Universidad Rey Juan Carlos

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA DE SOFTWARE

Curso Académico 2020/2021

Trabajo Fin de Grado

PLATAFORMA DE CHATS EN GRUPO IMPLEMENTADA EN ANGULAR Y NESTJS

Autor: Diego Pascual Ferrer

Director: Micael Gallego Carrillo

Agradecimientos

En primer lugar, agradecer a mis profesores de carrera y a mi tutor del trabajo de fin de grado por hacer esto posible.

A Ana, por animarme, apoyarme en los momentos de dificultad y comprender el tiempo del que disponía.

A mi familia y amigos, por interesarse en el TFG y sufrir los problemas conmigo y alegrarse por los momentos en los que “todo funciona”.

A Alejandro, por la ayuda introduciéndome al mundo del diseño de iconos y fondos, ayudándome cuando me superaba.

Resumen

El proyecto presentado, busca la creación de una aplicación web de chat moderno acompañado de una librería de componentes Angular, sobre el mecanismo de distribución de NPM. El objetivo principal es ofrecer una alternativa más visual a las librerías actuales de componentes relacionadas con los chats. La librería está planteada de modo que se pueda hacer una integración de los distintos componentes por separado. Los componentes que se ofrecen son: en primer lugar, un listado de chats con información sobre el estado del chat, incluyendo el último mensaje recibido, imagen, nombre y cantidad de mensajes sin visualizar; en segundo lugar, un componente más enfocado a la funcionalidad central de un chat, componiéndose de un área para el envío de mensajes, un bloque central dedicado a la visualización de mensajes y para finalizar un bloque superior enfocado a la información general del chat desplegado.

Un punto importante de esta librería es la posibilidad de adición de más componentes y la escalabilidad de los ya existentes (planteados en este proyecto), pudiendo incluir más funcionalidades dentro de estos en futuras versiones de la librería. La librería ha sido diseñada para Angular, al igual que la parte frontal de la aplicación sobre la que dicha librería sienta sus bases. El componente gráfico del chat se comunica mediante protocolos HTTP y Web Sockets con una parte trasera implementada en el framework NestJs para interactuar con los distintos usuarios y con la base de datos. En referencia a la base de datos se ha utilizado MongoDB, pero orientada a la nube.

El proyecto se ha planteado como una oportunidad de desarrollo completo de un chat con Web Sockets, el aprendizaje de una nueva tecnología como NestJs y la encapsulación de los apartados más útiles de dicha aplicación, que tengan referencia con la mensajería, para su futuro uso en diversos proyectos.

En este escrito se tratarán los siguientes puntos correspondientes a cada capítulo: en el primer capítulo se tratarán la motivación al desarrollo del proyecto y los objetivos que se pretenden alcanzar con el proyecto, el segundo capítulo abarcará la explicación de las metodologías usadas para afrontar el proyecto, las herramientas y las tecnologías en las que se sustenta el desarrollo del proyecto y finalmente las librerías y lenguajes usados para este desarrollo; en cuanto al tercer capítulo se explicarán los requisitos del proyecto, se repasará la arquitectura del proyecto mediante diagramas, se tratarán también los aspectos más relevantes en cuánto al diseño de la aplicación y de la librería, orientado a explicar las decisiones de diseño tomadas, los problemas encontrados, el diseño gráfico del proyecto y las pruebas. Como apartado final de la memoria se realizó una conclusión de los objetivos alcanzados, las futuras mejoras y conclusiones personales de los beneficios adquiridos al afrontar este proyecto. Añadido a esto, en la sección de anexos se adjuntaron detalles del funcionamiento de la librería y de la aplicación mediante la documentación realizada en el repositorio de Github para los futuros usuarios.

Índice general

Contenido

Capítulo 1 Introducción

En este capítulo se expondrán las motivaciones que hicieron nacer este proyecto, en que consiste, los objetivos y la evolución del proyecto a lo largo de su desarrollo.

I. Motivación

Dentro del marco del desarrollo web, un desarrollador que se enfrente a una aplicación completa se encuentra generalmente con una gran cantidad de apartados en los que centrarse: el desarrollo de la parte más visual, el desarrollo de la parte frontal, el de la parte trasera, que recibirá por lo general, peticiones de la parte mencionada anteriormente; todo ello para ofrecer al usuario una aplicación más interactiva. La realización de los test propios para cada una de las partes, control de versiones, despliegue de la aplicación en contenedores, registro de un dominio web para su despliegue, etc. Todo esto y más, supone una gran carga de trabajo para los desarrolladores. Es por esto por lo que van surgiendo una serie de facilidades que aligeran este trabajo, automatizando los apartados mencionados, como por ejemplo el caso de la integración continua con herramientas como Travis, que nos ejecutaría todos los test especificados en su fichero de configuración cada vez que se hace una subida a un repositorio. Otro ejemplo de esto son las librerías, código empaquetado que facilita la codificación de proyectos, con las librerías los desarrolladores pueden aprovechar el desarrollo de otras personas para aligerar el suyo propio.

De cara a acceder a estas librerías podemos encontrar gestores de paquetes como el conocido NPM, que se introduce en conjunto con la instalación de NodeJs, este gestor de paquetes se enfoca en código del lenguaje JavaScript. NPM es bastante relevante en el mundo del desarrollo web, debido a que este mundo tiene un claro favorito en su desarrollo y este es JavaScript, llegando a ser el lenguaje más usado según Stack Overflow (Stack Overflow, 2020) , como se puede ver en la Ilustración 1.

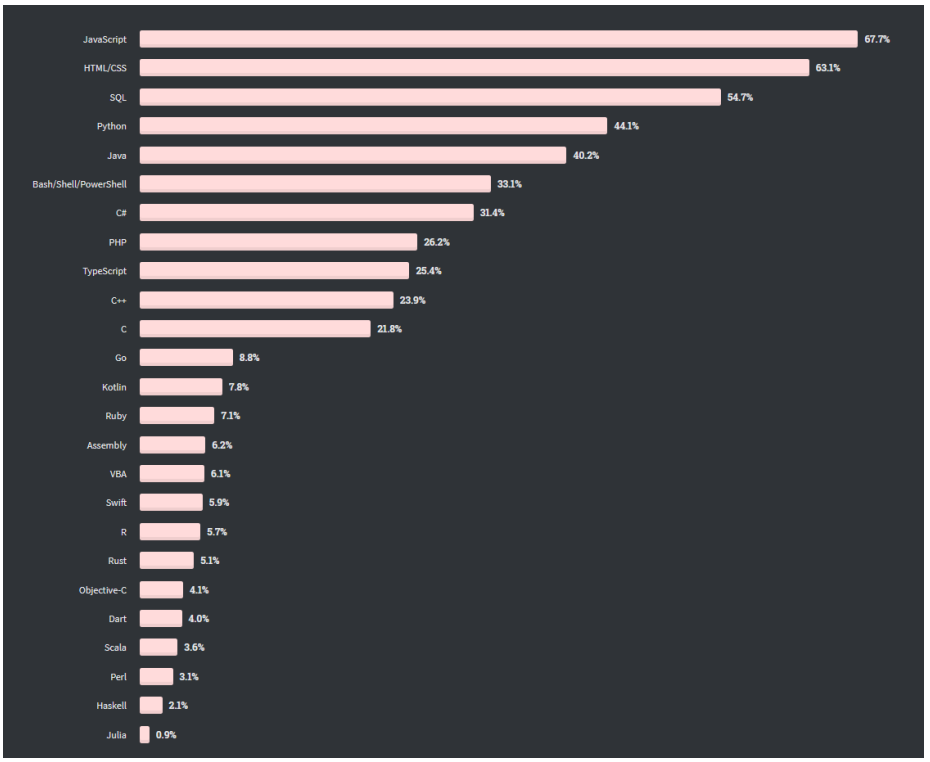


Ilustración 1 Encuesta de los lenguajes más usados en Stack Overflow 2020

Este gestor de paquetes favorece a tecnologías dependientes de este lenguaje, abarcando la parte frontal y pudiéndose usar sus librería en tecnologías como Angular, ReactJs o VueJs, y la parte trasera en frameworks como NodeJs o Nestjs.

Al comparar estas evidencias, se decidió enfocar el proyecto más aun en facilitar un poco más el trabajo a los desarrolladores web. Y, debido al rápido crecimiento de las necesidades de comunicación en el mundo actual; potenciado este año por la pandemia mundial del Coronavirus (portaltic, 2020), se centró el proyecto en relación con las aplicaciones de mensajería, y más en concreto con los chats.

En el día a día es bastante habitual encontrar un sinfín de aplicaciones web que se apoyan en la comunicación directa entre usuarios, como podrían llegar a ser las aplicaciones enfocadas a las redes sociales; o también, en la comunicación entre usuarios y robots que ayudan al usuario respondiendo dudas mediante comunicación por texto, se potencia su uso para ofrecer una asistencia directa en la situación del usuario en la aplicación. Por estas razones las aplicaciones que no contengan un chat meramente de asistencia pueden no atraer tanto el interés de los usuarios por la aplicación como otras aplicaciones que sí que lo tengan.

Debido a la complejidad de implementación de un chat en una aplicación web, requiriendo de conocimientos sobre envío de mensajes vía Web Sockets urge una librería que solucione estos problemas proporcionando una ayuda para codificar un chat de manera completa. De este punto es del que surgió la idea crear una librería para integrar con poca complejidad y sin requerir de un alto grado de conocimiento un chat. Dado que se pretende aportar el desarrollo a la comunidad se decidió que la licencia que se usaría sería Apache 2 para mantener el software de libre acceso, distribución y modificación.

Una vez decidido que se optaría por desarrollar una librería de chat, una de las decisiones más importantes fue establecer en que segmento del desarrollo debería suponer

un apoyo, pudiendo ofrecer soporte de servidor en el "backend" o un enfoque dedicado al "frontend", mediante componentes visuales que encapsularan la información del chat. Se optó por aportar una librería de la parte frontal. Finalmente se realizó una investigación de las librerías ya existentes, a modo de análisis de las ventajas y desventajas. Las librerías más relevantes analizadas son:

- Nebular Chat UI: es una librería que divide sus componentes entre el chat, el mensaje y el formulario de envío. Acepta envío de distintos formatos, permite arrastrar archivos para su reconocimiento, ofrece una serie de opciones de personalización con colores llamativos y se encuentra en proceso de desarrollo de las animaciones de los botones. El estilo de los componentes es bastante simple y poco modificable como se puede ver en la Ilustración 2, aunque tiene una gama de temas predefinida, los cambios realizados por los temas solo se reflejan en los marcos del componente. Añadido a esto, no ofrece la lógica para la realización de envíos de mensajes, solo es capaz de tratar los previamente introducidos en el chat.

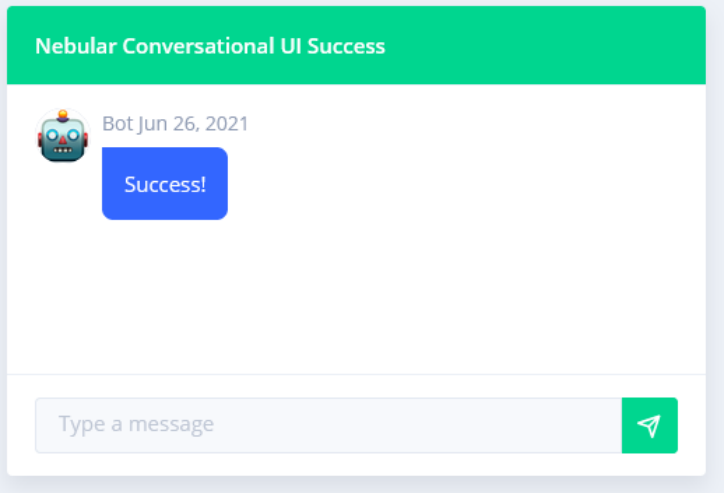


Ilustración 2 Librería Nebular Chat UI

- S-kalaus/ngx-chat-ui: es una librería con un único componente que empaqueta todo el chat. Aporta distintos esquemas sobre los que montar el chat y bastante modificables en cuanto a su estructura. El diseño gráfico es bastante simple como se puede visualizar en la Ilustración 3, poco agradable de cara al usuario y no ofrece temas para modificar sus colores y hacerlo más agradable.

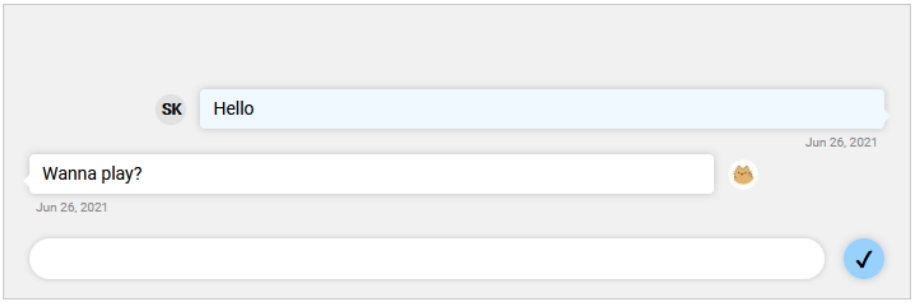


Ilustración 3 Librería s-kalaus/ngx-chat-ui

- Pazznetwork/ngx-chat: es una librería que ofrece el chat completo, incluyendo la API correspondiente para el envío de mensajes empaquetando todo por completo en un único componente. Acepta el envío de mensajes de texto plano pero no de archivos aunque permite al desarrollador integrar todo de una manera bastante simple. En cuanto al diseño tiene una interfaz de usuario bastante deficiente y no ofrece temas para mejorar el apartado visual, como se puede ver en la Ilustración 4; añadido a esto está encapsulado libre de modificaciones.

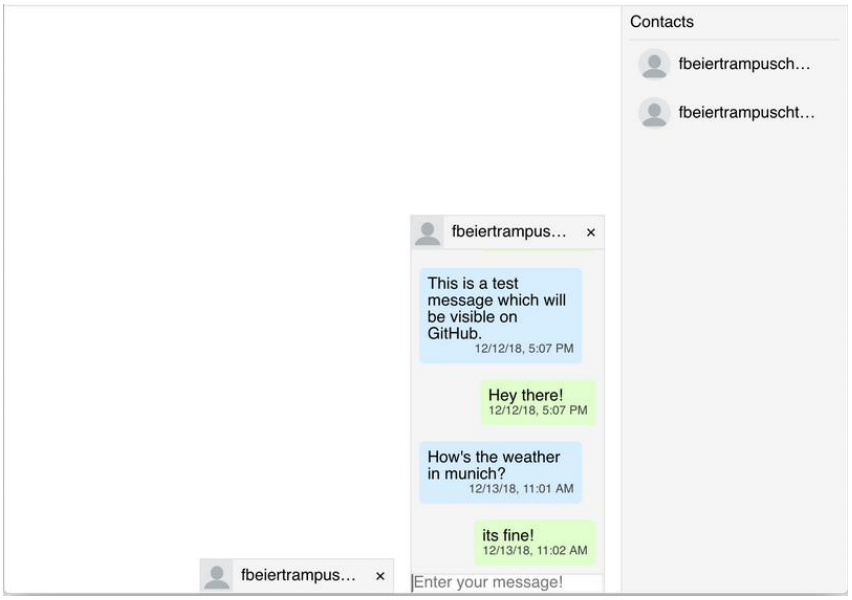
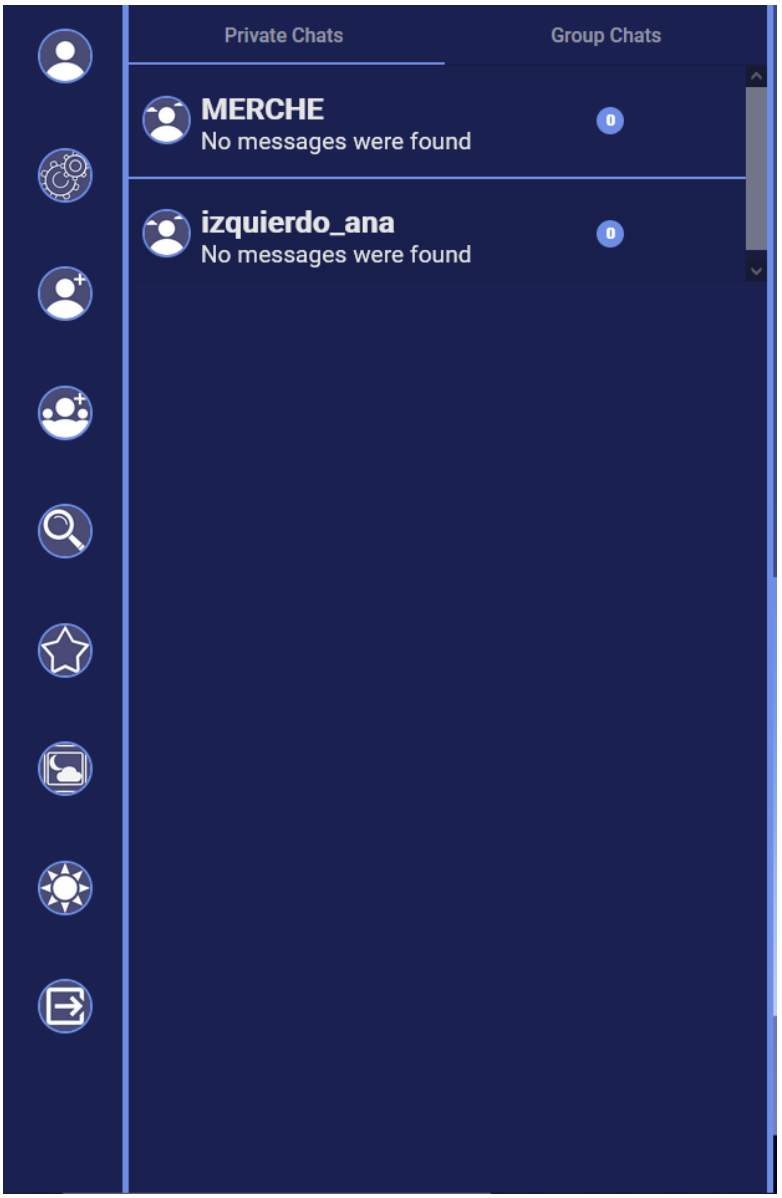


Ilustración 4 Librería pazznetwork/ngx-chat

Una vez analizadas las librerías, se llegó a la conclusión de que la librería creada debería aportar al usuario una interfaz más agradable, con mayor posibilidad de modificación en cuanto al diseño y que se encargara de los envíos de mensajes y archivos por completo. A modo de conceptualización del estilo de la librería se explicará a continuación en las imágenes de las pantallas 1 y 2 los componentes ChatList y ChatBox respectivamente y se hará un análisis de la interfaz de dichas pantallas.

