María Elena Jiménez Muñoz.

WHATSIM

Raúl Jiménez Galán

WHATSIM 23/05/2021

1. Índice

1.1 Lista de imágenes

2. Análisis previo

2.1 Introducción

Elegí hacer este proyecto usando angular porque me parecía muy interesante y quería ahondar más en él, y qué mejor que hacer un proyecto entero con frontend en angular y back en nodejs para conocer sus virtudes, defectos y curiosidades y conocer más.

Está pensado para emular chats de la conocida aplicación Whatsapp. Se imita el diseño de interfaz de la misma para dar el mayor nivel de realismo posible

2.2 Motivación

La mayor motivación que tuve para decantarme para realizar este proyecto, fue que tenía la idea de simular un chat de Whatsapp para un regalo de cumpleaños. Busqué herramientas para ello pero no encontré prácticamente nada como lo que yo tenía en mente y necesitaba, así que me plantee la idea de realizarlo como proyecto final y así fue.

2.3 Objetivos

Lamentablemente quizás la idea inicial era demasiado ambiciosa para una sola persona, con muy poca experiencia en angular y con un plazo de tiempo tan limitado. Muchos de los objetivos iniciales de la idea, no se han podido llevar a cabo, ya sea por cuestión de tiempo, limitación de la propia tecnología utilizada o por simple incapacidad para realizarlos.

Inicialmente la idea de la aplicación concebía la capacidad de estos objetivos:

- Registro con email, contraseña y nombre de usuario: este objetivo si ha sido cumplido sin mayor complicación.
- La página de inicio debería de estar elaborada, con datos como usuarios registrados, chats totales creados, mensajes totales: Este objetivo no ha podido ser cumplido debido al tiempo.
- Un usuario podría crear, editar y eliminar un número ilimitado de chats: Si se ha podido cumplir.
- Un usuario podría crear, editar y eliminar un número ilimitado de mensajes dentro de un chat: Este objetivo no se ha podido

cumplir al cien por cien, ya que los mensajes solo pueden ser creados pero no eliminados ni editados, aunque esto este tanto contemplado e integrado en el backend así como perfectamente contemplado en el frontend, incluso a tal punto de existir botones para lo mismo, no esta implementada la funcionalidad por falta de tiempo.

- Los chats contendrían un nombre de perfil, fecha de creación, título y foto de perfil: Aunque tanto en la parte backend (base de datos y servidor rest) y en los modelos del frontend este perfectamente integrado y planteado, la foto de perfil no ha podido ser llevada a cabo por falta tanto de tiempo para realizarlo, como para investigación.
- El chat debe de tener una apariencia muy parecida a la aplicación whatsapp para dar sensación de emulación completa: Este objetivo si se ha podido completar, aunque con mucho tiempo de esfuerzo y dificultades.
- El chat se podría ampliar en pantalla completa: Este objetivo fue contemplado, pero al final ni siquiera se investigó.
- El chat tendría un botón de despliegue para emojis: Se contemplo esta posibilidad en el último momento y se investigó,

pero no se pudo a llevar a cabo por falta de tiempo. Incluso hay un enlace de referencia en la bibliografía a una utilidad para uso de emojis.

- Los mensajes podrían mostrarse en una animación: Esto fue solo una idea y no se llegó a contemplar por la aparente complejidad de esta.
- Los mensajes contendrían información de la hora exacta del envió o recibimiento del mismo, tipo de mensaje (recibido o enviado), estado del mensaje y por supuesto el contenido: Si se ha podido cumplir sin mucha dificultad.
- Los chats pueden ser públicos, es decir ser visibles para cualquiera que tenga acceso al enlace, pero solo editables por el propietario o privados, que solo el propietario del chat pudiera acceder a él.
- La aplicación debería de contar con el protocolo seguro SSL: Se estudió la posibilidad para realizarla una vez estuviese todo finalizado y si había tiempo. Lamentablemente, al contar con dos servidores (backend y frontend) en la misma máquina, no es tan fácil. Se tendría que instalar apache (otro servidor web más) para

crear los llamados servidores virtuales, para poder redirigir subdominios a ciertos puertos. Hay un enlace en la bibliografía

3. Recursos necesarios

3.1 Recursos humanos

Solo yo he contribuido a la creación del proyecto, descontando las personas que hayan escrito o hecho las páginas de documentación y referencia y todas las personas que hayan contribuido a la creación de los recursos utilizados.

3.2 Recursos hardware

Para la creación de este proyecto he necesitado de varios elementos hardware:

- Mi ordenador personal: Para las tareas de investigación, documentación, diseño y elaboración del proyecto. Mi ordenador cuenta con 16GB de memoria ram DDR4 y un procesador de

penúltima generación de AMD con 8 núcleos. Mucho más que suficiente para todas las tareas realizadas

- Servidor vps: Alquiler de un servidor VPS con Linux, debian más concretamente, para la instalación y despliegue de el servidor backend, base de datos y servidor frontend. Cuenta con un procesador de 2 núcleos, 4GB de memoria ram y un disco duro ssd lo suficientemente rápido y amplio para su objetivo.

3.3 Recursos software

Al ser un proyecto de programación, son muchos los recursos software necesarios y no creo que sea posible enumerarlos a todos y cada uno de ellos, ya que son muchas las librerías y recursos utilizados y que están integrados en las tecnologías como angular. Pero si es posible hacer una enumeración a grandes rasgos:

- Programas utilizados:
    - Bitvise: Programa para la conexión SSH y FTTP al servidor

- Visual Estudio Code: Entorno de desarrollo tanto para la parte frontend en angular como backend.
- Plugins Visual Estudio Code: Al ser este un programa muy genérico, para ciertas especializaciones como angular, es necesario instalar ciertos plugins para su correcta interpretación y autocompletación.
- Photoshop: Necesario para la creación de los logotipos del proyecto
- Microsoft Word: Programa utilizado para la redacción de esta memoría
- DataGrip: Es un programa para la administración y visualización de bases de datos de la conocida empresa desarrolladora de entornos de desarrollo JetBrains, conocida por otros programas como Intellij Idea que es un IDE para Java entre otros.
- Google Chrome: Navegador utilizado tanto para las tareas de investigación y documentación, como para las de testeo y experimentación.

- Advanced Rest Client: Es una aplicación para probar servidores REST. La utilicé para comprobar el backend REST antes de tener listo nada del frontend.

- Tecnologías utilizadas:
  - Nodejs: Es una tecnología bastante conocida y usada alrededor de todo el mundo tanto particulares, pequeñas empresas o hasta multinaciones. Nodejs no es solo un simple gestor de paquetes, utilizado por ejemplo por angular. Es también un muy potente servidor web, escrito en javascript. Es utilizado para la parte backend del proyecto.
  - Angular: Es una tecnología web muy moderna y creciente en estos últimos años. Esta creada y mantenida por Google y esta pensada para crear aplicaciones web single page, es decir de una sola página. Es una tecnología muy potente pero muy compleja y extensa a la vez. Es la tecnología utilizada en la parte frontend.

  - MariaDB: Es una de las muchas tecnologías de bases de datos. Es una base de datos relacional y muy parecida (de hecho, heredada) a mysql. He decidido utilizar esta por su simplicidad frente a otras más modernas y más utilizadas junto a las tecnologías anteriores como mongodb.

- Librerías principales:
  - Expressjs: Es una librería para nodejs que facilita aún más la creación de servidores web, sobre todo de tipo REST como es este caso concreto. Es utilizado para la creación de la parte backend del proyecto
  - Sequelize: Se trata de una librería ORM para nodejs. Sirve para la gestión de bases de datos relacionales. Viene a ser, salvando las distancias, el equivalente en nodejs a hibernate para java. Se encarga, a través de modelos, de todas las operaciones de creación de tablas, y filas (datos) dentro de las mismas de una manera mucho más intuitiva y sin necesidad de utilizar lenguaje SQL. Es utilizado en la parte backend

- Angular material: Como su nombre indica, es una librería para angular, la cual añade gran cantidad de componentes con un diseño ya establecido. La guía de diseño seguida es Material Design de Google, como su nombre incida. Siendo Angular de Google es bastante comprensible que sea asi. Personalmente este diseño me parece muy bueno y agradable tanto para aplicaciones empresariales como informales. Esta librería sería equivalente al conocido Bootstrap.

