF7, RNF8

RF11■ Desarrollar interfaz gráfica

RNF	RNF9, RNF10

REQUISITOS NO FUNCIONALES

RNF1■ Restricciones de creación de un usuario
- **RNF1.1■** No puede existir mas de un usuario con el mismo correo electrónico
- **RNF1.2■** Puede existir multiples usuarios con el mismo nickname
- **RNF1.3■** La contraseña tiene que estar cifrada
- **RNF1.4■** El correo electrónico tiene que ser valido
- **RNF1.5■** El nickname tiene que ser valido
- **RNF1.6■** Un usuario puede ser identificado por un correo electrónico o un identificador único

RNF2■ Restricciones de login
- RNF2.1■ Los campos tienes que ser verificados antes de autenticarse

RNF3■ Restricciones creacion de las salas chat
- **RNF3.1■** Se pueden crear varias salas con el mismo nombre
- **RNF3.2■** Se puede crear una sala vacia

RNF4■ Restricciones para invitar usuarios a salas chat
- **RNF4.1■** No se puede invitar dos veces al mismo usuario
- **RNF4.2■** El propietario no puede invitarse a si mismo en su propio chat

RNF5■ Restricciones para un mensaje
- **RNF5.1■** Los mensajes no pueden estar compartido entre varias salas chat
- **RNF5.2■** No se puede enviar un mensaje vacio
- **RNF5.3■** Un usuario solo puede editar sus propios mensajes
- **RNF5.4■** Solo los Admin pueden consular los registros de edicion

RNF6■ Solo es accesible mediante previa autentificación

RNF7■ La API solo puede enviar mensajes en formato JSON, incluyendo errores

RNF8■ Las llamas a la API sin ninguna ruta registrada tienen que ser controladas

RNF9■ La interfaz gráfica tiene que ser altamente versatil ante cambios

RNF10■ La interfaz gráfica tiene que adaptarse a todo tipo de dispositivos

PLANTEAMIENTO DEL DESARROLLO

PLANTEAMIENTO BASE

Como primer paso para el desarrollo de la aplicación, se implementará el proyecto usando como base Docker, para el desarrollo de back-end se utilizara Symfony, y como apoyo al desarrollo se integrara:

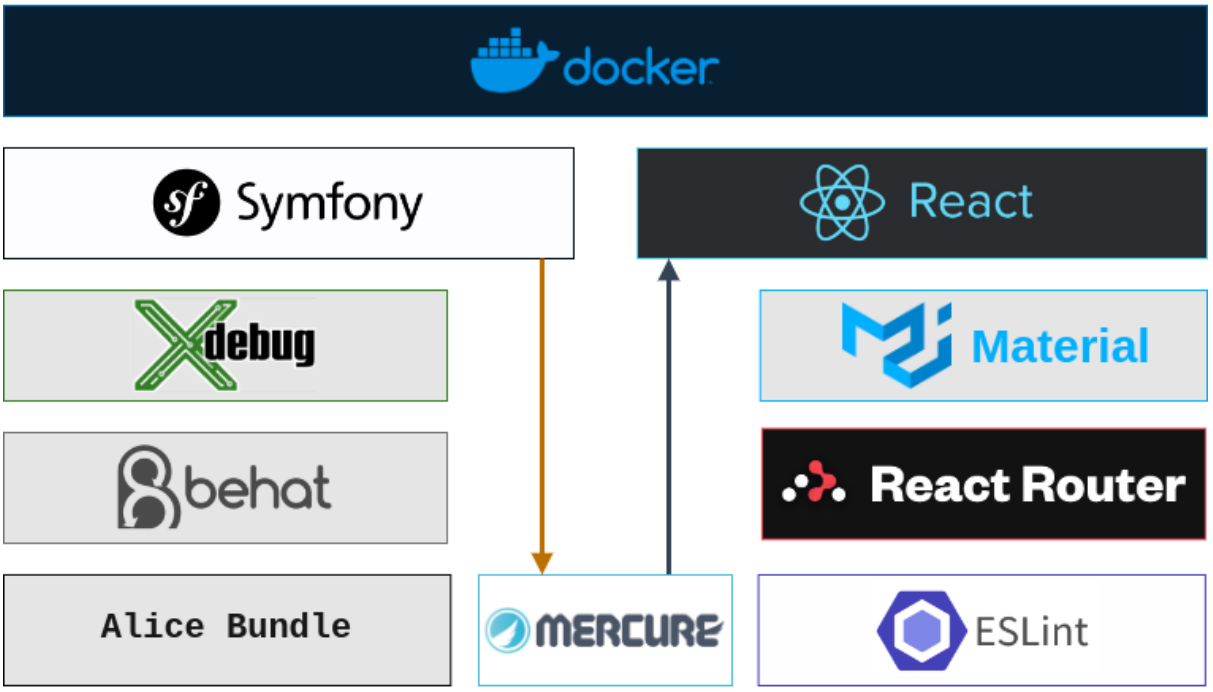
- XDebug
  Apoyo esencial para debuguear durante el desarrollo
- Alice Bundle
  Soporte para cargar datos en la base datos, util para crear una base de datos stable para ejecutar la bateria de tests.
- Behat
  Soporte de una bateria de tests, tambien sirve como documentación