En la Pantalla 1 se pueden visualizar dos secciones delimitadas por una barra vertical, la sección izquierda corresponde con el componente de opciones no presente en la librería, pero pertinente para explicar el estilo de la aplicación, desenvuelve tareas de ajustes del modo diurno, nocturno, creación de chats privados y grupales, cambio del perfil del usuario y funcionalidades que se implementarán en posteriores revisiones del proyecto. La sección derecha corresponde con el componente ChatList, con dos opciones, chats privados y grupales, cargados al momento por uno de los botones superiores. Este componente contiene los chats del usuario, estos chats están compuestos por la imagen del perfil del usuario en caso de ser un chat privado o la imagen del chat grupal, el nombre del chat, o en caso de ser un chat privado el nombre del usuario con el que se comunica, el último mensaje enviado y finalmente, la cantidad de mensajes sin leer.



Pantalla 1 Pantalla con la barra de opciones y el componente ChatList en modo nocturno

La Pantalla 2, muestra el componente ChatBox al completo, este se divide en tres partes: la superior que muestra la imagen del chat siguiendo los mismos criterios que ChatList, el nombre que se puede pulsar para activar un diálogo con información sobre el chat. La sección central que muestra los mensajes de texto, los links, las imágenes que se pueden ampliar en su versión reducida, para su posterior ampliación tras pulsarlo, los archivos de los diversos formatos y las imágenes que superen un determinado tamaño definidos por el icono de archivo, que tras pulsarse se abre en otra pestaña para abrirlo o descargarlo y en el caso de los vídeos y las imágenes se abre en pantalla completa y se reproduce. Y, para terminar, la sección inferior contiene un botón que activa el envío de archivos, o bien mediante un clic o bien arrastrando el archivo, el menú de emoticonos, el área de texto y finalmente el botón de envío del mensaje.

Pantalla 2 Pantalla del componente ChatBox en modo nocturno

II. Objetivos

En este apartado se tratarán los objetivos principales de los que parte el proyecto. Se pueden sintetizar en cinco puntos, enumerados a continuación y que se atuvieron a un desarrollo secuencial fijado por estos puntos y que corresponden con las etapas principales por las que se desarrolló el proyecto:

1. **Desarrollar una aplicación web simple,** compuesta por un servidor, una base de datos y un cliente, las funciones que cumpliría la aplicación en este punto son las de envío de mensajes de texto simples en tiempo real, creación de chats grupales, chats privados y la modificación del perfil del usuario.

2. **Mejora visual,** este apartado correspondería a la mejora del diseño de la aplicación mediante el diseño personal de los iconos de la aplicación, fondos de la aplicación y la introducción de distintos modos visuales: el modo diurno y el modo nocturno, todo este proceso se realizaría haciendo uso de herramientas de diseño gráfico.

3. **Introducción de funcionalidades en la aplicación,** una vez ya sentadas las bases de la aplicación corresponde incrementar la completitud de esta. Por ello en esta fase se introduciría el envío de imágenes, vídeos, archivos de texto, documentos para su descarga en el caso de los archivos y para su visualización en el caso de las imágenes y los vídeos. Añadido a las mejoras en el tipo de formatos aceptados para los envíos, se mejoraría la

comunicación vía texto añadiendo la detección de enlaces para progresar en cuanto al apartado visual. Para esto se debería contar con un apartado en el que se pudieran arrastrar los archivos a una zona y que fueran recogidos por parte de la aplicación. Finalmente se ofrecería al usuario la posibilidad de desplegar menús con información sobre los chats y se añadiría el uso de emoticonos mediante un menú que permita al usuario escoger el emoticono ideal, y a su vez permita al usuario tener una colección de los más recientes. Todas estas mejoras en el apartado visual se acompañarían de su correspondiente implementación en el servidor para dar respuesta a la actualización del lado del cliente.

4. **Creación de la librería,** en este momento se realizaría un análisis de los componentes introducidos en la aplicación, cuyo posible uso en la librería fuera el óptimo, de cara al desarrollo de un aplicación web de chat. Se implementarían dichos componentes encapsulados en la librería, partiendo de los componentes de la aplicación. Se procedería también a realizar la publicación de la librería en el gestor de paquetes NPM. Para finalmente, realizar la documentación de la misma para facilitar su uso.

5. **Introducción de la librería en la aplicación,** a modo de finalización y comprobación de la utilidad de la librería se realizaría una refactorización de la aplicación susituyendo el código requerido por la utilización de la librería generada en el objetivo anterior.

Capítulo 2 Metodologías, tecnologías y herramientas

En este apartado se explicará en detalle las metodologías, tecnologías y herramientas usadas en el proceso de desarrollo del proyecto. Se procederá a explicar previamente las metodologías, posteriormente las tecnologías usadas y finalmente las herramientas usadas para el desarrollo del proyecto.

I. Metodología

En este apartado se tratará el proceso que ha seguido el desarrollo del proyecto. Este proyecto se desarrolló siguiendo un proceso iterativo e incremental, siempre respetando el desarrollo en profundidad de las partes principales pertinentes. En otros términos, se realizó un prototipo de un tamaño reducido y funcional, con la finalidad de ir escalando las funcionalidades de este, mediante las diversas iteraciones que se realizaron hasta llegar a la fase final. Esto se logró siguiendo también un desarrollo incremental en profundidad hasta finalizar una funcionalidad concreta. Un ejemplo de esto es la creación del servidor de NestJs y sus funcionalidades más simples de una manera incremental en su etapa correspondiente, sin codificar más de la parte frontal hasta no alcanzar la finalización de esta etapa del lado del servidor.

La metodología usada concretamente fue Scrum (Kniberg, 2015), comprendida dentro de los marcos de la agilidad. La elección de este framework frente a otros reside en la necesidad de ir entregando software de calidad, dónde los requisitos son cambiantes y se ejecute en ciclos cortos y con una duración fijada de dos semanas, en este caso los ciclos se fueron adaptando a la estimación propuesta para cada sprint, comprendiendo en algunos casos dos meses y en otros una semana, en función de las tareas a realizar. La aplicación de este framework más clara fue la constante realimentación recibida por parte del tutor y personas cercanas a las que se les enseñó el producto al final de cara iteración. Esto supuso el cambio de requisitos constante, la priorización de otros requisitos.

De este modo se consiguió seguir un flujo de trabajo que dio cabida a la experimentación, partiendo de la introducción de nuevas tecnologías, inclusive introduciendo tecnologías que no están completamente maduras, en otras palabras, su soporte no es tan fuerte como el de tecnologías más asentadas. Añadido a lo mencionado anteriormente, esta metodología permite tener una visión global del estado del proyecto suponiendo un avance progresivo del proyecto en su completitud. Concretamente, se usaron sprints de 7 días a los que se les asignaban varias tareas, siguiendo la metodología Scrum y completando diversos objetivos en cada sprint, estos sprint se agruparon en fases correspondientes al desarrollo que se realizaba pudiendo contener algunas fases varios sprints y otras fases un solo sprint. Las fases han sido las siguientes:

- Fase 1: Desarrollo de una aplicacion (60 días)

    1. Se desarrolló un boceto inicial de la estructura principal del apartado

visual que ofrecería la aplicación, incluyendo los colores escogidos. Este boceto inicial se realizó haciendo uso de herramientas de diseño como Procreate.

2. Se realizó un primer prototipo frontal sobre lo que se apoyaría posteriormente la aplicación. Este prototipo se realizó usando Angular.

3. Una vez sentadas las bases del diseño principal, se desarrolló un servidor en NestJs con las funciones principales con las que contaría el prototipo inicial del proyecto. A medida que se implementaban las funcionalidades en la parte trasera del proyecto se creó la base de datos en MongoDB Atlas, para ofrecer una base de datos en la nube.

4. En este punto se dotó de funcionalidades la parte frontal del proyecto para comenzar a establecer la comunicación entre ambas partes y poder ofrecer un prototipo inicial completamente funcional.

5. Se desplegó la aplicación completa haciendo despliegue individual de la parte frontal y la parte trasera mediante Docker, no fue necesario el despliegue de la base de datos debido a su condición siempre desplegada en la nube.

- Fase 2: Mejora visual (15 días)

6. Se diseñaron los fondos e iconos de la aplicación en los diversos colores.

7. Se definieron los temas propios en el código para hacer referencia a los nuevos fondos e iconos de la aplicación.

- Fase 3: Introducción de nuevas funcionalidades en la aplicación (15 días)

8. Se integraron en el proyecto las funcionalidades propuestas en la revisión previa, integrando individualmente las funcionalidades en todo el proyecto, parte frontal, parte trasera y base de datos. Las funcionalidades implementadas son: envío de emoticonos, envío de archivos, envío de links clicables, representación de imágenes enviadas en pantalla completa.

- Fase 4: Creación de la librería (21 días)

9. Al finalizar esta última integración de funcionalidad en la aplicación, se procedió a crear la librería que contendría los componentes.

10. Una vez ya creada la librería dentro del proyecto principal se implementó el componente que contendría la lista de chats llamado “ChatList”.

11. Después se procedió a actualizar la información en el archivo README correspondiente a la librería.

12. En el siguiente paso se implementó el componente que contendría el chat desplegado y que portaría gran parte de la lógica de la aplicación, llamado “ChatBox”.

13. En este último paso de este sprint se actualizó la librería añadiendo al archivo README información del segundo componente.

- Fase 5: Introducción de la librería en la aplicación (7 días)

14. Se procedió a cambiar la aplicación e introducir el componente ChatList de la librería en la aplicación sustituyendo las dependencias existentes y adaptando el componente que contendría ese apartado de la libería.

15. Se introdujo el componente ChatBox de la librería en la aplicación y se hicieron los cambios pertinentes para integrarla correctamente.

16. Por último se realizó la memoria miéntras se realizaban los ajustes necesarios en la aplicación y la librería se publicaba al NPM.

II. Tecnologías

En este apartado se procederá a explicar las diversas tecnologías usadas en el proyecto, tanto para el desarrollo de la librería como para el desarrollo de la aplicación. Las tecnologías usadas en el proyecto son: Angular para el desarrollo del apartado frontal de la aplicación, NestJs para el desarrollo del servidor en la parte trasera, MongoDB Atlas como base de datos alojada en la nube y Docker para el despliegue inicial de la aplicación en un contenedor.

i. Angular

Angular es un framework de JavaScript open source desarrollado por Google, enfocado en la creación de aplicaciones web de una sola página, más conocidas por sus siglas en inglés SPA (Angular Team, 2020). Fue lanzado en 2016, conocido como Angular 2.0 debido a que es una reelaboración de AngularJS (Angular versión 1.0) que se presentó en 2010. El lenguaje de programación principal de Angular es TypeScript, un superconjunto de JavaScript/ECMAScript. Este framework además tiene como característica principal el uso del modelo vista controlador MVC.

El principal competidor de esta tecnología es ReactJS, por ello se realizará un análisis de las dos tecnologías a modo de comparativa para visualizar los beneficios de Angular frente a frameworks como ReactJS (Delgado, 2020).

En cuanto a la autosuficiencia, Angular es un framework completo que ya cuenta con funciones como enlazado de datos, enrutamiento de componentes, generación de proyectos, validación de formularios, inyección de dependencias, todo esto supone un gran avance frente a ReactJS debido a que React es una librería para el desarrollo UI, por esta razón requiere de librerías adicionales para lograr las mismas funcionalidades base que Angular.

En cuanto al rendimiento, React ha mejorado con la introducción del DOM virtual y Angular se ve afectado por el enlazado bidireccional de datos, haciendo que cuantas más vinculaciones tenga más observadores se creen y más se ralentice el proceso. Sin embargo, con la última actualización de Angular se ha equiparado en este aspecto con el rendimiento de React.

Haciendo referencia al lenguaje ambos frameworks permiten una detección de errores tipográficos ágil. Por un lado, React se apoya en JavaScript ES6 combinado con JSX y aporta una mayor legibilidad del código. Por otro lado, Angular permite una navegación por el código más rápida y por ello el proceso de refactorización y mantenimiento se vuelve más simple y veloz que en el caso de React.

Vinculado al concepto de la estructura de la aplicación React ofrece a los desarrolladores la libertad de elegir la estructura correcta para su desarrollo. Esto supone que los inicios de los proyectos sean más lentos. Además, React no ofrece el modelo y controlador por defecto, solo ofrece la capa correspondiente a la vista, las dos mencionadas se agregan mediante librerías. Por el contrario, Angular ofrece una estructura fija pero compleja. El código en Angular consta de diferentes componentes compuestos de un archivo HTML que corresponde a la vista, un archivo CSS o SCSS para definir estilos, los datos que se representarán en el componente que conforman el modelo y finalmente el archivo componente que corresponde con el controlador.

Valorando los componentes de la interfaz de usuario se identifica a React como un framework orientado a la comunidad debido a la gran oferta de librerías de pago y gratuitas, pero requiriendo de una instalación adicional de cara a usar componentes como los existentes en Material Design. En el caso de Angular proporciona ya un Material Design preconstruido. Debido a esto, el proceso de configuración de la interfaz de usuario se vuelve menos tediosa.

Una de las mayores ventajas de Angular frente a React es que permite la inyección de dependencias permitiendo la coexistencia de diferentes ciclos de vida para diferentes almacenamientos.

Como se mencionó previamente una de las debilidades más importante de Angular es el enlazado de datos bidireccional que afecta negativamente al rendimiento, ya que Angular desarrolla de manera automática un observador por enlace.

En materia de herramientas React requiere de varias herramientas para poder realizar una prueba del código completa, requiriendo de Enzyme en el caso de las pruebas de componentes, Jest para probar el código JS, React-Unit para las pruebas unitarias. A diferencia de React, en Angular se puede realizar una prueba de extremo a extremo usando una única herramienta, y las plataformas para esto son Jasmine, Protractor y Karma.

Por último, en cuanto al cambio de renderizado supone un gran avance por parte de React la utilización de un DOM virtual frente al DOM real en el que se apoya Angular, usando la detección de cambios para identificar los componentes que requieren una actualización.