3.4 Recursos gráficos

Las distintas variantes de logotipos creadas, están basadas en el logotipo original de Whatsapp.

ILUSTRACIÓN 1: LOGOTIPO OSCURO CON FONDO TRANSPARENTE

ILUSTRACIÓN 2: LOGOTIPO VERDE CON FONDO TRANSPARENTE

ILUSTRACIÓN 3: LOGOTIPO BLANCO CON FONDO VERDE

4. Planificación

Personalmente me cuesta mucho organizarme con mucha anterioridad y precisión ante algo. Me suelo organizar muy poco previamente y más bien sobre la marcha.

4.2 Secuencia de desarrollo

1. Una vez tenía más o menos claro que quería, cómo lo quería y que tecnologías iba a utilizar para ello, lo primero fue investigar ciertas dudas a grandes rasgos que me surgieron, como si se podía “conectar” angular con un servidor REST. Una vez que supe a grandes rasgos, era posible hacerlo, continúe adelante.
2. Después miré donde alquilar el vps, y decidí que en ovh. Una vez tenia el acceso a él, con debían instalado y actualizado, instale nodejs y npm, para así poder instalar angular e iniciar el servidor backend REST. Tambíen instale un servidor mariadb para la base de datos. No me surgió mucho problema con esto, ya que más o menos lo tengo controlado. De todas formas, siempre estaba con documentación delante.
3. Lo siguiente fue buscar ejemplos de aplicaciones angular conectadas a un backend REST. A esto se les llama aplicaciones CRUD. Una vez poco a poco lo iba entendiendo muy a grandes rasgos, fui más específico y busqué que el backend REST fuera

en node. Ahí es cuando descubrí la librería sequelize, la cual hasta entonces para mí era desconocida y que me ha servido de mucho en este proyecto.

4. Al descubrir la librería de sequilize, decidí que la base de datos iba a ser mariadb, ya que quería probar esta librería y no tengo tanta experiencia con otras bases de datos como mongodb.

5. Una vez ya tenía la idea de como iba a ser y como realizar el backend REST, me puse a idear la base de datos. Este fue el resultado

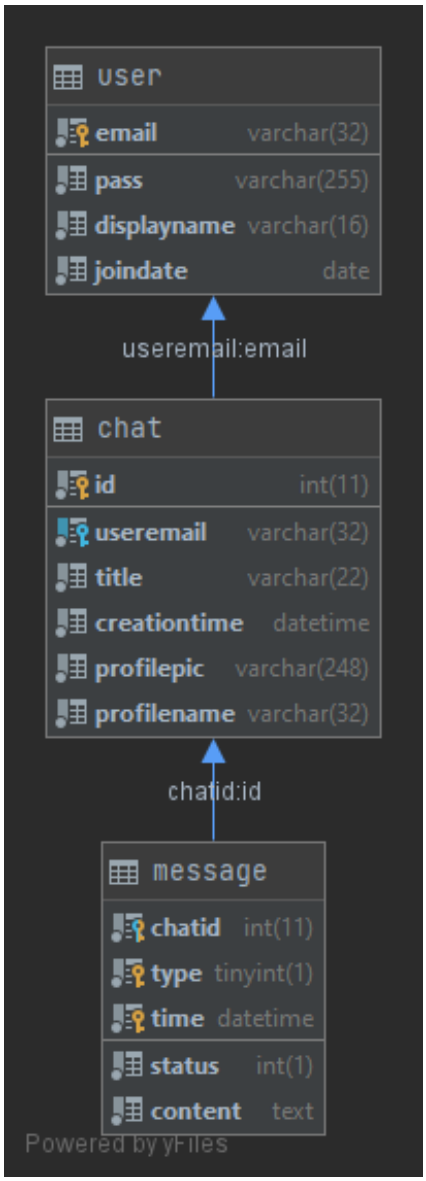


ILUSTRACIÓN 4: DIAGRAMA DE BASE DE DATOS

6. Tenía la base de datos creada y la idea del backend, así que con lo que ya sabía de nodejs y expressjs, me puse a hacer el backend, también con la vista puesta en los ejemplos de backend con sequelize y con su documentación. Lo primero hice pruebas usando sequilize, conectando, creando modelos...

Este es el resultado final hoy en día de la estructura del backend

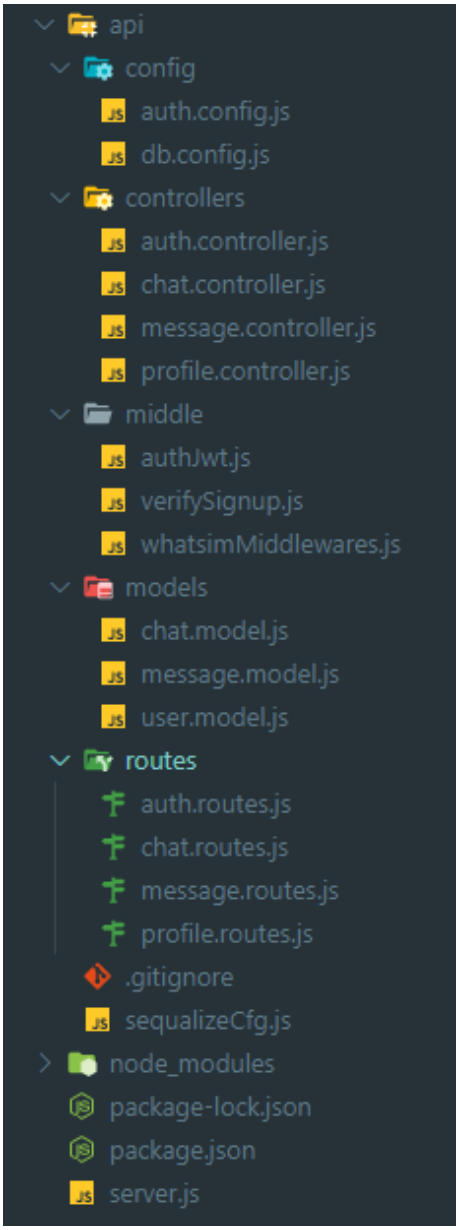


ILUSTRACIÓN 5: ESTRUCTURA DEL BACKEND

7. A continuación, dejé el backend listo para registro e inicio de sesión. Una vez aquí, era el turno de investigación para la parte frontend.
8. Tras mucho tiempo de investigación, pruebas y demás, cree el proyecto en angular e instalé angular material.
9. Investigando y probando otro mucho tiempo más, deje la barra de navegación, routing, registro e inició de sesión hechos. A partir de aquí fue todo mucho más fluido ya que ya iba comprendiendo mucho más angular y su metodología de trabajo.
10. Una vez ya tenía mas conocimiento y experiencia con angular, el resto del trabajo se facilitó mucho. Realizaba la parte backend para, por ejemplo, los chats y a continuación la parte frontend. Una vez acabado esa parte, hacía lo mismo con las siguientes. Así hasta terminarlo.

4.3 Actividades y administración del tiempo

Principalmente, el tiempo invertido se puede dividir en dos grandes grupos: Tiempo de investigación y tiempo de desarrollo. Se podría decir

que, aproximadamente más del 65% del tiempo, ha sido empleado en investigación.

Al principio invertía aproximadamente dos horas y media al día, durante tres días a la semana. Aunque desde finales de abril viendo como iba de tiempo, me puse más en serio y empecé a invertir muchas más horas y días. Empecé a invertir más de tres horas cada noche durante cinco días a la semana.