El desarrollo del back-end se centrara en el desarrollo de una API que formatea los resultados en JSON, solucion versatil para implementar el front-end en diversas aplicaciones.

Por otro lado el front-end tiene que estar preparado para ser dinamica y soportar un entorno altamente interactivo, para ello considero que la tecnología mas adecuada es React y como apoyo al desarrollo se integrara:

- Material (MUI)
  Herramienta similar a Boostrap, integrable con React
- React Router
  Permite el uso de enrutamiento dinamico
- ESLint
  Apoyo para encontrar y arreglar errores en el desarrollo de código en Javascript y React

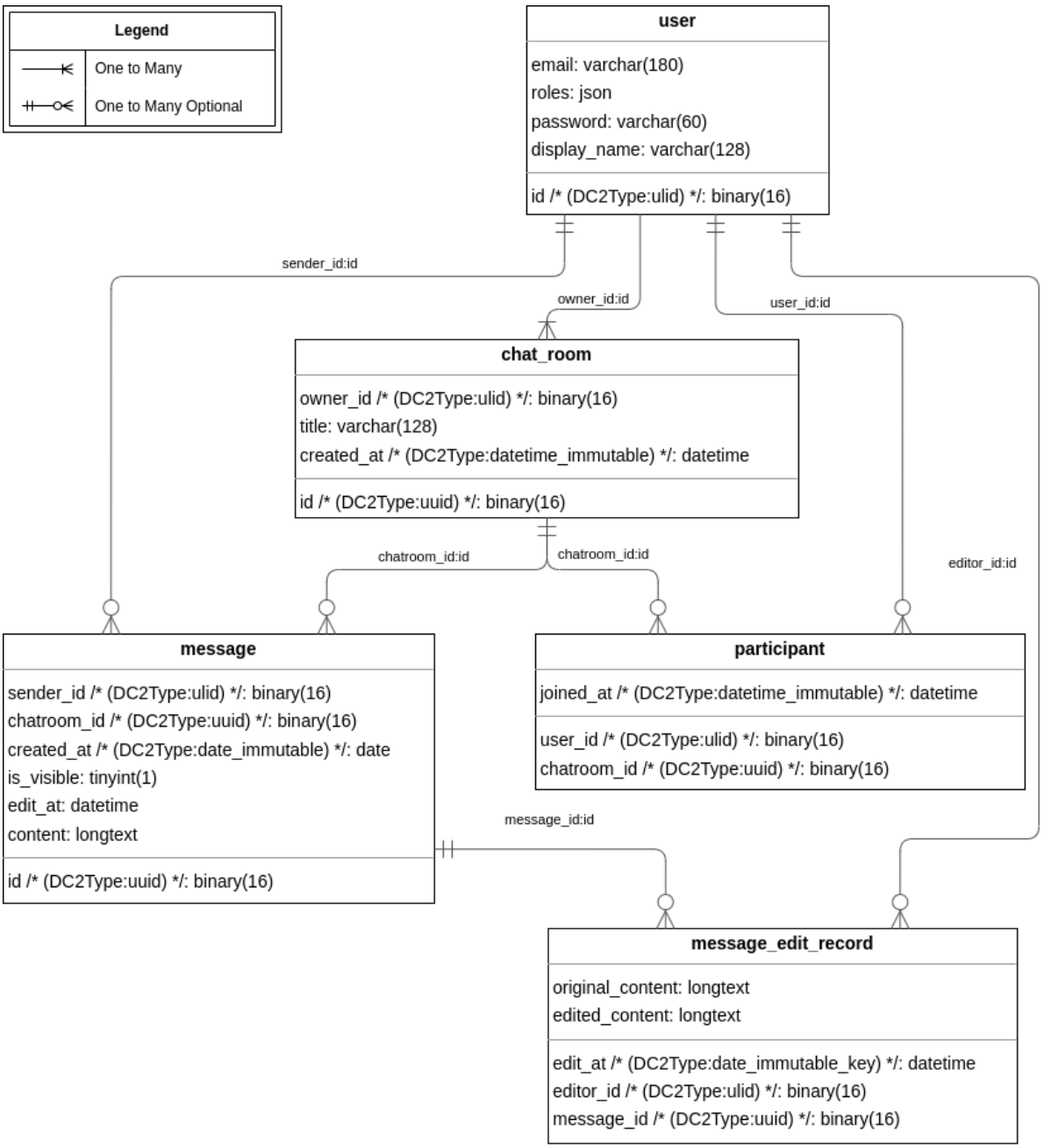
Por último para implementar comunicación en tiempo real he decidido usar SSE (Server Sent Events) con Mercure.



DISEÑO DE LA BASE DE DATOS

La base de datos esta diseñada en MySQL, con apoyo de las migraciones generadas por symfony, se integran dos tipos de identificadores diferentes ULID y UUID, en el caso particular de la entidad User, existe con el objetivo de ocultar y evitar que se disperse información sensible como es el correo electrónico.

A continuacion se muestra un diagrama Entidad Relación de la base de datos:



CARACTERISTICAS DE LA APLICACION