Atendiendo a estas consideraciones se llegó a la conclusión de que cada una de las dos opciones podría ser una buena opción, pero debido al rango de la aplicación y su tamaño, el problema sobre los observables mencionado anteriormente no supondría un problema debido a la pequeña cantidad de componentes involucrado en este proceso. Teniendo en cuenta la simplicidad de React se prefirió abordar el proyecto apoyándose en un framework más completo desde el inicio y que permite una configuración más veloz de respecto al diseño UI gracias a la librería Angular Material. Por estas razones y por la experiencia del desarrollador con la tecnología Angular se escogió el uso de esta última.

ii. NestJs

NestJs es un framework desarrollado en el lenguaje TypeScript que sirve como punto de partida para el desarrollo con NodeJs de manera escalable y está enfocado en el servidor. NestJs se creó en 2017 como una alternativa más madura al desarrollo realizado con Express. Ya que esta es una de las desventajas más conocidas de NodeJs, la falta de orden y un estándar en la construcción de soluciones sólidas y extensibles. Esta mejora por parte de NestJs proviene en parte de la utilización de TypeScript por defecto, aunque se pueda llegar a usar JavaScript para el desarrollo, se recomienda el uso de TypeScript que aporta capacidades como las que se pueden encontrar en lenguajes como Java o C#.

El formato de trabajo en NestJs puede resultar familiar al encontrado en Angular en relación con la creación del proyecto por la línea de comandos y la estructuración del proyecto, reflejándose esto incluso en la instalación de la línea de comandos de ambas tecnologías.

A su vez se puede encontrar una gran relación con frameworks como Spring Framework salvando las distancias que existen entre los lenguajes usados por los dos frameworks mencionados, Java por parte de Spring y TypeScript por parte NestJs. Un ejemplo de estas similitudes es el uso de anotaciones en los servicios con la anotación **`@Service`** en Spring y **`@Injectable`** en NestJs, esta similitud se hace más clara en el uso de controladores anotados como **`@Controller`** en ambas tecnologías.

Una de las razones más claras para la preferencia de NestJs frente al resto de tecnologías mencionadas es la facilidad que aporta al tener una integración de Web Sockets totalmente compatible con la usada en Angular ngx-Socket.io. Esto se hace más sencillo inclusive al hacer uso de sus WebSocketGateways que lo hacen compatible con todas las librerías disponibles siempre y cuando se use un adaptador. En la situación ofrecida por el proyecto no sería necesario el uso de un adaptador ya que NestJs tiene dos plataformas soportadas listas para usar Socket.io y ws. Como se mencionó anteriormente en este caso se hará uso de Socket.io que se explicará en detalle más adelante.

iii. MongoDB Atlas

MongoDB Atlas es un servicio global de base de datos de tipo Cloud, que permite crear clústeres completos de bases de datos, sin necesidad de una configuración avanzada exponiéndolas a su conexión por parte de las aplicaciones sin requerir de tareas de instalación y administración.

Una vez ya explicado que es MongoDB Atlas queda aclarar porque se hizo uso de una base de datos no relacional. Se tomó la decisión de hacer uso de bases de datos no relacionales debido a la posibilidad de mantenimiento de documentos con un modelo más flexible, posteriormente se tratará de manera específica la implementación de un modelo con distintos campos, a modo de ejemplo de esta ventaja frente a las bases de datos relacionales. Esta facilidad de cara a mantener estructuras heterogéneas perteneciendo al mismo modelo ha sido la razón de peso para la preferencia de este tipo de base de datos.

Lo anteriormente expuesto supuso la preferencia de bases de datos no relacionales frente a las bases de datos relacionales y concretamente la elección de MongoDB frente a otras bases de datos no relacionales debido al gran soporte que ofrece, la comunidad extensa que ha logrado y la documentación que aporta de cara a iniciarse en su uso. Este tipo de

base de datos es de tipo documental. Y dentro de las bases de datos no relaciones documentales es de las más populares.

iv. Docker

Docker es una plataforma de software que nos permite empaquetar software en unidades estandarizadas llamadas contenedores que incluyen todo lo requerido por el software que se empaqueta inclusive librerías, un sistema operativo y un tiempo de ejecución (Garzas, 2015).

Docker se postula como una alternativa a las máquinas virtuales siendo capaz de aislar el sistema operativo de un servidor mientras que las máquinas virtuales virtualizan el hardware del servidor. Esto permite que se puedan ejecutar en cualquier máquina con Docker instalado, independientemente del sistema operativo que la máquina tenga internamente, de las versiones que tenga instaladas de las diversas tecnologías. Todas estas facilidades hacen de Docker un entorno autocontenido, que permite la ejecución en diversas máquinas sin requerir de una configuración previa.

III. Lenguajes

En este apartado se explicarán los lenguajes usados para el desarrollo del proyecto en las diversas tecnologías. Para este proyecto la predominancia de un lenguaje ha sido clara. Este lenguaje ha sido TypeScript.

i. TypeScript

TypeScript es un lenguaje de tipo open source, nacido en 2012 a modo de solución por parte de Microsoft para el desarrollo de aplicaciones a gran escala, tanto para Microsoft de manera interna como para sus clientes.

TypeScript se basa en JavaScript, pero añadiendo como objetivo principal las definiciones de tipos estáticas, este lenguaje provee una forma de describir el formato de un objeto, o lo que es lo mismo provee de una manera de proporcionarle un tipo al objeto. Especificar el tipo de los objetos es opcional, no requiere de ese proceso debido a la inferencia de tipos potente que ofrece sin necesidad de escribir código adicional. Este lenguaje es un superconjunto de JavaScript. En resumen, esto supone que los programas de JavaScript son programas también válidos de TypeScript, a pesar de que sean dos lenguajes distintos.

Este lenguaje se compila en código JavaScript mediante el compilador propio de TypeScript. Al ser código que se compila a JavaScript este código puede ejecutarse en cualquier lugar en el que se pueda ejecutar JavaScript, pudiéndose ejecutar en el navegador, en NodeJs, etc. Este aspecto beneficia a TypeScript poniendo a disposición de los desarrolladores de las librerías y frameworks disponibles para JavaScript como ReactJs, VueJs, Angular, etc.

Entrando en detalle en el tema del tipado estático versus el tipado dinámico cabe destacar como punto fuerte del tipado estático que el uso de variables tipadas ofrece al programador una forma de auto documentar el código y proporciona de un mejor análisis para detectar errores existentes en la aplicación. Esto no quiere decir que no permita evitar

el tipado estático, como se mencionó previamente es un superconjunto de JavaScript por lo que el lenguaje da la opción al desarrollador de escoger el momento de usar una u otra característica, pudiendo usar en el mismo archivo ambas.

La razón de su predominancia en el proyecto reside en que el lenguaje de preferencia para ambas tecnologías, Angular y NestJs, es TypeScript. Esto supone una facilidad en el momento de codificación del proyecto no siendo necesario un cambio de lenguaje de cara a desarrollar la parte frontal y la parte trasera.

Cabe destacar como mención especial el uso del lenguaje JSON que es un formato de texto sencillo usado en el intercambio de datos y muy presente en el desarrollo web. Este se puede usar en proyectos JavaScript y TypeScript, aunque debido a su popularidad se procedió a su introducción en distintos lenguajes como C#, Java y Python entre otros muchos.

IV. Librerías

En este apartado se entrará en detalle en las librerías más relevantes usadas en el desarrollo tanto de la aplicación como de la librería construida a raíz de dicha aplicación.

i. Angular Material

Angular Material es una librería de componentes de interfaz de usuario que se basa en la guía de diseño de componentes de Material Design y se desarrolló por el equipo de Angular explícitamente para su integración en Angular. Material Design que es de dónde parte esta librería tuvo su origen en 2014 y fue inventado por Google. Esta librería proporciona una serie de componentes que tienen que ser importados en el módulo de manera específica.

A modo de comparación con otras librerías se realizará un análisis de los beneficios y debilidades de esta librería y posteriormente sobre su competidora Bootstrap. Los puntos fuertes son: su formato adaptativo, que adapta sus dimensiones a la pantalla en la que se encuentra. Esto es punto favorable de cara a su uso en Angular ya que esta tecnología se puede usar tanto para desarrollo de aplicaciones móviles como para desarrollo web. Tiene un diseño plano y minimalista. Está planteado para su uso mediante la aplicación de estilos básicos a dichos componentes, requiriendo de poco conocimiento para su utilización. Este último detalle a priori debería contabilizar como un punto a favor de esta librería, pero no es el caso, ya que, en el momento de requerir hacer modificaciones de los componentes de la librería, se obliga al usuario a evitar totalmente la encapsulación. A veces requiriendo de instrucciones de alta especificación como `!important` esto supone un problema cuando se dedica gran parte del tiempo al estilo de los componentes. A continuación, se procederá a explicar Bootstrap.

Bootstrap es una librería open source con su enfoque principal en usuarios que se inician en el desarrollo frontal. Es un framework intuitivo y específico de CSS, aunque, la mayor parte del código recaiga en las clases procesadas de CSS las plantillas en HTML son necesarias. Sus ventajas residen en los múltiples diseños ofrecidos por la comunidad, su facilidad de modificación.

En líneas generales, usar Bootstrap facilitaría un desarrollo sencillo de componentes simples, pero al estar trabajando con una interfaz más compleja usar una librería como

Angular Material, requiriendo solo de los componentes estrictamente necesarios reduce el tiempo de búsqueda de componentes en la comunidad de Bootstrap y de su adaptación al estilo de la aplicación.

Por ello, se hizo uso de la librería Angular Material, estrictamente para los componentes que necesitarán de modificaciones ligeras, siendo esta una de las fortalezas de Angular Material. En el caso de necesitar componentes con una alta modificación se procederá a hacer la implementación mediante SCSS y HTML de manera nativa.

ii. ngx-Socket-io y Socket.io

En este apartado se tratarán las dos librerías usadas para la implementación de la comunicación mediante Web Sockets. Siendo ngx-Socket-io la librería usada en la parte frontal y Socket.io en la parte trasera (Gwartney, 2021).

En primer lugar, se explicará Socket.io dado que es la librería principal, y de donde parten librerías como ngx-Socket-io. Socket.io es una librería open source con una amplia comunidad que nos permite mediante el uso de Web Sockets la construcción de aplicaciones con conexión persistente entre el cliente y el servidor. Por esta razón es necesario el uso de librerías para esta conexión en ambas parte, frontal y trasera. Cabe destacar que Socket.io no es una implementación de Web Sockets, simplemente se nutre de estos para realizar comunicaciones añadiendo metadatos adicionales a cada paquete.

Aunque su desarrollo en principio fue pensado para su uso en servidores que requirieran de NodeJs, se ha trasladado también a lenguajes como Java, Python y Go. En vista de la estrecha relación entre NodeJs y NestJs, no resulta extraño que esta librería se encuentre integrada por completo en esta última tecnología. Todo ello haciendo uso de su instrucción **`@WebSocketGateway`** dotando de capacidades de servidor de Web Sockets una puerta de enlace común. Es por esta razón que esta librería fuera la librería elegida para su inclusión en la parte trasera.

Con respecto a la parte frontal de la aplicación se hizo uso de la librería ngx-Socket-io. ngx-Socket-io es un módulo que implementa las funcionalidades del lado del cliente de Socket.io pero específica para Angular. Una de sus ventajas principales, añadido a su integración específica para Angular es la posibilidad de usar múltiples Sockets y endpoints proporcionando únicamente una configuración que incluya la dirección del recurso y las opciones pertinentes.

iii. Multer

El uso de esta librería reside en el intercambio de ficheros de diversos formatos entre el lado del cliente y el servidor. Para manejar la subida de archivos y su posterior guardado en base de datos se requiere de un middleware que intercepte los archivos que están siendo subidos. La librería escogida es Multer, que es un paquete creado para Express, pero integrado completamente en NestJs, solamente necesitando la instalación de sus tipos expresamente para tratar de mejor manera los ficheros que se envían. Una vez obtenido el fichero se obtiene su buffer de información para su posterior guardado en la base de datos.

iv. ngx-dropzone

Continuando con lo expuesto en el anterior punto, la librería ngx-dropzone proporciona en el lado del cliente un componente específico para Angular, ligero y altamente modificable enfocado en la subida de archivos. Como su nombre indica, proporciona una sección en la que arrastrar archivos o seleccionar archivos para su posterior subida y envío al servidor. Este componente permite mediante su configuración tratar el envío de múltiples archivos en un pack, limitar el tamaño de los archivos, indicar los formatos de archivos aceptados por el contenedor y ofrece una previsualización del contenido antes de su subida, permitiendo la eliminación inclusive de los archivos de manera individual.

v. ctrl/ngx-emoji-mart

Una de las funcionalidades implementadas en la aplicación fue el uso de emoticonos, para llevar a cabo la implementación de esta funcionalidad se hizo uso de la librería ctrl/ngx-emoji-mart, esta librería proporciona una ventana de diálogo desplegable mediante un botón con forma de emoticono que permite al usuario escoger emoticonos de una larga lista como podemos ver en la Ilustración 5. Esta librería tiene disponible el acceso a los sets de emoticonos de Apple que es el set por defecto, de Google, de Facebook y de Twitter, con distintas calidades.

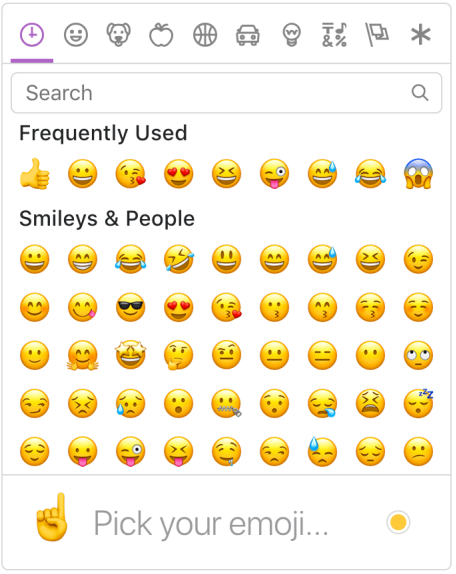


Ilustración 5 Diálogo de elección de emoticonos

También permite la implementación de emoticonos nativos en sets proporcionados al módulo, esta última funcionalidad no fue necesario su uso considerando que el set proporcionado por Apple tenía un diseño suficientemente agradable para los usuarios. Añadido a esto proporciona una variable de configuración responsable de implementar el modo oscuro de manera automática.

V. Herramientas

En este apartado se tratarán las herramientas usadas tanto para el modelado, el diseño y la codificación del proyecto. Las herramientas son las siguientes: de cara a la codificación, WebStorm, respecto a los navegadores, Firefox Developer Edition fue el escogido. Para el diseño de los iconos y fondos de la aplicación se han utilizado dos programas de edición gráfica diferentes. Para el diseño, el entintado y el color se ha utilizado el software de Procreate mientras que para redimensionar y escalar las imágenes y su exportación en máxima calidad se ha dado uso a Photoshop. En referencia a la base de datos se ha hecho uso de MongoDB Compass, para el control de versiones Git y Github, y finalmente para el despliegue en algunos puntos del desarrollo, se usó Docker Desktop.

i. WebStorm

WebStorm es un IDE profesional enfocado en JavaScript con una sencilla interfaz y gran cantidad de herramientas disponibles. A su vez es compatible con una amplia gama de tecnologías modernas que guardan relación con el lenguaje de programación JavaScript, el editor de estilos CSS y el lenguaje de marcado HTML. Aunque la versión de prueba solo alcanza los 30 días de uso, permite a los usuarios con condición de estudiantes universitarios su uso durante el periodo correspondiente al grado. Algunas alternativas que se valoraron para la realización de esta función fueron las siguientes:

- Visual Studio Code: Es un editor de código gratuito optimizado para el desarrollo de aplicaciones web y cloud. Es bastante sencillo de utilizar pero tiene un defecto que reside en su ligereza, y que para equipararse a la gran cantidad de herramientas que proporciona WebStorm requiere de configuraciones adicionales y de la instalación de plugins que vienen configurados ya en WebStorm. Aunque una vez todo configurado puede tener la misma potencia que WebStorm.

- Atom: Es un editor de código gratuito, también optimizado para el desarrollo de aplicaciones web. Es más simple que Visual Code y tiene menor soporte que este, pero a su vez es muy ligero.Es por esto por lo que el entorno que se escogió para el desarrollo de la aplicación fue WebStorm, por su sencillez de uso sin previas configuraciones, también debido a que en las condiciones del proyecto las tres opciones disponibles eran de carácter “gratuito” siendo en el caso de estas últimas gratuitas sin cumplir una serie de requisitos. Finalmente, el aspecto principal por el que se prefirió este IDE frente a los editores de código fue que proporcionaba una mejor gestión de los ficheros y una apariencia visual más agradable como se puede ver en la Ilustración 6.