5. Desarrollo del proyecto

5.1 Secuencia real de desarrollo

Anteriormente ya he citado la secuencia de desarrollo a grandes rasgos. A continuación, explicaré una a una las tareas:

- Lo primero que hice, como ya mencioné anteriormente, fue la parte del inicio de sesión y registro de nuevas cuentas de usuario en la parte backend. Aproximadamente invertí unas 20 horas en esto.

- Lo siguiente fue realizar la barra de navegación en angular, la parte front. Aún no tenía mucha experiencia con angular y me costó más de lo debido, unas 15 horas aproximadamente.

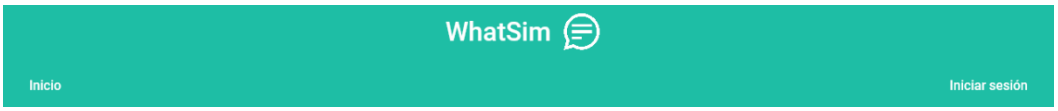


ILUSTRACIÓN 6: BARRA DE NAVEGACIÓN

- A continuación, realice el registro de nuevas cuentas y el inicio de sesión en la parte front. Me llevó más de lo debido por la

curva de aprendizaje de angular, unas 20 horas.

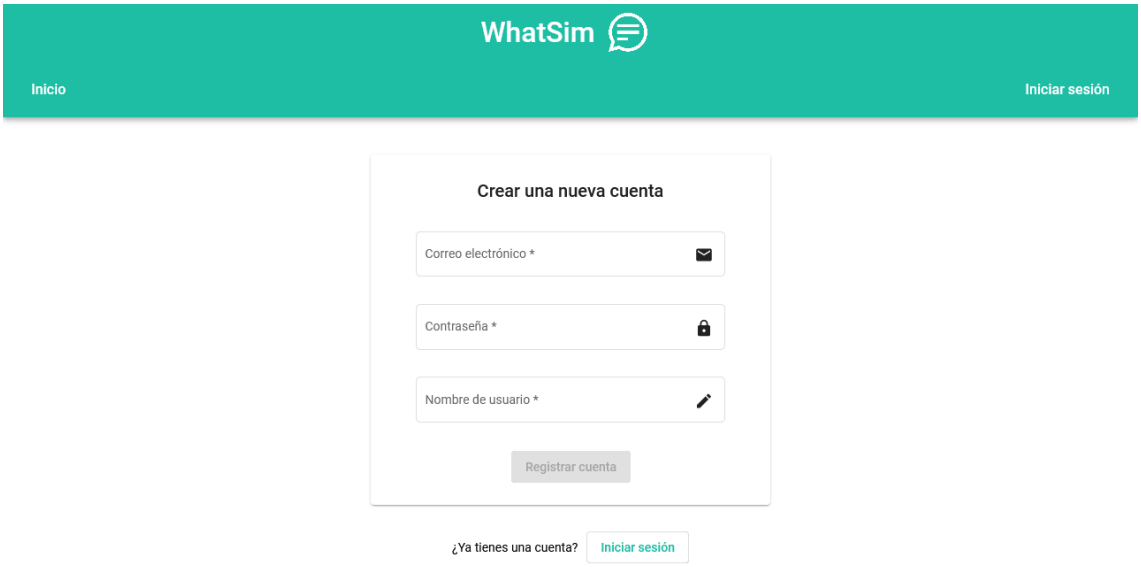


ILUSTRACIÓN 7: REGISTRO DE CUENTA

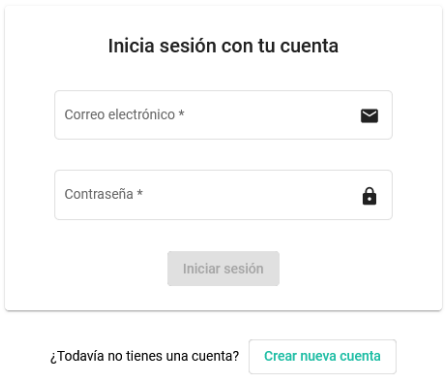


ILUSTRACIÓN 8: INICIO DE SESIÓN

- Lo siguiente que hice fue realizar mi propio componente de errores en angular. Este componente esta implementado en

toda la aplicación excepto en el inicio de sesión y registro. Pueden ser mostrados como modal o como un componente insertado en la página. También realice una página para mostrar cuando la ruta introducida en el navegador no sea válidas. Aproximadamente me llevó unas 10 horas realizarlo.

ILUSTRACIÓN 9: ERROR PÁGINA NO ENCONTRADA

- Lo siguiente fue realizar toda la programación en el backend referente a los chats: creación, edición y eliminación. No me

llevó mucho más de 2 horas pudiendo haber sido menos de no ser por su magnitud y tiempo de pruebas.

- Para la parte de chats en frontend, al tener que realizar un servicio y varios componentes, con su lógica y diseño, tardé mas que en la parte backend, unas 16 horas.

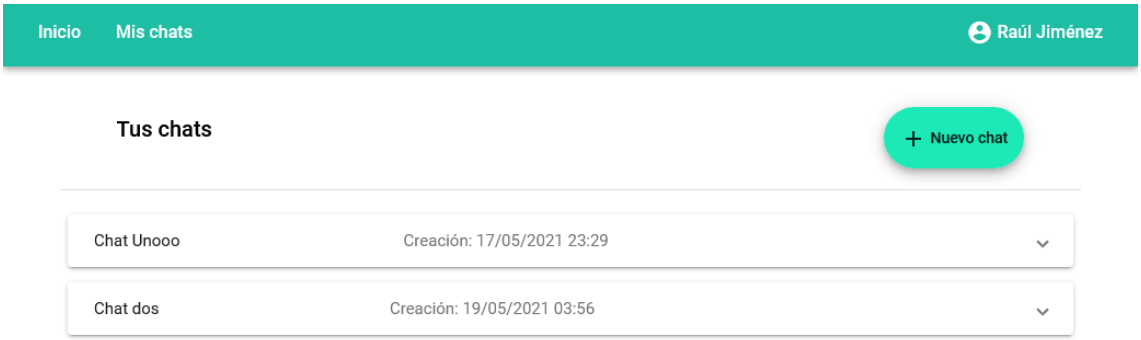
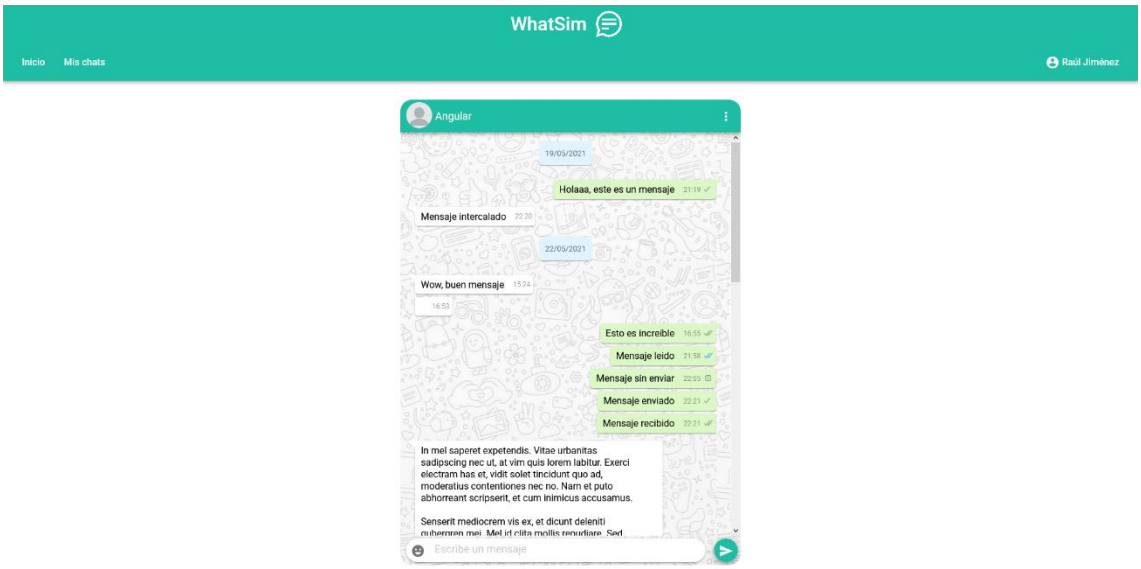