A continuación se van a mostrar las caracteristicas finales del proyecto soportadas por bateria de tests behat que se pueden consultar en el repositorio de GitHub en la carpeta features.

Las siguientes características usan como base ejemplos que se almacenan en la base de datos gracias a Alice Bundle.

Caracteristica: API relacionada con el usuario

Escenario: consultar todos los usuarios del sistema

Dado que un usuario está autentificado como un usuario que posee el rol de ADMIN,
Cuando se envia una petición *'GET'* a la ruta *'/api/users'*
Se recibe una respuesta con estado 200 y un cuerpo JSON
La respuesta contiene un array donde cada elemento contiene:

- Identificador del usuario: id
- Correo electrionico: email
- Nombre visible del usuario: displayName
- Roles de los usuarios: roles

Escenario: un usuario se autentifica correctamente

Cuando se envia una peticion 'POST ' a la ruta '/api/login' con un cuerpo JSON:

- Correo electrónico: email
- Contraseña: password

Se recibe una respuesta con estado 200:

- Identificador del usuario: id
- Correo electrionico: email
- Nombre visible del usuario: displayName
- Roles de los usuarios: roles
- Token: token

Escenario: registrar un nuevo usuario correctamente

Cuando se envia una peticion 'POST ' a la ruta '/api/signup' con un cuerpo JSON:

- Correo electrionico: email
- Nombre visible del usuario: displayName
- Contraseña: password

Se recibe una respuesta con estado 200:

- Identificador del usuario: id
- Correo electrionico: email
- Nombre visible del usuario: displayName
- Roles de los usuarios: roles

Caracteristica: API relacionada con las salas chat

Escenario: consultar las salas chat del usuario

Cuando se envia una petición *'GET'* a la ruta *'/api/chat'*
Se recibe una respuesta con estado 200 y un cuerpo JSON
La respuesta contiene un array donde cada elemento contiene:

- Identificador de la sala chat: id
- Propietario de la sala chat: owner
    - Identificador del usuario: id
    - Nombre visible del usuario: displayName
- Titulo de la sala chat: title
- Fecha de creación: createdAt
- Participantes de la sala chat: participants

Escenario: invitar a un usuario a una sala chat

Cuando se envia una petición *'GET'* a la ruta *'/api/join/chat'*
Se recibe una respuesta con estado 200:

- Identificador de la sala chat: id
- Titulo de la sala chat: title
- Fecha de creación: createdAt
- Propietario de la sala chat: owner
    - Identificador del usuario: id
    - Nombre visible del usuario: displayName
- Participantes de la sala chat: participants

Escenario: consultar los mensajes de una sala chat

Cuando se envia una petición *'GET'* a la ruta *'/api/messages/chat/<chat-id>'*
Se recibe una respuesta con estado 200 y un cuerpo JSON
La respuesta contiene un array donde cada elemento contiene:

- Identificador del mensaje: id
- Fecha de creación: createdAt
- Fecha de edicion (opcional): editAt
- Propietario del mensaje: sender
    - Identificador del usuario: id
    - Nombre visible del usuario: displayName
- Contenido del mensaje: edit

Escenario: consultar los mensajes de una sala chat

Cuando se envia una petición *'POST'* a la ruta *'/api/message'* con un cuerpo JSON:

- Identificador de la sala chat: chatroom
- Contenido del mensaje: mensaje

Se recibe una respuesta con estado 200 y un cuerpo JSON
La respuesta contiene un array donde cada elemento contiene:

- Propietario del mensaje: sender
  - Identificador del usuario: id
  - Nombre visible del usuario: displayName
- Identificador de la sala chat: chatroom
- Propietario del mensaje: sender
  - Identificador del usuario: id
  - Nombre visible del usuario: displayName
- Fecha de creación: createdAt
- Contenido del mensaje: edit

Escenario: editar un mensaje

Cuando se envia una petición *'PATCH'* a la ruta *'/api/message/<chat-id>'*:

- Contenido del mensaje: mensaje

Se recibe una respuesta con estado 200 y un cuerpo JSON
La respuesta contiene un array donde cada elemento contiene:

- Identificador del mensaje: id
- Fecha de creación: createdAt
- Fecha de edición: ediatAt
- Contenido del mensaje: content
- Propietario del mensaje: sender
  - Identificador del usuario: id
  - Nombre visible del usuario: displayName

Caracteristica: API

La aplicación debe tener una API consistente por ello cualquier peticion con prefijo '/api/' debe devolver como respuesta un JSON en caso de error o una ruta invalida se devolvera una respuesta JSON con formato:

- Código respuesta: code
- Descripción: message
- (Exclusivo para entornos de desarrollo: 'test', 'dev')
  - Entorno de desarrollo: environment
  - Detalles de la excepción: detail
    - Trace: trace
    - Tipo de la excepción: type
    - Mensaje original: message
    - Código original de la excepción: code

DESPLIEGE DEL PROYECTO EN LOCAL

Para desplegar el proyecto en local, es necesario tener instalado en la maquina local Docker y ejecutar los siguientes comandos del Makefile definido en el proyecto:

- Run: *make up*
- Run: *make init*
- Visit http://localhost:8000

Para más información visita el repositorio GitHub.