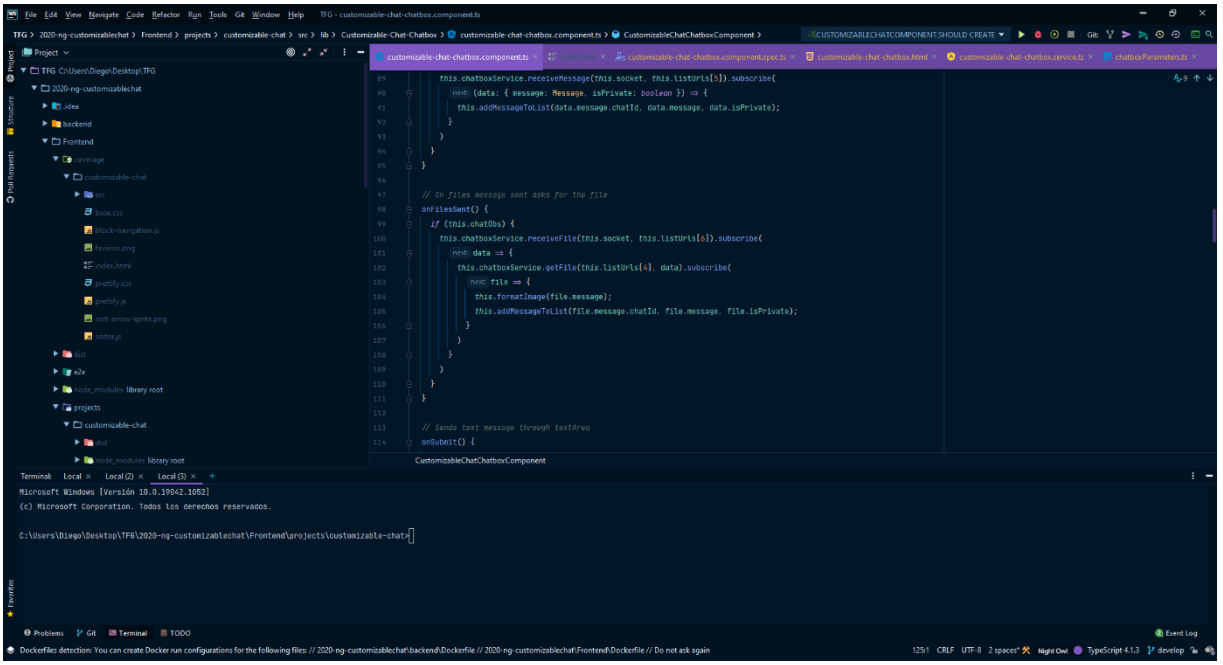


Ilustración 6 Interfaz gráfica WebStorm por IntelliJ

ii. Firefox Developer Edition

El navegador que se escogió para el desarrollo de la aplicación fue Firefox Developer Edition. Las razones de esta elección frente a otros navegadores como Google Chrome fue su rendimiento desde la actualización Quantum. En contraste con el incremento de rendimiento de Firefox sobre Google Chrome se llegó a la conclusión de que la implementación de las herramientas de desarrollo en Google Chrome es más directa que la presente en Firefox. Pero este no es el caso de su versión para el desarrollador, que se encuentra 12 meses por delante en todo momento de su versión por defecto.

Además, Firefox Developer Edition ofrece la inclusión de herramientas de desarrollo de manera nativa en el propio navegador. Este no es el caso de Google Chrome, que requiere de la instalación de plugins para poder adquirir estas características. Herramientas como Eyedropper que permitió la manipulación de colores en el diseño, y un mayor control de las animaciones en su desarrollo, visualizado en la Ilustración 7, supusieron la elección de Firefox Developer Edition frente a Google Chrome.

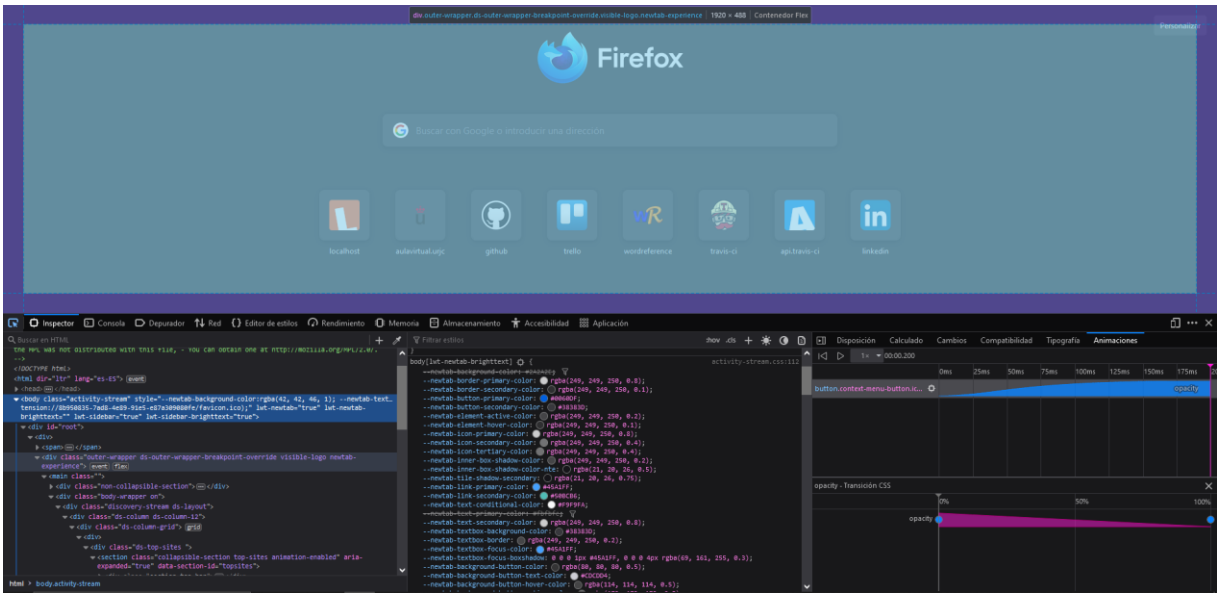


Ilustración 7 Firefox Developer Edition

iii. Procreate

Procreate es un editor gráfico para diseño digital con una interfaz sencilla, pero a la vez muy versátil. Aunque este programa está destinado al dibujo en digital desde su lanzamiento en 2011 ha ido innovando su aplicación para convertirse en una opción más frente a otras competidoras.

Creada para IOS, Procreate ofrece al usuario la sensación física del diseño tradicional, pero manteniendo las ventajas de una plataforma digital. Su precio es bastante asequible y su licencia es de por vida, lo que hace que usuarios la prefieran frente a otros programas con una licencia mensual.

Se ha decidido utilizar este programa por su calidad en los pinceles que incorpora por defecto, así como por la facilidad de importar pinceles de terceros e incluso de crear los suyos propios, funcionalidad visualizada en la Ilustración 8. La posibilidad de crear múltiples capas y editarlas individualmente es también una función muy práctica que ayuda a realizar un trabajo final de gran calidad en los diseños. Las herramientas usadas en esta aplicación han sido las siguientes:

- Pinceles de Boceto: con estos se realiza el primer boceto rápido, realizando un dibujo de la idea principal sin mucho detalle

- Pinceles de entintado: el segundo paso es utilizar este tipo de pinceles donde se pasa a limpio el boceto principal y se da detalle a nuestro diseño.

- Bote de pintura: es una herramienta rápida con la que se aplica color al dibujo en una capa específica e incluso en un parte concreta del dibujo.

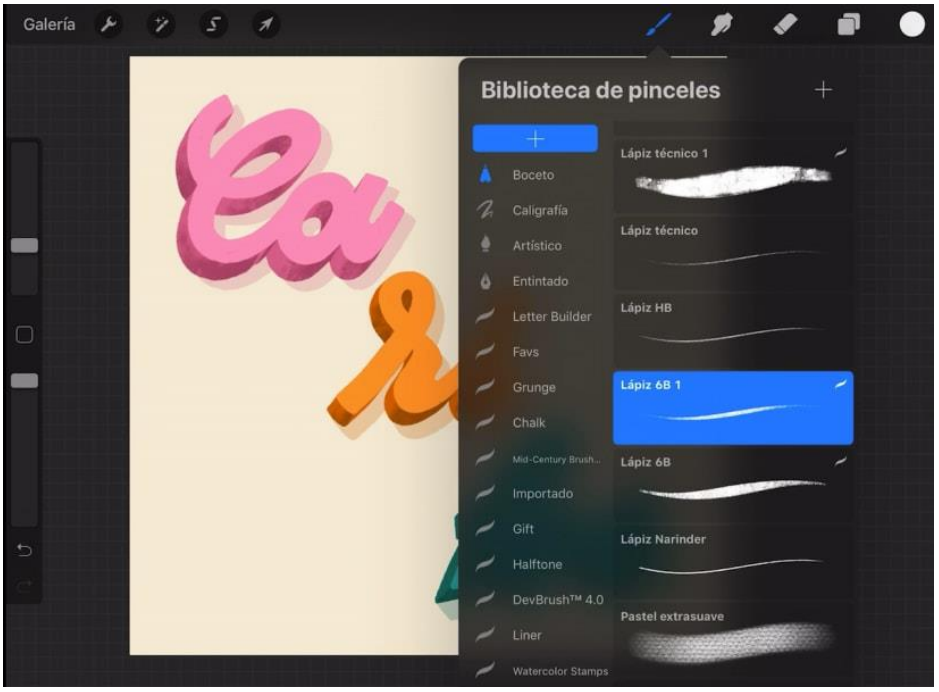


Ilustración 8 Procreate

Si bien Procreate es un software completo con muchas herramientas efectivas tiene algunas limitaciones frente a otros, como por ejemplo la pérdida de calidad cuando se redimensiona el tamaño de una imagen. Es por eso por lo que se utiliza Photoshop para solventar este problema.

iv. Photoshop

Photoshop es un software de edición de imágenes pertenecientes al paquete de herramientas de Adobe. Aunque su principal función fue el retoque de fotografías y gráficos desde 1990, los usuarios le han dado múltiples usos a esta, desde el diseño, el dibujo en digital e incluso la edición de vídeos. Esta herramienta es compatible con Windows y MacOS, aunque se puede llegar a usar en Linux mediante herramientas de terceros, Photoshop tiene una interfaz bastante más compleja que Procreate, pero ofrece una mayor cantidad de opciones al usuario como se puede ver en la Ilustración 9. Su precio no es excesivamente caro y su licencia es mensual, ofreciendo ventajas especiales a los estudiantes.

Como se mencionó anteriormente, la principal razón de uso de esta herramienta ha sido el redimensionamiento del tamaño de las imágenes para de esta manera, escalar los diseños realizados sin que estos pierdan calidad y puedan ser utilizados a lo largo de la aplicación. Para la exportación de las imágenes se ha utilizado este mismo programa pues

ofrece calidades superiores frente a otras opciones en el mercado.

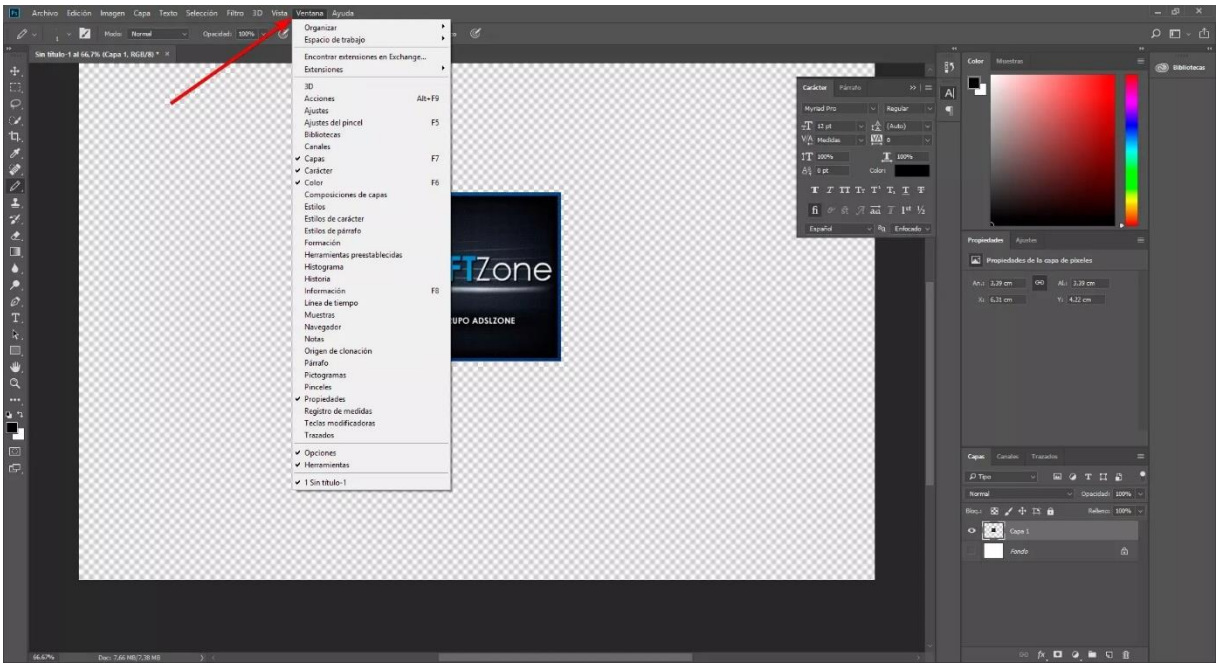


Ilustración 9 Photoshop

v. MongoDB Compass

MongoDB Compass es una herramienta con interfaz gráfica dedicada al manejo de una base de datos de MongoDB. Esta herramienta tiene como funcionalidades incluidas las mismas que MongoDB Shell, o lo que es lo mismo una herramienta de MongoDB para tratar con bases de datos de esta tecnología por la línea de comandos. Añadido a esto permite realizar análisis de los datos que se encuentran en la base de datos, proporcionando a su vez gráficos de las propiedades de cada colección como se puede ver en la Ilustración 10.

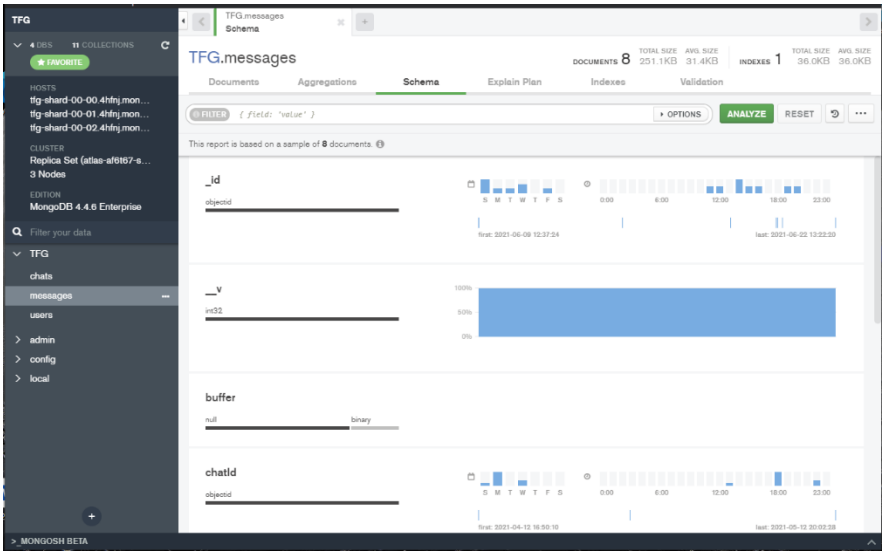


Ilustración 10 MongoDB Compass

Por las razones mencionadas anteriormente se prefirió hacer uso de la versión con interfaz gráfica ya que permite una mejor visualización y asociación de los datos. En comparación con su versión en línea de comandos se puede añadir que, al ya poder realizar las consultas necesarias a través del servidor vía comandos, no supone un aporte mayor la utilización de MongoDB Shell.

vi. Git y Github

Git es una herramienta enfocada en el control de versiones con un sistema distribuido, optimizada de cara al mantenimiento de versiones de aplicaciones cuando tienen un gran número de archivos de código fuente. A su vez, una de las principales razones por las que se popularizó esta herramienta fue por poder llevar un registro de los cambios realizados sobre unos mismos archivos por las distintas personas involucradas en el proceso.

Github es un portal que se creó para alojar el código de las aplicaciones subidas por los desarrolladores. Su creación, como su nombre indica, reside en la herramienta de control de versiones mencionada anteriormente, Git. Así pues, Github es una herramienta que sirve para gestionar código que utilice el sistema de Git. El repositorio usado para alojar el código se puede visualizar en la Ilustración 11. Git permite la utilización de herramientas como el "branching" que permite mantener varias ramas de desarrollo con distinto estado del repositorio, o el "merging" que ofrece la posibilidad de hacer combinación de ramas.