ILUSTRACIÓN 10: MIS CHATS

- A continuación realice toda la programación en backend referente a los mensajes: visualización, creación, edición y eliminación, aunque como ya he mencionado anteriormente solo visualización y creación son usadas en el frontend. Esto me llevó aproximadamente una hora

- Lo siguiente fue la parte final, la visualización y gestión de los chats con los mensajes. Esta parte debía tener la interfaz muy similar a la aplicación Whatsapp, así que esta tarea me llevo muchísimo tiempo. Solo el desarrollo calculo que me costó unas 40 horas con las pruebas incluidas.

ILUSTRACIÓN 11: VISUALIZACIÓN DE CHATS



Así fue el desarrollo a grandes rasgos, posteriormente procederé a incluir imágenes de partes importantes de código y mejor explicamiento de ciertas partes.

Todas las cantidades de hroas citadas en las diferentes tareas expuestas anteriormente, son orientativas y solo referentes al tiempo de desarrollo y pruebas, el tiempo de investigación es contado por separado.

Si sumamos todas las horas, nos da 114, ciento catorce horas de desarrollo y testeo de la aplicación. Como ya dije anteriormente, esta parte era solo el 35% ya que el 65% restante era tiempo de investigación.

Con esta información, podriamos calcular aproximadamente que el tiempo de investigación fue de 211 horas y el tiempo total del proyecto de 325, aunque como ya dije, esto es muy poco exacto.

5.2 Diagramas y lógica

En esta sección se mostrarán los diagramas que yo creo importantes mostrar y se explicará la lógica de la aplicación. Un breve resumen de la lógica de aplicación sería decir que es una aplicación cliente-servidor.

5.2.1 Diagramas

- Base de datos: Lo más esencial

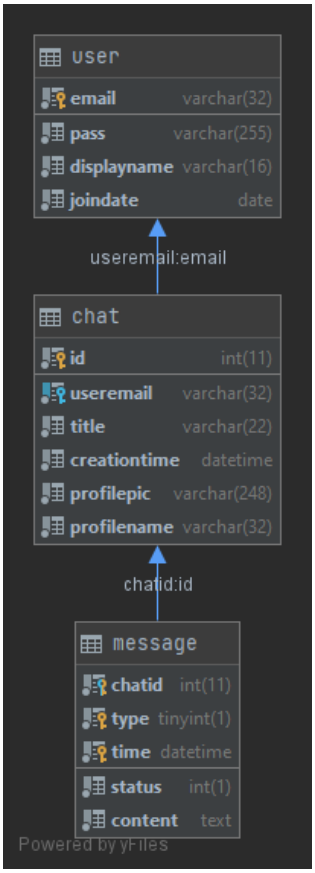


ILUSTRACIÓN 12: DIAGRAMA BASE DE DATOS

- Estructura de archivos backend:

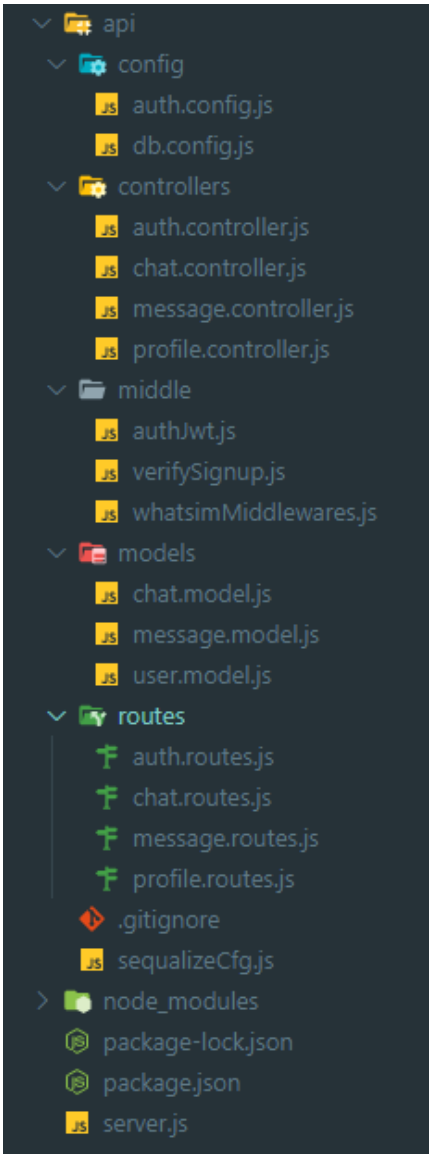


ILUSTRACIÓN 13: ESTRUCTURA BACKEND

- Estructura de archivos frontend:

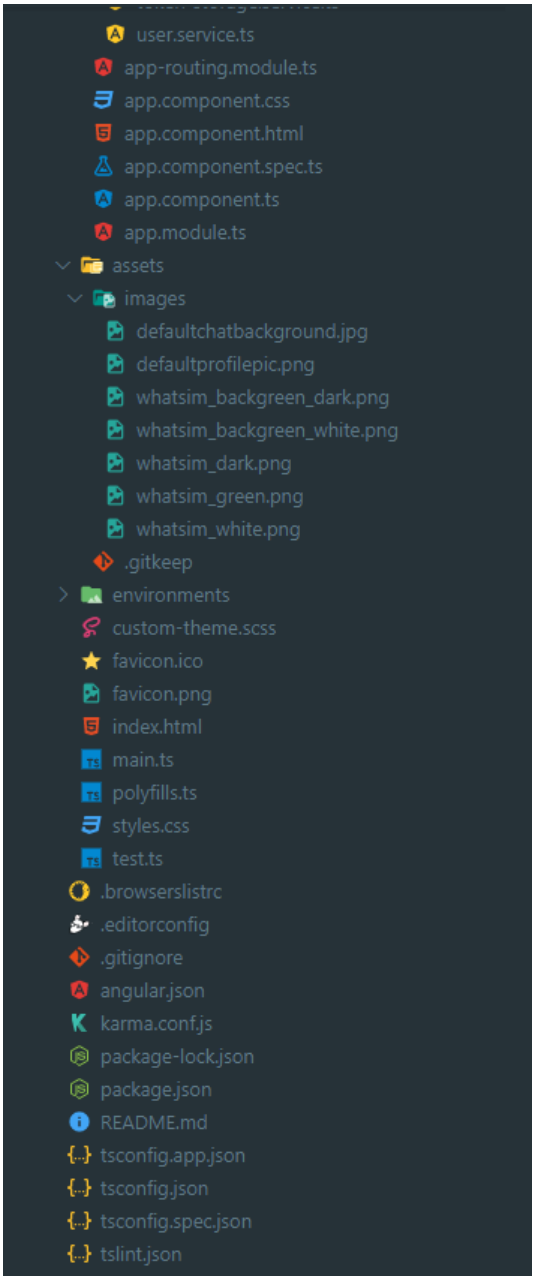


ILUSTRACIÓN 14: ESTRUCTURA FRONTEND

No creo que ninguna otra clase modelo sea muy revelante, ya que las clases modelo tanto del backend como del frontend son exactamente lo

mismo en ambos casos y que en la base de datos. Procederé a explicar la lógica de la aplicación

5.2.2 Lógica de aplicación

La lógica de aplicación se podría interpretar en resumidas cuentas como una aplicación cliente-servidor, aunque resulte poco intuitivo llamarlo así ya que tenemos dos servidores, o tres contando la base de datos, y el cliente sería el navegador web.

El cliente al entrar en la dirección de la aplicación, se esta conectando al servidor de angular, y este le está respondiendo, enviando todo lo necesario para ejecutar en su navegador.

Cuando se necesita hacer una petición al servidor backend, como, por ejemplo, cuando se envían los datos de inicio sesión y devuelve el resultado satisfactorio o negativo, es el propio navegador del cliente el que se conecta al servidor REST y hace la petición con los datos que él ha introducido en la interfaz de usuario y no el servidor angular, como se tiende a pensar.

Cuando el servidor REST backend envía el resultado de la consulta del usuario a su navegador cliente, angular lo intercepta, ya que evidentemente tiene que saber toda información de las respuestas para así poder proceder bien a mostrar los datos necesarios, o bien a mostrar el error específico.

Este sería el gráfico de interacción con la aplicación:

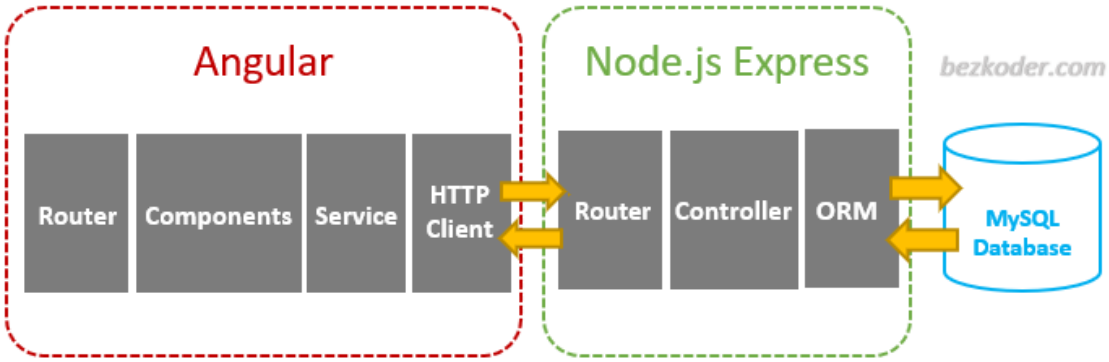


ILUSTRACIÓN 15:DIAGRAMA INTERACCIÓN

El backend tiene definidos una serie de errores que puede devolver como respuesta a peticiones de datos si algo va mal. Aquí están recogidos:

Error	Descripción
unknown_error	Error desconocido
account_not_found	Cuenta (email) no encontrada
account_wrong_credentials	Email o contraseña incorrectos
Invalid_web_token	Token de sesión invalido
account_already_exists	Email ya registrado
displayname_already_exists	Nombre ya registrado

chat_not_found	Chat no encontrado o no se puede acceder a él

El inicio de sesión es con Json Web Tokens (JWT). Es una tecnología bastante utilizada por grandes compañías como Amazon, simple y a la vez segura. Cuando un usuario inicia sesión, se le envía un token, y este será usado para todas las operaciones que requieran autentificación. Digamos que este token identifica al usuario. Este sencillo diagrama lo explicará mejor:

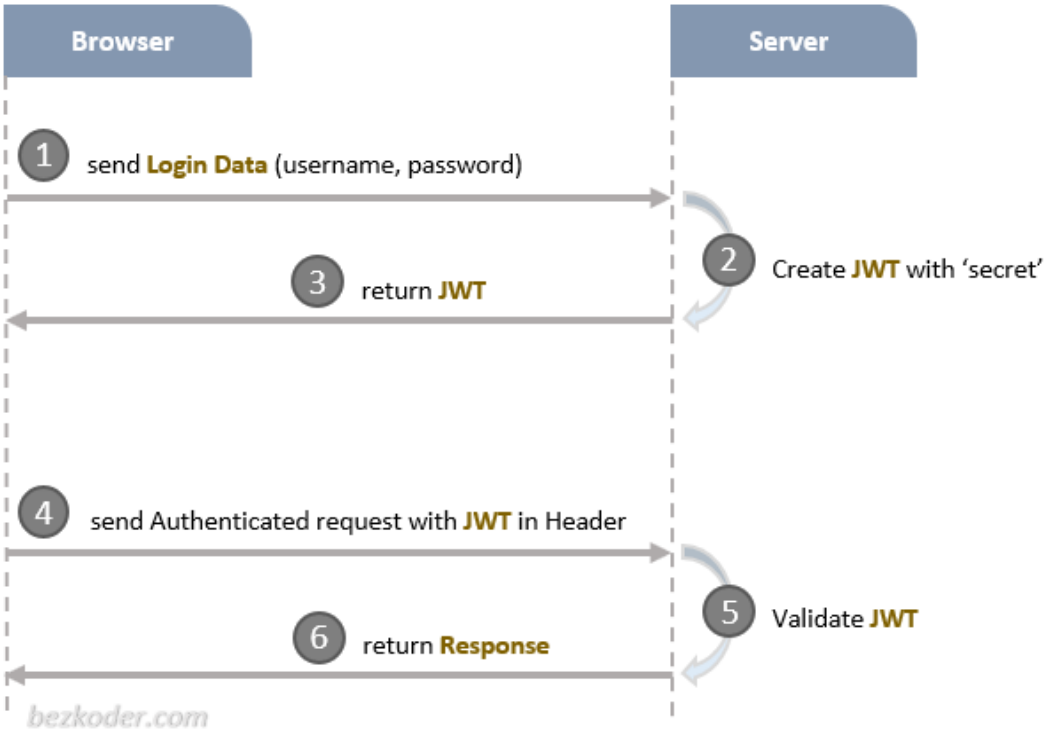


ILUSTRACIÓN 16: DIAGRAMA JWT

En resumen: El usuario interactúa con angular y a través de sus utilidades interactúa también con el servidor REST backend, este interactuando a su vez con la base de datos y devolviéndole resultados en json al usuario, esto es capturado y mostrado en forma de interfaz al usuario por angular.

5.3 Interacción con el usuario

El usuario interactúa con la interfaz de usuario, la parte frontend hecha con angular.

La interfaz esta principalmente creada usando angular material. Librería para angular que añade todo tipo de componentes en material design.

Como he dicho, al estar hecho con angular material, toda la interfaz de usuario es material design, diseño minimalista y moderno.

5.4 Código fuente

En este apartado expondré el código de las partes que considero más importantes tanto de la parte backend como de la parte frontend.

5.4.1 Código backend

De la parte backend mostraré el archivo principal (el que ejecuta node), una ruta y su respectivo contrador, y el middlewere que maneja los jwt y es utilizado en el contrador mostrado

- Archivo principal (server.js):

```
//Importación de todos los módulos necesarios
const express = require("express");
const bodyParser = require("body-parser");
const cors = require("cors");

const app = express();


app.use(cors());
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

//Conectamos sequilize
const db = require("./api/sequalizeCfg.js");
db.sequelize.sync({force: false, alter: false}).then(() => {
  console.log('Connected to Database');
});

app.get("/", (req, res) => {
  res.json({ message: "Whatsim API Rest" });
});

//Importamos las rutas que hemos creado
```

```
require('./api/routes/auth.routes')(app);
require('./api/routes/profile.routes')(app);
require('./api/routes/chat.routes')(app);
require('./api/routes/message.routes')(app);

//Definimos numero de puerto para poder manipularlo más facilmente
const PORT = 8080;
app.listen(PORT, () => {
  console.log("Whatsim Rest API Server running on port "+PORT);
});
```