Por consiguiente, el aspecto clave que hizo que el proyecto se decantará por el uso de Git fue su posibilidad de actualizar el estado del código sin conexión. Esta decisión permitió realizar confirmaciones de la actualización del código sin llegar a actualizar el estado del repositorio externo. El repositorio de este proyecto alojado

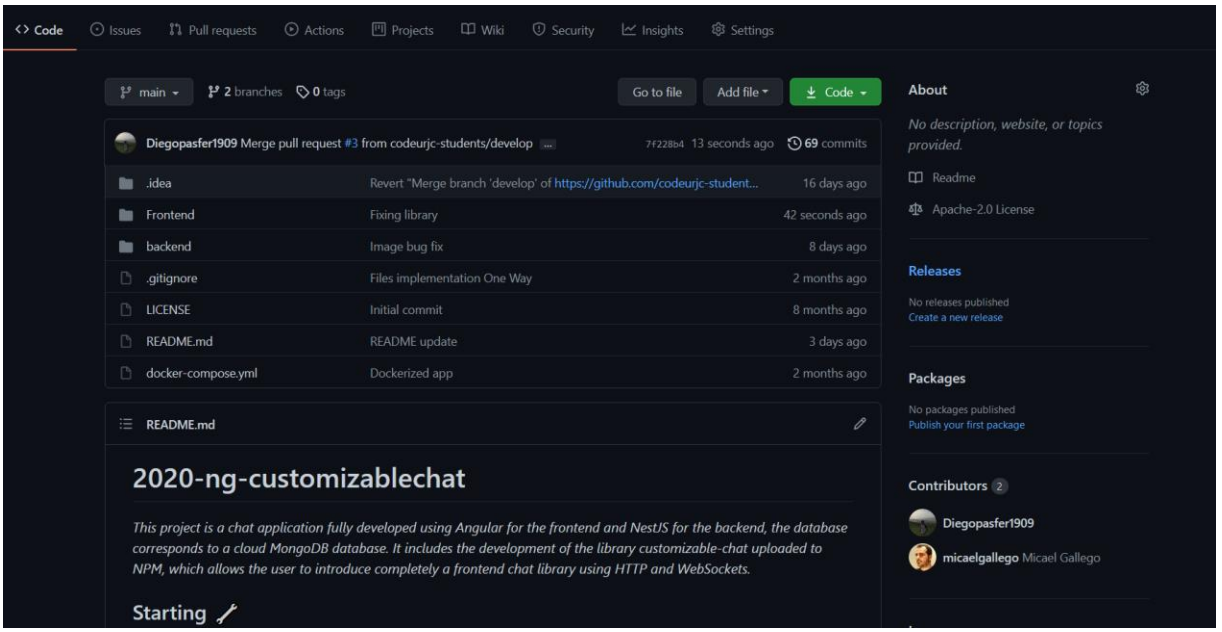


Ilustración 11 Repositorio Github del proyecto https://github.com/codeurjc-students/2020-ng-customizablechat

vii. Docker Desktop

Docker Desktop es una herramienta que permite de una manera sencilla ejecutar, compilar, depurar, y probar aplicaciones que estén dockerizadas. O lo que es lo mismo, permite realizar las acciones mencionadas anteriormente haciendo uso de contenedores que almacenen esas aplicaciones, o que incluso lleguen a contener sistemas operativos completos.

Esta herramienta permite la manipulación mediante una interfaz visual, de los distintos contenedores que se encuentren usándose por el ordenador que los ejecuta como se puede ver en la Ilustración 12. A su vez Docker Desktop presenta otras características muy relevantes como son: ciclos rápidos de prueba de edición, notificaciones de cambio de archivo, soporte de red empresarial integrado y flexibilidad de cara al uso de proxies y Redes Virtuales Privadas.

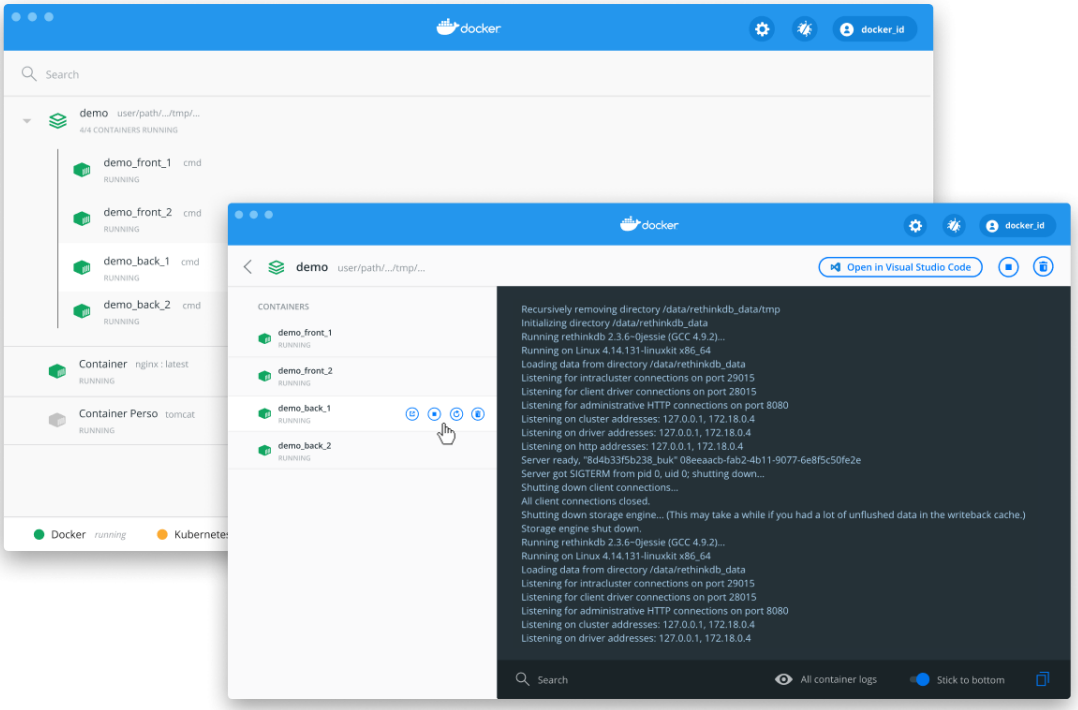


Ilustración 12 Docker Desktop

En el contexto de este proyecto al hacerse uso del sistema operativo Windows, el uso de la aplicación Docker Desktop es de manera nativa. La razón de esto reside en el hecho de que Docker y sus contenedores se ejecutan dentro de una máquina Linux intermedia creada solo para la ejecución de estos contenedores. Docker Desktop se aprovecha de la virtualización nativa disponible en Windows que es Microsoft Hyper-V.

Capítulo 3 Descripción Informática

En este apartado se describirá con detalle el proyecto realizado atendiendo a los requisitos establecidos para el proyecto, su arquitectura, el diseño e implementación, la interfaz de usuario y las pruebas.

I. Requisitos

El proceso de trabajo se ha desarrollado por etapas, por esta razón, tiene sentido dividir la especificación de los requisitos en las etapas principales, que los momentos de especificación también se ceñían a esas etapas. Se ha usado como notación RF para indicar que es un requisito funcional, RNF para indicar que es un requisito no funcional (Requeridos Blog, 2018).

i. Etapa 1 Desarrollo de una aplicación

Identificador	Nombre	Descripción
RF-001	Partes de la aplicación	La aplicación una vez abierta deberá contar con 3 apartados distintos: un menú de opciones, un listado de chats y una caja para el chat desplegado
RF-002	Inicio de sesión	El sistema tendrá una pantalla dedicada al inicio de sesión de la aplicación y será completado mediante un formulario.
RF-003	Registro	La aplicación tendrá una pantalla dedicada al registro de nuevos usuarios, y será completado mediante un formulario.
RNF-001	Inicio de sesión y registro	La aplicación mostrará el inicio de sesión y el registro en una misma pantalla mediante una animación de cambio de los formularios.
RF-004	Botón mostrar contraseña	La aplicación tendrá un botón en el formulario de inicio de sesión que mostrará la contraseña escrita al usuario, por defecto se encontrará tapada por puntos.
RF-005	Barra opciones	La aplicación tendrá un apartado dedicado a las opciones y contendrá los

		siguientes botones: botón de acceso al perfil del usuario, botón de ajustes, botón para crear chat privado, botón para crear chat grupal, botón de búsqueda, botón de favoritos, botón para acceder a la galería de imágenes, botón de cierre de sesión
RF-006	Barra chats	La aplicación en la pantalla principal mostrará un apartado dedicado a listar los chats grupales, otro a listar los chats privados y finalmente todos los chats en otro apartado.
RNF-002	Botones cambio de tipo de chat	La aplicación proporcionará al usuario el cambio de los tipos de chat privado, grupal y todos mediante botones que realicen una animación de despliegue a la derecha.
RF-007	Botón para cada chat	La aplicación dispondrá de un botón por cada chat para desplegar este como principal.
RF-008	Despliegue chat	La aplicación mostrará un apartado en la pantalla principal al ser desplegado un chat mostrando en la parte superior los datos del chat, en la parte media los mensajes cargados de la base de datos y en la inferior una entrada de texto.
RF-009	Envío de mensajes	La aplicación contará con un botón para enviar el mensaje escrito en la entrada de texto inferior del chat, una vez enviado el mensaje deberá aparecer en el listado de mensajes del chat abierto.
RNF-003	Posición mensajes	La aplicación mostrará en el lado izquierdo de la pantalla de chat los mensajes enviados por los participantes del chat y en el derecho y de otro color los enviados por el usuario de la aplicación.
RNF-004	Colores	La aplicación usará los colores #ffffff, #00acee, #ace7ff y #ffffff usando el sistema hexadecimal para el fondo, los

		bordes, los fondos de las secciones del chat y la letra respectivamente.
RF-010	Envío mensajes servidor	El servidor al recibir un mensaje de texto de un chat redistribuirá el mensajes a todos los participantes del chat independiente del tipo de este chat, privado o grupal, guardando el mensaje en la base de datos.
RF-011	Recepción de chats	El servidor proporcionará al usuario los chats que correspondan al usuario una vez detecte su conexión
RF-012	Recepción mensajes	El servidor proporcionará al usuario los 10 últimos mensajes del chat seleccionado por el usuario.

Tabla 1Etapa 1 Desarrollo de aplicación web

ii. Etapa 2 Mejora visual

Identificador	**Nombre**	**Descripción**
RNF-001	Definición de temas	La aplicación contará con dos tema, tema claro y tema oscuro, el primero usará los colores definidos en el RNF-004 de la Etapa 1 y el tema oscuro usará los colores #6e8ee6 para los bordes, #1a204f para los fondos y #000000 para los textos
RNF-002	Fondos	La aplicación para indicar que el chat no está desplegado mostrará un fondo del color del tema propio, mostrando un fondo distinto en caso de estar desplegado
RF-001	Cambio de modos	La aplicación cambiará el modo de diurno a nocturno y viceversa con cada clic del usuario en el botón de cambio de modo, cambiando el tema de toda la aplicación. El servidor guardará la configuración en la base de datos.
RNF-003	Cambio de iconos	La aplicación proporcionará un diseño único a los iconos de los botones de toda la aplicación.

RNF-004	Cambio de archivos de estilos	La aplicación contará con archivos SCSS para la estilización del HTML, sustituyendo los archivos CSS por SCSS

Tabla 2 Etapa 2 Mejora visual de la aplicación

Los fondos mencionados en los requisitos se explicarán en detalle y mostrarán más adelante.

iii. Etapa 3 Introducción de nuevas funcionalidades en la aplicación

Identificador	Nombre	Descripción
RF-001	Envío de archivos	La aplicación permitirá el envío de archivos de cualquier formato y enviará un mensaje a los usuarios conectados del mensaje para que se descargue. El servidor guardará ese archivo en base de datos como un mensaje.
RF-002	Envío de links	La aplicación permitirá al usuario enviar mensajes con links que al emitir un clic por parte de un usuario le redireccionará al link.
RF-003	Envío de emoticonos	La aplicación proporcionará al usuario un menú para elegir emoticonos de la lista de Apple y permitirá su envío en cualquier mensaje de texto.
RF-004	Reestructuración chats	La aplicación dispondrá ahora de chats privados y grupales eliminando el apartado dedicado a todos los chats.
RF-005	Ampliación imágenes	La aplicación permitirá al usuario clicar en las imágenes enviadas en el chat y se ampliarán a pantalla completa.
RF-006	Apertura de archivos	La aplicación proporcionará al usuario la posibilidad de clicar los archivos enviados por un chat y que se abran en una pestaña nueva para su descarga o apertura en caso de ser un archivo o visualización en caso de los vídeos.

Tabla 3 Introducción de nuevas funcionalidades en la aplicación

iv. Etapa 4 Creación de la librería

En este apartado se tendrán en cuenta solo los requisitos que corresponden a la librería. Para indicar a que componente hace referencia el requisito se indicará en el identificador con C1 si es para el componente de listado de chats y C2 si es para el componente del chat desplegado. En caso de no corresponder a ninguno de los dos y corresponder a la librería en sí, no lo acompañará ningún identificador adicional.

Identificador	Nombre	Descripción
RF-001	Componentes de la librería	La librería contará con dos componentes distintos, el primero relacionado con el listado de chats y el segundo relacionado con un chat desplegado.
RF-002/C1	Botones cambio de chat	El componente proporcionará dos botones para cambiar entre los chats privados y los chats grupales.
RNF-001/C1	Partes de un chat	El componente por cada chat mostrará el icono de perfil del chat, el nombre del chat, o del participante en caso de ser un chat privado, el último mensaje enviado, la cantidad de mensajes sin visualizar.
RF-003/C2	Envío de mensajes de texto	El componente permitirá al usuario mediante la introducción de texto el envío de mensajes y/o links. El componente enviará ese mensaje al servidor proporcionado.
RF-004/C2	Recepción de mensajes de texto	El componente reaccionará a los mensajes recibidos en el chat actual introduciéndolo en el listado de mensajes del chat actual independientemente del tipo del mensaje.
RF-005/C2	Envío de archivos	El componente permitirá al usuario enviar archivos mediante el desplegable dedicado a adjuntar archivos confirmando el envío mediante un botón de enviar.
RF-006/C2	Introducción de emoticonos	El componente permitirá al usuario introducir emoticonos en la entrada de texto mediante un diálogo de selección.

RF-007/C2	Ampliación imágenes	El componente permitirá al usuario clicar en las imágenes enviadas en el chat y se ampliarán a pantalla completa
RF-008/C2	Apertura de archivos	El componente proporcionará al usuario la posibilidad de clicar los archivos enviados por un chat y que se abran en una pestaña nueva para su descarga o apertura en caso de ser un archivo o visualización en caso de los vídeos.

Tabla 4 Creación de una librería

v. Etapa 5 Introducción de la librería en la aplicación

Identificador	Nombre	Descripción
RNF-001	Introducción de los componentes de la librería en la aplicación	La aplicación sustituirá los componentes y apartados de la aplicación correspondiente al listado de mensajes y al chat desplegado por el uso de los componentes de la librería.

Tabla 5 Introducción de la librería en la aplicación

VI. Arquitectura y Análisis

En este apartado se describirán los aspectos de alto nivel de la aplicación y para esto se hará uso de diversos diagramas. Se introducirá el apartado con el Diagrama 1 que sirve de introducción a la arquitectura principal de la aplicación, siendo una arquitectura SPA y relaciona el cliente en Angular con el servidor en Nestjs; este servidor contiene un servidor para peticiones de tipo HTTP y un servidor adicional dedicado exclusivamente a los Web Sockets.

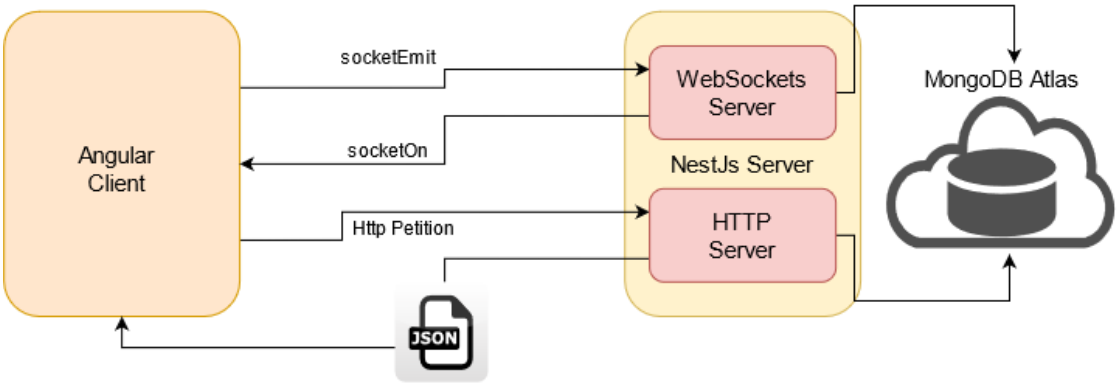


Diagrama 1 Arquitectura SPA introductoria de la aplicación

El Diagrama 2 es un diagrama de clases y vistas de la parte frontal de la aplicación, debido al patrón MVC que implementa Angular podemos representar la vista que se relaciona con cada uno de los componentes. Estos se identifican por su nombre acompañado de la palabra controlador, esto se debe a que los controladores en Angular son los propios archivos de TypeScript llamados componentes. Cabe destacar que el Diagrama 2 pertenece a la etapa previa a la construcción de la librería. Posteriormente visualizaremos este diagrama, pero haciendo uso de la librería para poder tratar los cambios.

El componente inicial es el controlador de Login que mediante el servicio de Login proporciona al controlador de Frame los datos del inicio de sesión o registro. Una vez enviada la información el componte hace una redirección al componente de Frame. Este componente aporta a los siguientes componentes la estructura visual y los datos del usuario obtenidos en el inicio de sesión o registro. Esto se hace de manera directa teniendo en cuenta que el componente Frame es el componente "padre" de los componentes Options, Chat List y Chatbox. Esta construcción en distintos componentes de lo que aparentemente podría ser una misma vista se debe a la escisión de los componentes para su posterior encapsulación en la librería, y a su vez, para separar información no requerida por todos los apartados. En cualquier momento de la aplicación si el usuario clica el botón de desconexión, el componente de Options se comunicaría con el componente de Login mediante el servicio de Login para indicarle que quiere cerrar su sesión.

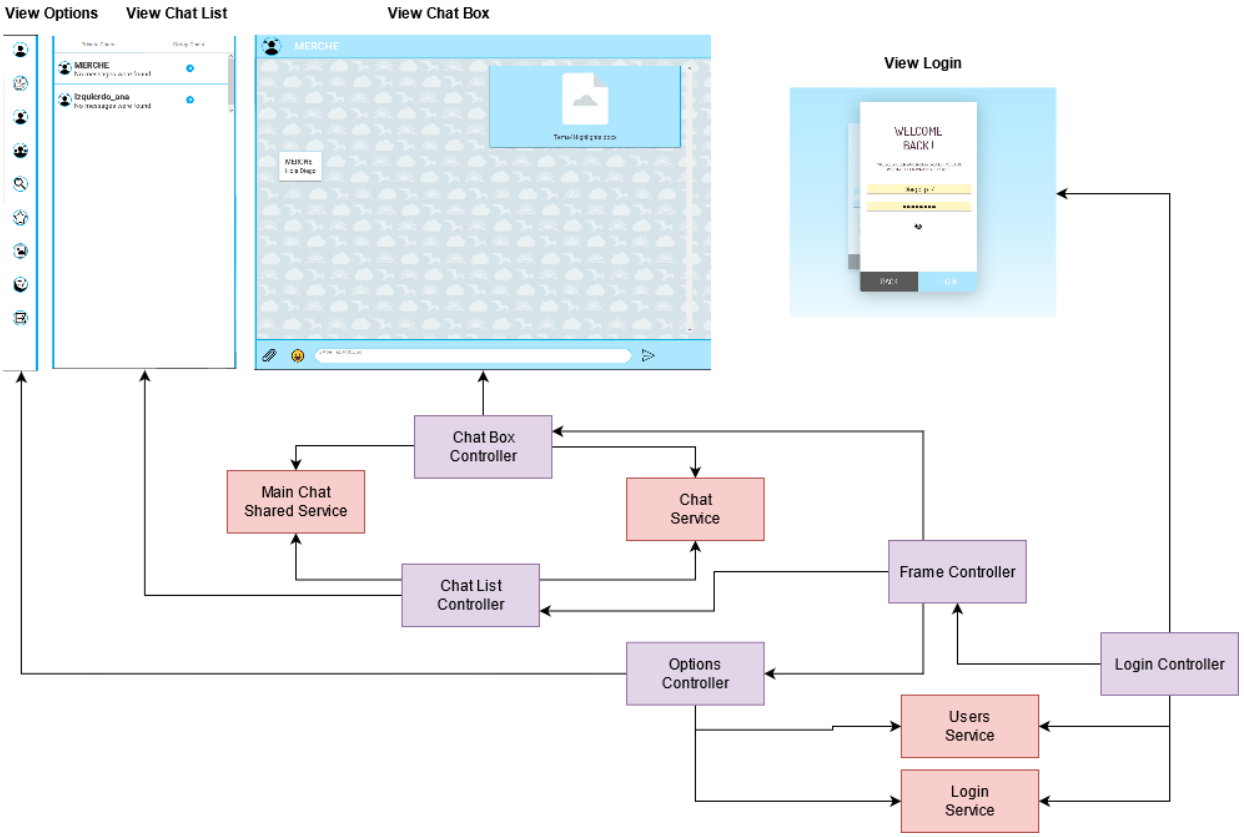


Diagrama 2 Diagrama de clases y vistas sin uso de la librería

A su vez el componente de Options se comunica con el componente de Login en caso de realizar por parte del usuario un cambio o bien, del modo en el que quiere visualizar la aplicación, diurno o nocturno, o bien, del perfil del usuario, es decir, de su imagen de perfil o de su descripción. Estos cambios mencionados se hacen a través del servicio Users.

En cuanto a los componentes Chatbox y ChatList hacen uso del servicio Main Chat Shared, esto se debe a que como componentes "hermanos" la forma de compartir información es a través de un servicio, como también ocurre con los componentes que no tienen relación. La información que se transmite entre estos componentes es la selección de un chat por parte del usuario, esta comunicación mediante el servicio permite al componente ChatList enviar a Chatbox el chat seleccionado por el usuario. Una vez más esta comunicación se podría haber realizado sustituyendo la relación de los dos componentes por una relación "padre-hijo", con esta última situación el componente ChatList contendría al componente Chatbox incluyendo su vista, incrustada en el primer componente. Esta última toma de decisión en cuanto al diseño se debió en gran parte a la futura implementación de una librería en función de estos componentes, haciendo a ambos cuanto más independientes se pudiera, salvando las relaciones claras entre uno y otro.

El Diagrama 3 corresponde a la arquitectura del servidor en NestJs de la aplicación. Este diagrama permite una visualización de los controladores, el Gateway y los servicios que se crearon para permitir la comunicación con la parte frontal de la aplicación.

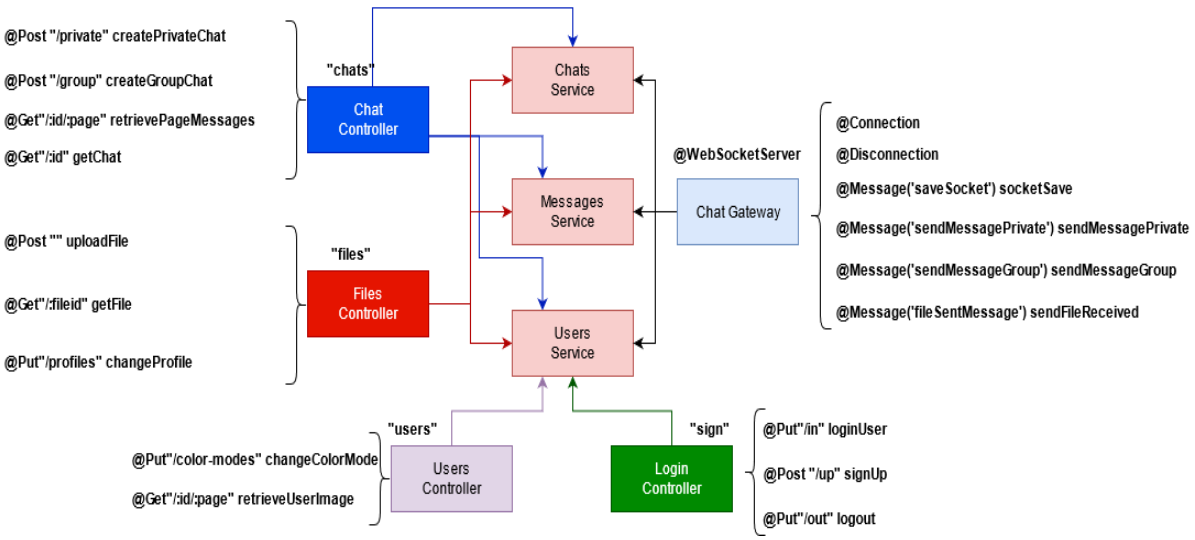


Diagrama 3 Diagrama del servidor en NestJs

En este diagrama se reflejan todas las peticiones que se pueden realizar en el al servidor, estas peticiones, aunque corresponden a la etapa previa a la introducción de la librería no sufrieron cambios.

En primer lugar, destaca la orientación del diagrama, enfocado principalmente a los servicios, esto se debe a que son los que se comunican de manera directa con la base de datos contactando con las colecciones correspondientes a cada Schema definido en el servicio correspondiente.

Cabe destacar el servicio más importante y eje central de la aplicación el servicio Users, del que hacen uso todos los controladores de la aplicación, incluido el Chat Gateway, la razón de esto es que en todos los controladores hay al menos una petición que requiere más información de los usuarios, en el caso del controlador de chats para poder crear grupos, crear chats privados, añadir contactos a los distintos grupos posteriormente a su creación, todas estas peticiones hacen uso del servicio Users.

Atendiendo al Gateway, este requiere del uso del servicio de usuarios para añadir la actividad del usuario, en otras palabras que se ha conectado, guardando en base de datos su actividad a verdadero y el identificador de su WebSocket para enviar mensajes directos a la persona, todo esto mediante los Web Sockets, o borrando este identificador y cambiando su actividad a falsa en la desconexión, producida o bien por una desconexión brusca cerrando la pestaña de la aplicación, y reaccionando a ello mediante los Web Sockets y sus métodos para reconocer la desconexión o bien de la manera habitual mediante la llamada al cierre de sesión. También requiere de la actividad de los usuarios para poder mandarle el mensaje por Web Sockets si se encuentra activo, usando el identificador obtenido de la consulta a la colección de usuarios.

El controlador de usuarios como su nombre indica, tiene trato con los usuarios y por ende se relaciona con la colección de usuarios. Por un lado, para notificar el cambio de tema de un usuario a la base de datos. Por otro lado, para obtener la imagen del usuario y poder mostrarla. En los chats privados la imagen es propia de este objeto y se encuentra en su colección. Esta situación cambia con los chats grupales, cuya imagen se encuentra ya integrada en la colección de chats.

De manera semejante, el controlador Files requiere del servicio de usuarios ya que modifica el perfil del usuario. En si la creación de este controlador reside en facilitar el mantenimiento de la aplicación reservando la modificación de cualquier archivo a este controlador, haciendo uso de la librería Multer mencionada anteriormente para el trato de los archivo.

Como se explicó previamente el cierre de sesión se produce de dos maneras distintas, la forma abrupta y la forma elegante con cierre de sesión explícito por parte del usuario. Esta segunda corresponde al controlador Login que se encarga del cierre de sesión por medio de la petición del usuario, también comprueba la existencia de un usuario con las claves correctas en el inicio de sesión y la corrección de los datos enviados para el registro de un nuevo usuario, siendo todas estas peticiones mencionadas anteriormente las relacionadas con el servicio Users.

En referencia al servicio Messages, se observa que tiene también una gran importancia en la aplicación. Esto es así, ya que los mensajes son la forma de comunicación principal de la aplicación al ser una aplicación de mensajería mediante conversaciones.

El controlador Files hace uso de este servicio de cara a obtener ficheros pedidos por los usuarios y la subida de archivos a los chats. Para comprender mejor el funcionamiento de la primera característica se tienen que tratar con más nivel de detalle los Web Sockets, el uso de esta herramienta para la comunicación en tiempo real de manera bidireccional es clave, esto no quiere decir que no tenga sus limitaciones. Sus limitaciones residen en el tamaño del dato que se quiere comunicar, siendo insignificante el envío de un mensaje de texto en comparación con el envío de un vídeo con alta calidad. Es por ello por lo que el envío de archivos se produce en dos pasos distintos.

Antes de seguir examinando este esquema se explicará esta secuencia de acciones en detalle. En el Diagrama 4 se puede visualizar el funcionamiento de esta comunicación. Este diagrama refleja la subida de un archivo desde la parte frontal, el posterior envío de ese archivo mediante el servicio Chat usando una petición HTTP de tipo @Post. Una vez esta petición contacta con el controlador Files envía un archivo a la base de datos para su guardado con el formato del Schema de Message, que posteriormente revisaremos en el

diagrama de entidades. Todo este proceso se realiza haciendo uso del servicio Messages. Una vez subido el archivo a la base de datos, la base de datos devuelve el mensaje que se guardó, después de pasar por el controlador de la parte trasera se devuelve a la parte frontal el identificador del mensaje. El cliente tras recibir este mensaje envía una petición al servidor mediante Web Sockets indicando al servidor de Web Sockets que tiene que notificar a los usuarios del chat.

Para obtener los datos del chat y de los usuarios a los que enviar el mensaje de que tiene un archivo nuevo se realiza una petición a la base de datos para obtener el chat en el que se hizo la petición, y de una manera más concreta los usuarios de este chat. Una vez recogidos los participantes del chat se realiza una petición adicional para obtener los Web Sockets concretos de los usuarios activos.

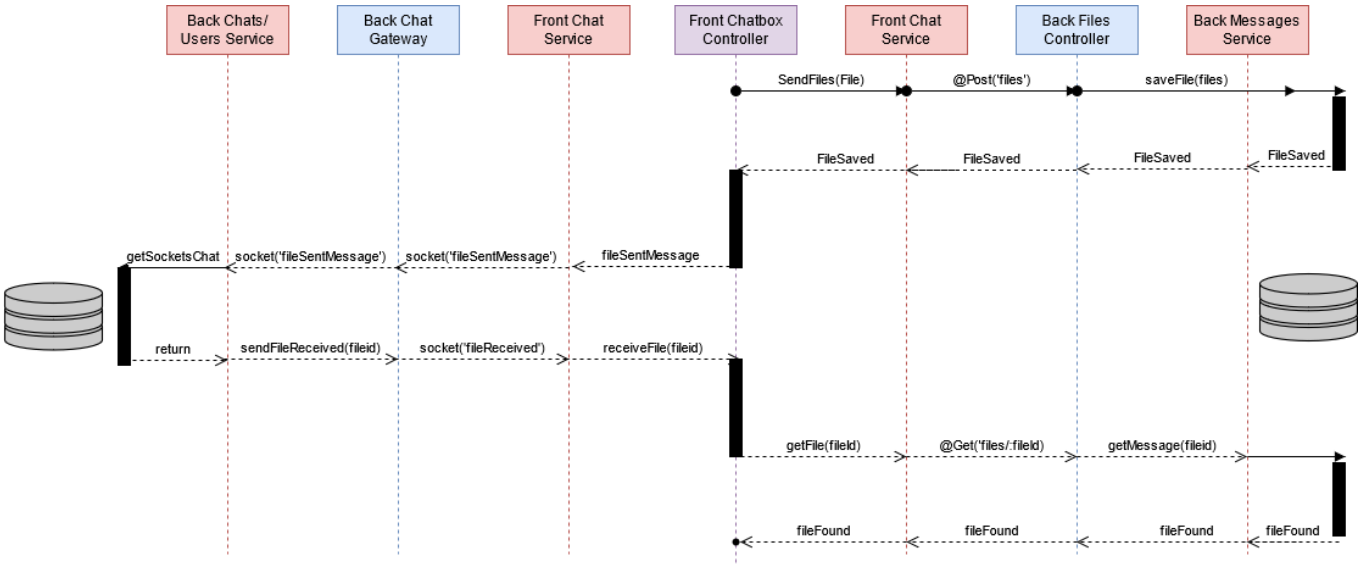


Diagrama 4 Diagrama de secuencia del envío de un archivo

Una solución de cara a evitar realizar una petición adicional y que se planteó fue el añadir un campo adicional a los chats tanto grupales como privados en el que se indicaran los Web Sockets de los usuarios conectados, esta solución propuesta evitaría el uso de peticiones adicionales tanto para el envío y comunicación de los mensajes como de los archivos. Se tomó la decisión de evitar este camino ya que podrían surgir dos problemas:

El primer problema se podría generar debido al inicio de sesión de un usuario participante en un chat en el que otro usuario estuviera enviando el mensaje, si el mensaje se hubiera enviado ya y se estuvieran revisando los usuarios a los que realizar el envío del mensaje podría ocurrir que una vez revisados iniciará sesión el usuario produciendo que no se le enviara este mensaje. Esta situación se vería reducida su posibilidad si la comprobación se realiza sobre el usuario justo en el momento del envío y no sobre el chat.