- Archivo de inicialización y configuración del ORM sequilize (/api/sequilizeCfg.js): En este archivo podemos observar que tiene dos grandes bloques importantes: la parte de configuración de la conexión a la base de datos y la parte de configuración de los modelos (tablas) y relaciones entre ellas:

```
const config = require("./config/db.config.js");

const Sequelize = require("sequelize");
const sequelize = new Sequelize(
  config.DB,
  config.USER,
  config.PASSWORD,
  {
    host: config.HOST,
    dialect: config.dialect,
    operatorsAliases: false,

    pool: {
      max: config.pool.max,
      min: config.pool.min,
      acquire: config.pool.acquire,
```

```
      idle: config.pool.idle
    }
  }
);

const db = {};

db.Sequelize = Sequelize;
db.sequelize = sequelize;

db.user = require("./models/user.model.js")(sequelize, Sequelize);
db.chat = require("./models/chat.model.js")(sequelize, Sequelize);
db.message = require("./models/message.model.js")(sequelize, Sequelize);

/*CREAMOS LAS RELACIONES ENTRE USUARIO Y CHAT (OneToMany) */
//Un usuario tiene varios chats
db.user.hasMany(db.chat, {
  foreignKey: "useremail",
  sourceKey: "email",
  as: 'Chats'
});
//Un chat pertenece a un usuario
db.chat.belongsTo(db.user, {
  foreignKey: "useremail",
  targetKey: "email"
});

/*CREAMOS LAS RELACIONES ENTRE CHAT Y MENSAJE (OneToMany) */
//Un chat tiene varios mensajes
db.chat.hasMany(db.message, {
  foreignKey: "chatid",
  sourceKey: "id",
  as: 'Messages'
});
//Un mensaje pertenece a un chat
db.message.belongsTo(db.chat, {
  foreignKey: "chatid",
  targetKey: "id"
});
```

- Archivo de rutas de chat (/api/routes/chat.routes.js): Este archivo se encarga de administrar las rutas del chat y gestionar las peticiones y middleweres por los que tiene que ser verificada la petición para poder llevarse a cabo. Este archivo es cargado junto a los demás archivos de rutas en el fichero server.js. Éste consulta al controlador según la ruta:

```
/*
Archivo de rutas de chat
*/
//Importamos nuestro middle de JWT
const { authJwt } = require("../middle/whatsimMiddlewares");
//Importamos el controlador de chat
const controller = require("../controllers/chat.controller");

module.exports = function(app) {
    app.use(function(req, res, next) {
        res.header("Access-Control-Allow-Headers","x-access-
token, Origin, Content-Type, Accept");
        next();
    });

    app.post("/api/chat",
        [authJwt.verifyToken],
        controller.create
    );

    app.get("/api/chat",
        [authJwt.verifyToken],
        controller.getAll
    );

    app.get("/api/chat/:id",
        [authJwt.verifyToken],
        controller.getById
```

```
    );

    app.delete("/api/chat/:id",
        [authJwt.verifyToken],
        controller.delete
    );

    app.put("/api/chat/:id",
        [authJwt.verifyToken],
        controller.update
    );

    app.get("/api/chat/:id/detailed",
        [authJwt.verifyToken],
        controller.getDetailedById
    );
};
```

- Archivo controlador de chats (/api/controllers/chat.controller.js): Este archivo es el encargado de administrar los chats con la base de datos usando el ORM sequilize. En este caso el archivo es demasiado grande y solo pondré un método de ejemplo. En el anterior archivo (el de rutas) se puede contemplar cuantos métodos tiene:

```
/*
Este es el contralador de los chats. Desde aqui se manejara las peticiones de ch
ats y mensajes de un usuario
*/

//Importamos sequilize y modelos necesarios
const db = require("../sequalizeCfg.js");
```

```
const User = db.user;
const Chat = db.chat;
const Message = db.message;

//Crear chat nuevo
exports.create = (req, res) => {
    //Comprobamos el usuario con el token
    User.findOne({where: {
        email: req.email
    }}).then(user => {
        if(!user) return res.status(404).send({ message: "No acount" , error: "a
ccount_not_found"});
        else{
            Chat.create({
                useremail: user.email,
                title: req.body.title,
                creationtime: Date.now(),
                profilename: req.body.profilename
            }).then(chat => {
                if(chat)res.status(200).send({message: "Chat created ok" });
                else res.status(500).send({message: "An error ocurred", error: "
uknown_error" });
            }).catch(err => {
                res.status(500).send({message: err.message, error: "uknown_error
" });
            });
        }
    }).catch(err => {
        res.status(500).send({ message: err.message, error: "uknown_error" });
    });
};
```

Archivo de gestión de JWT (/api/middle/authJwt.js): Este fichero se encarga de la verificación del JWT de la petición. Es un middlewere, lo que significa que va entre medias, es decir: Una petición, por ejemplo, de creación de chats, se realizaría desde el controlador de chats como hemos visto anteriormente, pero para

ello se debe de estar autenticado, ahí es donde entra este middlewere, esta entre medias de la petición y el controlador que la gestiona. Este middlewere recibe una petición y comprueba que esta logueado buscando el JWT asociada ella, si no lo encuentra o es defectuoso se considera que no esta logueado y, en este caso concreto, como en todas las rutas de chat requerimos este middlewere, se devolvería un error. Si el token es correcto y si identifica a un usuario, se guarda la cuenta de usuario en la petición, para que pueda acceder a ella el controlador correspondiente:

```
/*
Este middlewere será el encargado de la autentificación con Json Web Tokens
*/
//Necesitamos importar la librería
const jwt = require("jsonwebtoken");
const config = require("../config/auth.config.js");

//Comprobamos el token (si lo hay) en el request
verifyToken = (req, res, next) => {
  let token = req.headers["x-access-token"];

  //Existe el token ?
  if (!token) {
    return res.status(403).send({
      message: "No token",
      error: "invalid_webtoken"
    });
  }

  //Comprobamos si el token es correcto y se lo asignamos al request
  jwt.verify(token, config.secret, (err, decoded) => {
    if (err) {
      return res.status(401).send({
        message: "Forbidden",
        error: "invalid_webtoken"
      });
    }
    req.email = decoded.usermail;
```

```
    next();
  });
};

const authJwt = {
  verifyToken: verifyToken,
};
module.exports = authJwt;
```

5.4.2Código frontend

La parte frontend es bastante más extensa, así que se me hace mucho más complicado elegir que archivos mostrar, pero creo que lo más sensato sería mostrarlos de la parte de chats, ya que en el ejemplo del backend he escogido lo mismo, aunque también incluiré otros ficheros importantes como el de rutas y el el archivo de lógica del componente propio de errores.

En el caso de angular, los dos grandes tipos de fichero, o más bien, elemento, serían componente y servicio. En este caso para los chats, es necesario emplear un servicio y varios componentes.

- Servicio de chats (chat.service.ts): Este es el archivo que maneja las peticiones de chat al servidor REST. Debe de ser

implementado por todos los componentes que lo requieran, lo veremos a continuación en el componente de lista de chats:

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Observable } from 'rxjs';
import { TokenStorageService } from './token-storage.service';

const API_URL = 'http://whatsim.eltrueno.es:8080/api/chat/';

@Injectable({
  providedIn: 'root'
})
export class ChatService {

  token: String;
  headers: any;

  constructor(private http: HttpClient,  private tokenStorage: TokenStorageServi
ce) {
    this.token = tokenStorage.getToken();
    this.headers = {headers:
      new HttpHeaders(
        {'Content-Type': 'application/json',
      'x-access-token': this.token.toString()
    })};
  }

  //Devuelve chats de un usuario
  getAll(): Observable<any> {
    return this.http.get(API_URL, this.headers);
  }

  //Devuelve en detalle (con mensajes) un chat especificado por su ID
  getDetail(id: any): Observable<any> {
    return this.http.get(API_URL+id+"/detailed", this.headers);
  }

  //Crear un nuevo chat especificando sus datos
  create(data: any): Observable<any> {
    return this.http.post(API_URL, data, this.headers);
  }
```