El segundo problema está relacionado con la escalabilidad, cada vez que un usuario quisiera cerrar su sesión, el servidor debería realizar un cierre de sesión en cada una de las estructuras que contuvieran a este usuario como activo, añadido a esto, en caso de que el usuario realizara un cierre de sesión inesperado mediante el cierre de la pestaña se tendría que hacer uso del identificador del WebSocket en vez del nombre de usuario para el cierre de sesión, ralentizando este proceso. Este problema se acrecentaría a medida que un usuario participara en una mayor cantidad de chats.

Con respecto al esquema general de peticiones y en referencia al servicio Messages, en su interacción con el controlador Chat encontramos que este realiza consultas de los mensajes de un chat que se envíe de la parte frontal, de esta manera se pueden rescatar los últimos diez mensajes y rescatar en sentido inverso al curso del tiempo mensajes más antiguos, es decir la petición mediante paginación obtiene los diez últimos resultados de ese chat, en caso de existir más mensajes, en una siguiente petición, habiéndose incrementado la página, obtendría los siguientes 10 mensajes más antiguos. Esta petición es muy frecuente debido a que cada vez que se abre un chat por primera vez en una sesión se rescatan los últimos diez mensajes.

El último elemento relacionado con el servicio Messages es el Gateway Chats, este Gateway al ser el epicentro de envío y recepción de mensajes basa prácticamente su implementación en esto. Es por ello por lo que las peticiones de envío de mensajes tanto privados como grupales se realizan desde este Gateway, todo mediante el uso de los Web Sockets. Estas peticiones usan el servicio Messages para el guardado de estos mensajes en la base de datos. Añadido a esto en este Gateway se recibe la petición vía WebSocket que refleja que un mensaje de tipo archivo se ha guardado con éxito en la base de datos y se tiene que comunicar al resto de usuarios que participen en el chat.

Y con el Gateway mencionado en el párrafo anterior se terminan las interacciones con el servicio Messages. A continuación, se explicará el último servicio encontrado en el servidor, el servicio Chats. Este servicio de igual modo que el servicio Messages tiene interacciones con los controladores de Chat, de Files y finalmente interacciona con el Gateway Chat.

Por lo que se refiere al controlador Chat, tiene la mayor interacción con el servicio Chats puesto que hace uso de este servicio tanto para la creación de nuevos chats grupales como de chats privados. Además, añade contactos a los grupos de manera posterior a su creación aumentando la lista de participantes y recupera los chats cuando se hace una petición explícita desde la parte frontal.

El segundo con más interacción con este servicio es el Gateway Chat, las peticiones de este gateway en su totalidad corresponden con la obtención de los participantes de un chat tanto privado como público para el envío de mensajes privados, públicos o para comunicar a los usuarios participantes que se les ha mandado un archivo al chat que tienen que descargar mediante el formato explicado en el anterior diagrama. De este diagrama esta última interacción corresponde con la recepción del mensaje vía Web Sockets “fileSentMessage”.

El último controlador involucrado con este servicio es el controlador Files, este tiene solo una interacción con el servicio Chats, esta interacción se debe a enviar en el formato completo a qué tipo de chat corresponde el archivo que se tiene que descargar. Esta situación facilita que desde el “frontend” se manipule mejor a que lista de chats corresponde, facilitando la búsqueda del chat al que corresponde, solo necesitando el identificador del chat que ya viene contenido en el mensaje que se envía.

Otro rasgo de especial mención sobre el diagrama es el nombre indicativo que es la raíz de la dirección de las peticiones, siendo en todos los controladores la primera palabra del controlador. Por ejemplo, en el caso del controlador Chats la raíz de las peticiones que recibe este controlador es “chats”, esto así en todos los casos menos en el controlador Login que su identificador es “sign” y en el Gateway que solo recibe peticiones de Web Sockets.

En el Diagrama 5 se tratarán las entidades y sus relaciones para con la aplicación.

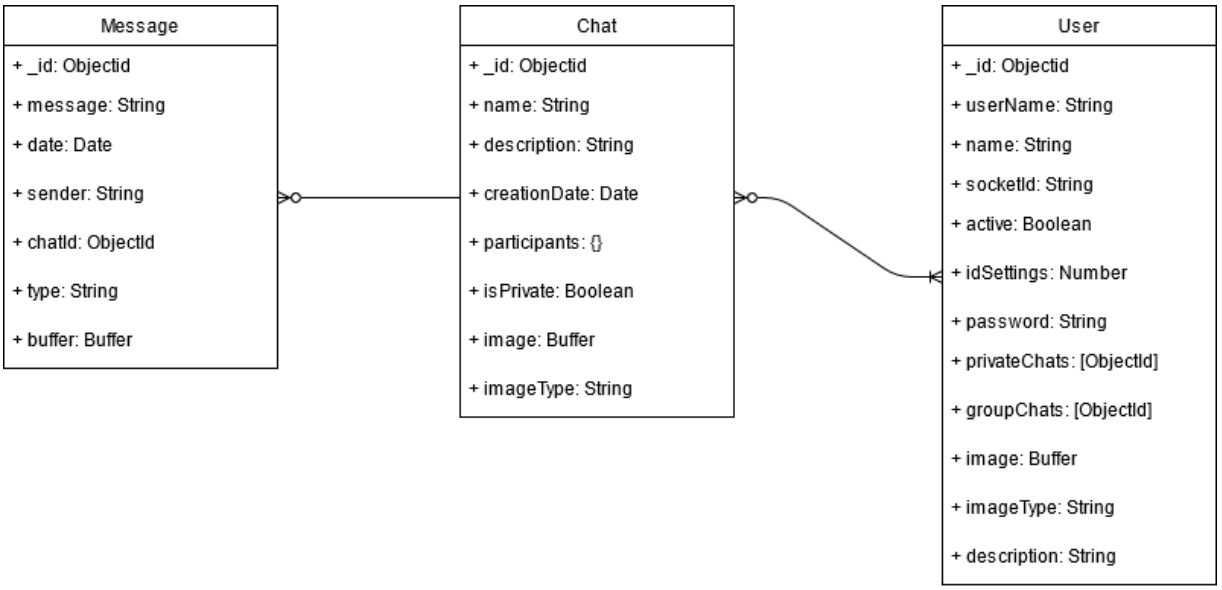


Diagrama 5 Diagrama Entidad Relación Base de Datos

Como podemos ver en el Diagrama 5 el eje central de interacciones es el Schema Chat, las relaciones que podemos encontrar en este Schema son las siguientes:

- Chat-Message: esta relación es de 1 Chat a n Message opcional, la razón de esto es que la existencia de Message depende de la existencia de Chat, un chat puede tener Message pero un Message no puede no tener un Chat asignado, esto supondría un problema ya que se quedaría sin una referencia en la base de datos. Para futuras implementaciones en las que se incluya la opción de borrado de chats se debería hacer un borrado de los mensajes de este chat para no desparejar esta relación. Al ser una base de datos no relacional no se puede hacer un borrado en cascada y se quedarían descuadradas las instancias de Message no borradas.

- Chat-User: esta relación es de n Chat opcional a n User, la razón de esta relacion reside en que un User no tiene por qué tener Chat creados todavía pero seguir existiendo, por el contrario la existencia de un Chat implica que mínimo en el caso de ser privado haya al menos dos User en este, siendo el caso de un grupo mayor la cantidad de usuarios que participan en este.

Cabe destacar que al estar usando una base de datos no relacional se puede hacer uso de estructuras polimórficas para los mismos Schemas. Este uso de las bases de datos no relacionales es bastante frecuente dado que es uno de los mayores beneficios que ofrece. En el caso de esta base de datos se ha hecho uso de esta facilidad en dos casos distintos, aunque los dos dentro del mismo Schema. Los dos casos son los siguientes:

- Caso 1 Existencia de imagen: como se mencionó anteriormente una de las características de la entidad Chat es su imagen. En la creación de un chat grupal el formulario especifica que se tiene que incluir una imagen para poder crear el chat grupal, dicho de otro modo, incluir una imagen para el chat grupal es un requisito para su creación. Esta situación no ocurre de igual manera para la creación de un chat privado, esto se debe a que para la muestra de la imagen de un chat privado se introduce la imagen de perfil del usuario opuesto, es decir, si Pepe y Juan crean un chat privado Juan visualizaría la imagen de perfil de Pepe miéntras que Pepe visualizaría la imagen de perfil de Juan. Es por ello que en la base de datos se pueden encontrar documentos que tienen como propiedades **`image`** e **`imageType`** y también se pueden encontrar chats en los que estas dos propiedades no figuren. Esta ventaja frente a las bases de datos no relacionales permite tener en cuenta esos campos solo si se va a procesar un chat grupal, situación para la que se conoce con antelación que tipo de chat se va a tratar.

- Caso 2 Participantes en un chat: a lo largo de la aplicación ha habido una serie de tomas de decisiones clave. Una de ellas fue el uso del Schema Chat con estructura variable, como lo hemos podido ver en el anterior caso. Este no es el único cambio realizado sobre este Schema, también se utilizó la propiedad **`participants`** de manera variable, en el caso de los chats grupales siendo esta propiedad un array de string miéntras que en los chats privados esta propiedad portaba un string como tipo. Esta situación se debe a que un chat grupal puede tener múltiples miembros, pero por el contrario un chat privado solo involucra a dos participantes. Es por ello que el guardado de un chat privado es más rebuscado, guardándose en la propiedad **`name`** el nombre del usuario que crea el chat y en **`participants`** guardar el nombre del otro usuario que participa en el chat pero que no lo creó. La razón de guardado de los datos en estas dos propiedades en vez de recurrir a usar el espacio de **`participants`** reside en el hecho de no tener que usar un array para guardar dos posiciones, y a la vez mantener como nombre del chat un valor nulo. A modo de aclaración a continuación se ha proporcionado un diagrama que indica los tipos de esta propiedad. Este diagrama corresponde con el formato de visualización de esquemas en bases de datos no relaciones que se basan en documentos. El Diagrama 6 se ha basado en un paper de investigación de tres doctores en informática de la Universidad de Murcia (Hernández Chillón, Feliciano Morales, García Molina, & Sevilla Ruiz, 2017). Al ser una tecnología practicamente en desarrollo y con pocos años de uso no se ha creado todavía un estándar para especificar mediante diagramas el cambio en un Schema de sus propiedades; a pesar de que es algo bastante habitual relacionado con estas nuevas tecnologías. Es por ello que se usó este formato de cara a la representación de esta situación.

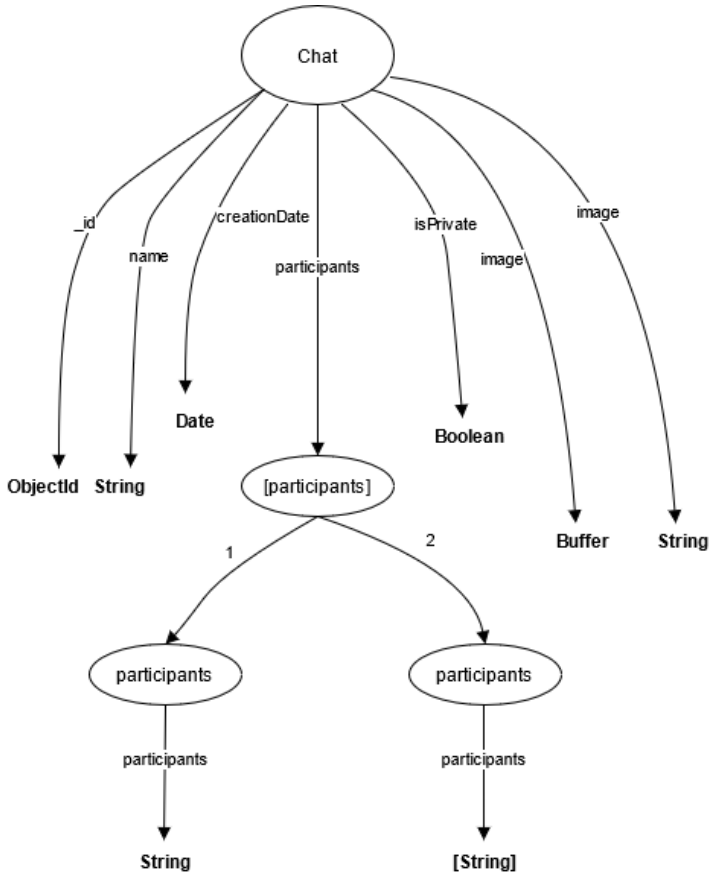


Diagrama 6 Diagrama de visualización de esquemas en bases de datos NoSQL

En el Diagrama 6 se puede ver el Schema de Chat desglosado, cada rama representa uno de los diversos atributos del Schema, una vez llegado el atributo de **`participants`** encontramos una división de dos posibles situaciones. En la primera la rama **`participants`** marcada por un 1 tiene valor de **`string`**, mientras que para la segunda rama marcada por un 2 el valor es de **`[String]`** representando un array de **`strings`**.

A continuación, se mostrará el Diagrama 7 que ejemplifica la estructura del componente principal de la librería. El diagrama siguiente solo hace referencia al componente ChatBox de la librería ya que es el componente con mayor interacción de la librería y el componente que permite una mejor ejemplificación del cambio que supone la introducción de la librería.

Se ha imitado el formato del diagrama inicial del apartado sin la librería añadida, a modo de comparación de los cambios que introdujo la librería en el diseño de la aplicación.

El primer cambio que más resalta es la desaparición de la conexión con el servicio Chat, que al incluir la librería las peticiones y envíos y recepciones de Web Sockets se evita tener conexión con este servicio y se integran sus métodos en la librería para empaquetar toda la lógica en el módulo ChatBox de la librería. Añadido a esto se integran las peticiones del servicio Main Chat también el servicio de la librería. Pero no se termina de eliminar la conexión con este servicio ya que es de dónde obtiene el chat que se ha abierto, para su paso como **`@Input()`** a la librería. Lo siguiente que vemos es que el controlador de ChatBox de la aplicación pierde gran parte de la lógica que le involucraba y sólo incluye en su vista correspondiente la directiva del módulo de la librería que va a integrar, siendo en este caso

ChatBox. Este cambio supone que la vista de la aplicación se reduce solo a una comprobación de la existencia de un chat, para que en caso contrario muestre un fondo diseñado. En caso de que sí que encuentre un chat abierto invoca a la librería y le pasa los datos para su muestra en pantalla mediante la vista de la librería, que incluye todo lo que incluía la aplicación en etapas previas.

Se han realizado también cambios en las tres vistas relacionadas con la parte principal de la aplicación de cara a que se visualicen en los dos diagramas los dos modos, diurno y nocturno. Estos cambios no afectan en nada al diagrama y son meramente visuales.