```
  //Borrar un chat especificando su ID
  delete(id: any): Observable<any> {
    return this.http.delete(API_URL+id, this.headers);
  }

  //Actualizar un chat especificando su id y sus datos nuevos
  update(id: any, data: any): Observable<any> {
    return this.http.put(API_URL+id, data, this.headers);
  }
}
```

- Lógica componente lista de chats (chatlist.component.ts): Este fichero es el encargado de la lógica de la lista de chats. Es bastante extenso:

```
import { Component, OnInit } from '@angular/core';
import { Chat } from 'src/app/models/chat.model';
import { ChatService } from '../../../services/chat.service';
import { DatePipe } from '@angular/common'
import { ConfirmationComponent } from '../../modals/confirmation/confirmation.co
mponent';
import {ChatmodalComponent} from '../chatmodal/chatmodal.component';
import { MatDialog } from '@angular/material/dialog';


@Component({
  selector: 'app-chatlist',
  templateUrl: './chatlist.component.html',
  styleUrls: ['./chatlist.component.css']
})
export class ChatlistComponent implements OnInit {

  chats: Chat[];

  hasError = false;
  errorShort: String;
  errorLong: String;

  constructor(private chatService: ChatService,public datepipe: DatePipe,public
dialog: MatDialog) { }

  private loadChats(){
    this.chatService.getAll().subscribe(
      data =>{
        this.chats = data.chats;
    }, err => {
      console.log(err);
      this.hasError = true;
      this.errorShort = err.error.error;
```

```
      this.errorLong = err.error.message;
    }
    );
  }

  ngOnInit(): void {
    this.loadChats();
  }

  private formatCreationTime(chat: Chat): String {
    return this.datepipe.transform(chat.creationtime, 'dd/MM/yyyy HH:mm');
  }

  private deleteInDb(chat: Chat): void{
    this.chatService.delete(chat.id).subscribe(
        data =>{
        console.log(data);
        this.loadChats();
      }, err => {
        console.log(err);
        this.hasError = true;
        this.errorShort = err.error.error;
        this.errorLong = err.error.message;
      }
    );
  }

  private showDeleteConfirmation(chat: Chat): void{
    const dialogRef = this.dialog.open(ConfirmationComponent, {
      data: {description: "¿Deséas eliminar permanentemente el chat?", deletion:
true}
    });
    dialogRef.afterClosed().subscribe(result => {
      if(result==="confirm"){
        this.deleteInDb(chat);
      }
    });
  }

  private editInDb(chat: Chat, data: any): void{
    this.chatService.update(chat.id, data).subscribe(
        data =>{
        this.loadChats();
      }, err => {
        console.log(err);
        this.hasError = true;
        this.errorShort = err.error.error;
        this.errorLong = err.error.message;
      }
    );
  }

  private createInDb(data: any): void{
    this.chatService.create(data).subscribe(
        data =>{
        this.loadChats();
      }, err => {
        console.log(err);
        this.hasError = true;
        this.errorShort = err.error.error;
        this.errorLong = err.error.message;
      }
    );
  }


  private showEditModal(chat: Chat): void{
```

```
    const dialogRef = this.dialog.open(ChatmodalComponent, {
      data: {chat: Object.create(chat)}
    });
    dialogRef.afterClosed().subscribe(result => {
      if(result){
        let editedChat:Chat = result;
        this.editInDb(chat, editedChat);
      }
    });
  }

  public showCreationModal(): void{
    const dialogRef = this.dialog.open(ChatmodalComponent, {
      data: {chat: {}}
    });
    dialogRef.afterClosed().subscribe(result => {
      if(result){
        let newChat:Chat = result;
        this.createInDb(newChat);
      }
    });
  }

}
```

- Plantilla de componente de lista de chats (chatlist.component.html): Este archivo es la maquetación del componente de lista de chats. Se comunica con el archivo de lógica y aplica los estilos de la hoja css definida en el archivo typescript chatlist.component.ts:

```
<whatsim-error *ngIf="hasError" type="dialog" error="{{errorShort}}"></whatsim-
error>
<div class="header-flex">
    <h2 style="text-align: center; align-items: center;">Tus chats</h2>
    <button mat-fab aria-
label="Crear un nuevo chat" color="accent" (click)="showCreationModal()" style="
width: auto; padding: 0 15px;border-radius: 50vw;">
        <mat-icon>add</mat-icon> Nuevo chat
    </button>
</div>
<div class="content-flex" style="margin-top: 2vh;">
    <mat-divider></mat-divider>
```

```
</div>
<div *ngIf="chats" class="content-flex" style="margin-top: 2vh;">
    <mat-accordion *ngFor="let chat of chats" style="margin: 1vh;">
        <mat-expansion-panel>
          <mat-expansion-panel-header>
            <mat-panel-title>
              {{chat.title}}
            </mat-panel-title>
            <mat-panel-description>
              Creación: {{formatCreationTime(chat)}}
            </mat-panel-description>
          </mat-expansion-panel-header>
          <div class="content-flex">
            <div class="row-flex"><b>Título:</b>{{chat.title}}</div>
            <div class="row-
flex"><b>Nombre del perfil:</b>{{chat.profilename}}</div>
            <div class="row-
flex" *ngIf="chat.profilepic"><b>Foto del perfil:</b>{{chat.profilepic}}</div>
            <button routerLink="/chats/{{chat.id}}" mat-raised-button aria-
label="Editar chat" color="primary" style="margin: auto; width: fit-content;">
                <mat-icon>launch</mat-icon> Abrir en detalle
            </button>
          </div>
          <mat-divider style="margin: 15px 0px;"></mat-divider>
          <div class="header-flex">
            <button mat-stroked-button aria-
label="Editar chat" color="accent" (click)="showEditModal(chat)">
                <mat-icon>edit</mat-icon> Editar
            </button>
            <button mat-stroked-button aria-
label="Editar chat" color="warn" (click)="showDeleteConfirmation(chat)">
                <mat-icon>delete</mat-icon> Eliminar
            </button>
          </div>
        </mat-expansion-panel>
    </mat-accordion>
</div>
<div *ngIf="!chats" class="header-flex" style="margin-top: 2vh; justify-
content: center;">
    <h3 style="text-align: center; align-
items: center;">No tienes ningún chat aún</h3>
</div>
```

- Archivo de gestión de rutas (app-routing.module.ts): Este fichero es creado al elegir la opción de ruta en la creación del proyecto angular y define las rutas y componentes que van a cargar las mismas:

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';;
import {AuthRedirectersService} from './services/auth-redirecters.service'

import { HomeComponent } from './components/home/home.component';
import {SignupComponent} from './components/signup/signup.component'
import {SigninComponent} from './components/signin/signin.component'
import {ProfileComponent} from './components/profile/profile.component'
import {ErrorpageComponent} from './components/error/errorpage.component'
import {ChatlistComponent} from './components/chat/chatlist/chatlist.component'
import {ChatdetailComponent} from './components/chat/chatdetail/chatdetail.compo
nent'

const routes: Routes = [
  {path: '', component:HomeComponent},
  {path: 'home', redirectTo:'/'},
  { path: 'signup', component: SignupComponent },
  {path:'signin', component: SigninComponent},
  {path: 'profile', component: ProfileComponent, canActivate: [AuthRedirectersSe
rvice]},
  {path: 'chats', component: ChatlistComponent, canActivate: [AuthRedirectersSer
vice]},
  {path: 'chats/:id', component: ChatdetailComponent, canActivate: [AuthRedirect
ersService]},
  {path: '404', component: ErrorpageComponent},
  {path: '**', redirectTo: '/404'}
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

- Archivo de lógica del componente propio de errores (error.component.ts):

```
import { Component, OnInit, Input } from '@angular/core';
import { TokenStorageService } from '../../services/token-storage.service';
import {Router} from '@angular/router';
import { MatDialog } from '@angular/material/dialog';
import { ErrormodalComponent } from './errormodal.component';

enum Error {
  unknown_error,
  account_not_found,
  account_wrong_credentials,
  invalid_web_token,
  account_already_exsists,
  displayname_already_exsists,
  chat_not_found
}

@Component({
  selector: 'whatsim-error',
  templateUrl: './error.component.html',
  styleUrls: ['./error.component.css']
})
export class ErrorComponent implements OnInit {

  @Input() error: String;
  @Input() type: String;
  @Input() description: String;

  errorType: Error
  message: String

  useDialog: Boolean = false;
  needSignout: Boolean = false;
  compVisible: Boolean = true;

  constructor(private tokenStorage: TokenStorageService, private router: Router,
 public dialog: MatDialog) { }

  ngOnInit(): void {
    try {
      var err:Error = (<any>Error)[this.error.toString()];
      this.errorType = err;
      switch (this.errorType) {
        case Error.unknown_error:
          this.message = "Ha ocurrido un error desconocido";
          break;
        case Error.account_not_found:
          this.message = "No se ha podido encontrar la cuenta";
          break;
        case Error.account_wrong_credentials:
          this.message = "Los credenciales son incorrectos";
          break;
        case Error.invalid_web_token:
          this.message = "El token de sesión es invalido";
```

```
          this.description = this.description==null? "Prueba cerrando sesión e i
niciando de nuevo. Se cerrará sesión automáticamente al cerrar este dialogo":nul
l;
          this.needSignout = true;
          break;
        case Error.account_already_exsists:
          this.message = "La cuenta ya está registrada";
          break;
        case Error.displayname_already_exsists:
          this.message = "El nombre de usuario ya se ha utilizado";
          break;
        case Error.chat_not_found:
            this.message = "El chat no existe o no es posible acceder a el";
            break;
        default:
          this.message = "Ha ocurrido un error desconocido";
          break;
      }
      if(this.type=="dialog"){
        this.useDialog = true;
        if(this.needSignout){
          this.openDialogAndSignout();
        }else this.openDialog();
      }
    } catch (error) {
      console.log("No se encontro error: "+error);
    }
  }


  openDialog(): void {
    const dialogRef = this.dialog.open(ErrormodalComponent, {
      data: { errorTitle: this.message, errorDescription: this.description }
    });
  }

  openDialogAndSignout(): void {
    const dialogRef = this.dialog.open(ErrormodalComponent, {
      data: { errorTitle: this.message, errorDescription: this.description }
    });
    dialogRef.afterClosed().subscribe(result => {
      this.tokenStorage.signOut();
      //window.location.reload()
      this.router.navigate(['/signin']);
    });
  }

  close(){
    this.compVisible = false;
  }

}
```

6. Fase de pruebas

Durante la fase de desarrollo, hacer pruebas a todo lo que iba haciendo era lo normal y se me hace imposible enumerar todo o ni siquiera

recordarlo. Lo que sí puedo recordar y enumerar es la fase de testeo final, con otras personas implicadas

6.1 Errores encontrados

Como dije, durante la fase final de testeo incluyendo a otras personas se lograron encontrar ciertos errores, algunos solucionados y otros no, a continuación, enumerados:

- El componente propio de errores no está integrado en inicio de sesión ni registro: No solucionado
- Botón de emojis, borrar chat y editar chat no funcionan porque no están implementados: No solucionado.
- Al enviar un mensaje con tabulaciones y saltos de línea, estos no se ven reflejados en los mensajes: Este problema fue resuelto aplicando ciertos atributos CSS en el componente del mensaje.
- Incorrecto funcionamiento del mensaje azul en el chat en detalle. El mensaje se debería mostrar cuando un mensaje era de diferente día al anterior, pero este solo se mostraba cuando

el tiempo entre ambos era de más de 24 horas, causando así que mensajes de diferente día con menos de 24 horas entre si pudieran parecer del mismo día y generando confusión: Este error ha sido solucionado cambiando el funcionamiento de ciertos métodos en el archivo de lógica del componente de chat en detalle.

7. Coste del proyecto

El coste del proyecto se reduce, principalmente al alquiler del servidor vps durante aproximadamente 4 meses.

El servidor cuesta 12 Euros al mes, iva incluido, asi que, multiplicado por cuatro, el coste de este sería de unos **48 Euros** en total.

También se podría incluir el dominio utilizado para el fácil acceso a la aplicación, pero ya que es un coste muy bajo y que tengo a mi cargo durante ya muchos años, he decidido no incluirlo.

8. Conclusiones

8.1 Reflexión

Tras todo el tiempo cumplido e invertido en este proyecto y aunque no se hayan podido cumplir muchos de los objetivos e ideas inicialmente planteadas, considero que el resultado final es, aunque mejorable, bastante bueno. Una aplicación en angular conectada a un servidor REST y base de datos SQL, con registro e inicio de sesión, que muestra chats emulando a la conocida aplicación Whatsapp.

Me ha gustado adentrarme en el mundo de angular. Aunque al principio cueste adaptarse a angular debido a su compleja curva de aprendizaje, una vez le vas cogiendo el punto te das cuenta de que no es tan difícil como parecía en un principio y de que es inmensamente potente y te proporciona soluciones simples y rápidas a muchos problemas comunes en este ámbito. Personalmente, es probable que haga más de algún pequeño proyecto más en angular para seguir aprendiendo esta tecnología, es más, seguramente continúe mejorando y ampliando este proyecto en concreto.

8.2 Futuras mejoras

Como posibles mejoras, no me queda más que recurrir a los objetivos no cumplidos citados con anterioridad en este documento en el apartado 2.3

9. Dificultades y problemas

El principal problema encontrado para mi fue la falta de tiempo y de inspiración. Al principio no sabía muy bien por donde empezar y me costó hacerlo.

Otro problema ha sido la curva de aprendizaje tan compleja de angular como ya he mencionado en más ocasiones.

10. Manual de instalación

El manual de instalación tanto de la parte backend como frontend, está publicado junto con todo el código fuente en dos repositorios de github en la organización https://github.com/RJG-WhatSim-TFG

- Servidor REST (backend): https://github.com/RJG-WhatSim-TFG/Backend
- Servidor Angular (Frontend): https://github.com/RJG-WhatSim-TFG/Frontend

11. Bibliografía

- https://www.digitalocean.com/community/tutorials/how-to-install-mariadb-on-debian-10
- https://docs.bitnami.com/aws/infrastructure/mariadb/configuration/create-database/
- https://tecadmin.net/install-angular-on-debian/
- https://senthilk979.medium.com/buidling-simple-todo-application-with-angular-8-and-node-js-e5be236086df
- https://www.positronx.io/angular-8-node-express-js-file-upload-tutorial/
- https://www.youtube.com/watch?v=O76FbfTbPH4
- https://morioh.com/p/2b0f211a33e6
- https://bezkoder.com/angular-11-node-js-express-mysql/

- https://bezkoder.com/angular-11-crud-app/
- https://bezkoder.com/node-js-express-angular-10-jwt-auth/
- https://bezkoder.com/angular-11-jwt-auth/
- https://bezkoder.com/node-js-jwt-authentication-mysql/
- https://blog.angular-university.io/angular-jwt-authentication/
- https://sequelize.org/master/manual/
- https://material.angular.io/
- https://www.freakyjolly.com/angular-material-modal-popup-example/
- https://bezkoder.com/node-js-sequelize-pagination-mysql/
- https://bezkoder.com/angular-11-pagination-ngx/
- https://codepen.io/
- https://github.com/scttcper/ngx-emoji-mart
- https://www.digicert.com/kb/ssl-support/apache-multiple-ssl-certificates-using-sni.htm
- https://angular.io/guide/deployment
- https://es.stackoverflow.com/
- https://google.com/
- https://www.w3schools.com/