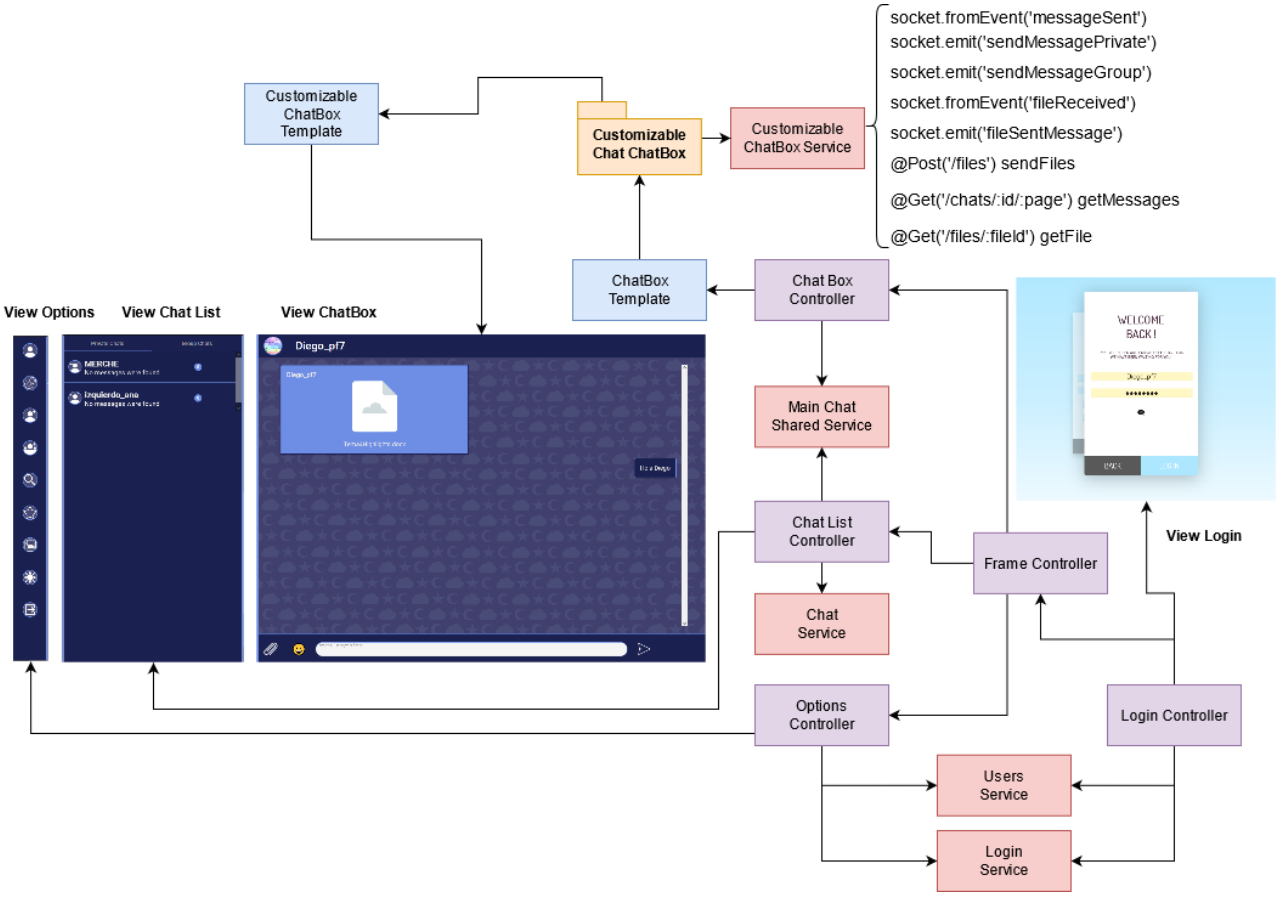


Diagrama 7 Diagrama de clases y vistas con el módulo ChatBox de la librería

Conviene a modo de finalización de la explicación de este diagrama tratar las peticiones que se realizan en el servicio que integra este módulo de la librería. La mayoría de estas corresponden al envío de mensajes tanto grupales como privados mediante un WebSocket, asignándose los textos “sendMessagePrivate” y “sendMessageGroup” a estas peticiones. Peticiones como “fileReceived” o “messageSent” corresponden con la comunicación por parte del servidor a los usuarios. La confirmación del envío de un archivo a descargar se realiza con el identificador “fileSentMessage”. Finalmente, el resto de las peticiones vía protocolo HTTP corresponden con la subida de archivos, la petición de los mensajes de un chat en caso de ser necesario, o la obtención de un archivo que recibió el usuario respectivamente.

VII. Diseño e implementación

En este apartado se tratará con más detalle la aplicación y librería internamente.

i. Decisiones de diseño

Una de las decisiones más relevantes en el desarrollo de la aplicación fue la separación del uso de Angular Material para el desarrollo de ciertos aspectos de la aplicación, relevándose el apoyo en esta librería para el desarrollo fundamentalmente de los diálogos de la aplicación debido a su comodidad. El uso de esta librería para el desarrollo de algunos componentes supuso un problema por su poca libertad, ya que los componentes generaban en tiempo de ejecución nuevos elementos en el DOM y con una gran encapsulación. Para solventar este problema a la hora de ofrecer temas distintos con la librería Angular Material se puede definir una serie de temas, y sustituir sus opciones principales a la hora de dar color mediante sus parámetros cambiando los valores de estos. Este parámetro llamado **`color`** permite opciones como **`primary, warn y accent`**. Estos diferentes valores permiten cambiar el color a una de las tres opciones establecidas que pueden ser configuradas. A priori, mediante el cambio del valor se debería poder establecer un tema, pero esto no llega a ser así de manera completa, en caso de requerir la modificación de los bordes o del encabezado del componente que se utiliza requiere de su cambio a través de las hojas de estilos o definiendo paletas que no llegan a ofrecer el resultado esperado, saltando en algunos momentos la encapsulación modificando los estilos y usando atributos con el modificador **`!important`** para situar esa propiedad como alta importancia por encima de otras que tengan el mismo valor. Esto se debe al uso de identificadores en algunos elementos compilados en vez de clases, la forma de obtener una mayor prioridad que la que proporciona un id es el uso de esa propiedad.

Por las razones previamente listadas se decidió evitar el uso de algunos componentes de Angular Material en algunos casos o saltar su encapsulación. De cara a incluir temas en la aplicación se cambió la forma de incluir estilos, previamente se usaba CSS para editar los estilos, pero principalmente por la creación de temas se cambió al uso de tecnologías como SCSS que permitió mantener gran parte del código ya escrito sin modificar ya que al ser un superconjunto de CSS no requiere de modificaciones (Borody, 2018). Esta tecnología permite la definición de variables y funciones, lo que carga un poco de lógica sobre la estilización, pero no afecta al rendimiento al ser posteriormente compilada a CSS. La primera parte importante de este código consiste en la definición de un **`mixin`** que es una forma de tener una serie de declaraciones de CSS que se quieren reusar a lo largo de la aplicación. Aprovechando esta funcionalidad que proporciona SCSS se definen una serie de temas mediante la declaración de un mapa de SCSS para el que se asociará a clave su valor, estas claves deben de ser iguales en los dos temas para que en el cambio de tema solo se requiera asociar por la clave a la que se accede. Esto proporciona no solo una opción de definir colores sino de definir también imágenes. Ahora en la aplicación cada vez que queramos asociar un valor a una propiedad que corresponda con los temas se usa la instrucción **`themed('')`** con el nombre de la clave del mapa a la que se quiere acceder entre paréntesis y comillas esto haría que se procesen tantos estilos distintos en la aplicación como temas se hayan definido, en este caso dos, uno por el “light” y otro por el “dark”. Esto no se ejecutaría de la manera esperada si no fuera por las dos funciones siguientes: **`themify`** y **`themed`** son las que se encargan de iterar por todos los temas correspondientes definiendo contenido que se empaqueta y compila posteriormente como se puede visualizar en el fragmento de Código 1; la función **`$theme-map`** se define de manera global para su

uso interno, de igual manera sucede con **$theme-map** solo que en este caso se define como nulo de manera global para no poder acceder al mapa de temas desde fuera del **mixin**.

```
$themes: (
  light: (
    bg-login: linear-gradient($light-blue, #caefff, $white),
    bg-app: $white,
    main-color: $light-blue,
    medium-color: $light-medium-blue,
    text: $black,
    text-right: $light-grey,
    background-chat:$white-smoke,
    background-textarea:$white,
    background-textarea-text:$black,
    highlight-text-color:$white,
    message-left-border:$light-blue,
    message-right-border:$light-medium-blue,
    message-right-fill:$light-blue,
    message-left-fill:$white,
    background-chat-opened: url("/assets/Fondo-Diurno2.png"),
    modal: rgba(255, 255, 255, 1),
    modal-text: #4e2f42,
    background-chat-closed: url("/assets/CityDay1.png"),
    user-default-icon: url("/assets/UserDay3.png"),
  ),
  dark: (
    bg-login: linear-gradient($black, $dark-blue, $dark-medium-blue),
    bg-app: $dark-blue,
    main-color: $dark-blue,
    medium-color: $dark-medium-blue,
    text: $white,
    text-right: #e6e6e6,
    background-chat:$dark-pale-blue,
    background-textarea:$white-smoke,
    background-textarea-text:$light-grey,
    highlight-text-color:$black,
    message-left-border:$dark-blue,
    message-right-border:$dark-medium-blue,
    message-right-fill:$dark-blue,
    message-left-fill:$dark-medium-blue,
    background-chat-opened: url("/assets/Fondo-Nocturno2.png"),
    modal: rgba(26, 32, 79, 1),
    modal-text: $white-smoke,
    background-chat-closed: url("/assets/CityNight.png"),
    user-default-icon: url("/assets/UserNight2.png"),
  ),
);




@mixin themify($themes: $themes) {
  @each $theme, $map in $themes {

    .theme-#{$theme} & {
      $theme-map: () !global;
      @each $key, $submap in $map {
```

```
        $value: map-get(map-get($themes, $theme), '#{$key}');
        $theme-map: map-merge($theme-map, ($key: $value)) !global;
      }

      @content;
      $theme-map: ¡null !global;
    }

  }
}

@function themed($key) {
  @return map-get($theme-map, $key);
}
```

Código 1 Definición de temas en SCSS

Como se mencionó previamente otra decisión de diseño que afectó al desarrollo de la aplicación fue la separación de los componentes de la aplicación, al ser una aplicación con una sola pantalla con gran interacción, dejando de lado los inicios de sesión y registros, tendría cierta cabida la propuesta de usar un solo componente que no requiriera de interacciones entre sí para tratar los datos, dotando a una vista de una gran carga de datos. Esta fue la idea principal y la razón por la que se construyó un componente llamado Frame, que posteriormente se reutilizaría.

En contraste con lo anterior se realizó un planteamiento de uso de una posible librería para crear un chat. Buscando de esta manera que cualidades se necesitarían para realizar este cometido. De este análisis se llegó a las siguientes conclusiones:

- Todos los chats de un mínimo de relevancia contienen siempre dos apartados de manera general, una interfaz con un listado de chats y otra interfaz distinta para tratar un chat desplegado, es decir, un chat que se ha abierto. Por consiguiente la librería que se fuera a desarrollar debería al menos contar con estas dos interfaces.

- Por lo general un chat siempre tiene un apartado de configuración del mismo, pero este no tiene una relevancia similar a un listado de chats y a un chat desplegado. Es por esto que este apartado debería ser parte de la aplicación pero no ser parte de la librería ya que cada aplicación que fuera a utilizar la librería se enfocaría en realizar los ajustes pertinentes para complementar a la librería.

- Finalmente, es bien sabido que las aplicaciones de mensajería no solo tienen su nicho en el mundo web, sino que, a día de hoy su mayor nicho se encuentra en el mundo de las aplicaciones móviles. En definitiva, los dos módulos de la librería deberían llegar a poder integrarse en aplicaciones móviles, de cara a la finalización del proyecto o a integraciones futuras. Con el propósito de realizar una futura integración de esta librería en dispositivos móviles se decidió implementar los módulos lo más adaptables a esta implementación. Debido a la naturaleza de estos dispositivos los tamaños que manejan no suelen ser tan grandes como lo son los de las aplicaciones web visualizadas por lo general en un portatil o una pantalla de mayor dimensión. De manera indiscutible se podría afirmar que dichos módulos no aparecerían a la misma vez en la pantalla de un dispositivo móvil, es

decir la navegación sería de pantallas distintas, una con el listado de chats y la otra con el chat desplegado, de manera que para alcanzar la segunda se clicará en uno de los chats de la primera, cambiándose de pantalla y de módulo, y en caso de querer retroceder hacerlo mediante un botón pero sin ser parte del segundo módulo. Esto supone que los dos componentes deberían estar separados, es decir, no pertenecer al mismo controlador.

Dicho lo anterior, la librería debería contar con dos componentes ChatList y ChatBox. Esto hizo que el componente mencionado anteriormente, Frame, se encargará de contener los componentes, esto haría que una vez iniciada la sesión de un usuario los datos del usuario se transmitieran al componente "padre" Frame, mediante el cambio de ruta, y este retransmitiera esta información a todos los componentes "hijos" en forma de **`@Input().`** Al no realizarse cambios de la información del usuario, por parte del usuario y que estos afecten al curso de estos componentes, se puede evitar el uso de métodos que se invoquen cuando se produzcan cambios en las variables, como **`ngOnChanges.`** Por lo tanto, la interacción que se realizará en la aplicación y con gran peso es la que se pueda realizar entre estos componentes.

Por esta decisión tomada es por lo que se tuvieron que crear servicios intermedios como MainChat, cuya razón de creación reside en comunicar por parte del listado de chats al componente ChatBox que chat se desea desplegar. A su vez, esto generó una gran complejidad en el código, plagando de interacciones entre estos dos componentes. Para evitar una reducción del rendimiento de la aplicación se intentó suplir esta reducción implementando métodos como el siguiente para aprovechar las peticiones que se realicen en su máximo. En vez de realizar peticiones cada vez que se abriera un chat y renovar la petición cada vez que se cambiara de chat, se prefirió cargar un listado de chats tanto privados como grupales en el componente ChatBox, estas variables irían cargando con cada apertura de un chat una posición con los mensajes cargados, y si el chat se había abierto con anterioridad simplemente no realizar la petición, así de esta manera ir cargando los datos de los chats de manera progresiva.

```
ngOnChanges(changes: SimpleChanges): void {
  for (const propName in changes) {
    if (changes.hasOwnProperty(propName)) {
      switch (propName) {
        case 'chatObs': {
          if (this.chatObs) {
            this.page = 0;
            if (this.chatObs.isPrivate) {
              this.position = this.findChatInList(this.listChatsPrivate,
this.chatObs._id)
              if (this.listChatsPrivate[this.position].messageList.length
== 0) {
                this.getMessages();
              }
            } else {
              this.position = this.findChatInList(this.listChatsGroup,
this.chatObs._id);
              if (this.listChatsGroup[this.position].messageList.length ==
0) {
                this.getMessages();
              }
            }
          }
        }
```

```
        }
      }
    }
}
```

Código 2 Carga de datos en el cambio de chat

En el fragmento de Código 2 se refleja el cambio de un chat haciendo uso de **ngOnChanges** (N., 2019). Antes de examinar el código interno es necesario hacer un apunte de por qué se hace uso de **ngOnChanges** de esta manera. **ngOnChanges** es un método que reacciona al cambio de cualquier dato que este enlazado a una directiva, en el caso de este código corresponde con cualquier propiedad que se introduzca en el componente ChatBox mediante la anotación **@Input().** Esto no generaría un problema en aplicaciones pequeñas en las que solo una propiedad se anotara con esa anotación, esto no es el caso de esta aplicación. Al reaccionar este método con cada propiedad supondría una ejecución innecesaria ya que solo se requiere de la notificación del cambio de la propiedad **chatObs**. Una solución muy típica y poco útil es la siguiente, que carga de mayor dificultad y no soluciona el problema.

```
OnChanges() {
  if (this.myFirstInputParameter) {
    this.doSomething(this.myFirstInputParameter);
  }
}
```

Con esto cuando se detectara que el primer parámetro ha cambiado se produciría el método **doSomething()** y se aplicaría sobre el parámetro, ¿esto funciona como debería? La respuesta es no. Esta función realiza correctamente su propósito si el parámetro que se comprueba es el último en ser modificado, en caso de ser el primero, se evaluaría la instrucción a verdadero dando paso a la ejecución de la función **doSomething(),** hasta ahora todo correcto. Pero ¿si ahora se modifica el segundo o el tercer **@Input()** que pasaría? La instrucción se evaluaría a verdadero porque el **myFirstInputParameter** habría cambiado su valor a uno cualquiera, y para cada parámetro se realizaría la ejecución de la función realizando modificaciones sobre el parámetro varias veces de manera innecesaria. Es aquí donde se introduce el uso de **SimpleChanges**, que vemos a continuación en el fragmento de Código 3 de manera genérica y del que se ha hecho uso en el fragmento de Código 2 ya adaptado a este proyecto.

```
ngOnChanges(changes: SimpleChanges) {
  for (const propName in changes) {
    if (changes.hasOwnProperty(propName)) {
      switch (propName) {
        case 'myFirstInputParameter': {
          this.doSomething(change.currentValue)
        }
      }
    }
  }
}
```

Código 3 ngOnChanges que reacciona a un parámetro concreto

SimpleChanges almacena qué parámetro ha realizado la petición, esto no soluciona que sea llamado varias veces, pero si permite conocer cuál de todos realizó la llamada y en caso de ser necesario también se podría obtener el valor anterior antes del cambio. No se hizo uso de esta funcionalidad ya que no aportaba ninguna facilidad añadida a la situación. Una vez reconocido que valor ha producido la llamada solo hace falta comparar si es el parámetro que se busca que se modifique y ya realizar la función en caso de que si lo sea.

Una vez explicado esto se puede enfocar la explicación en el código que se ejecuta a raíz de cada cambio de chat, que es el siguiente:

```
this.page = 0;
if (this.chatObs.isPrivate) {
  this.position = this.findChatInList(this.listChatsPrivate,
this.chatObs._id)
  if (this.listChatsPrivate[this.position].messageList.length == 0){
    this.getMessages();
  }
}else {
  this.position = this.findChatInList(this.listChatsGroup,
this.chatObs._id);
  if (this.listChatsGroup[this.position].messageList.length == 0) {
    this.getMessages();
  }
}
```

Código 4 Sección de ngOnChanges dedicada a cargar mensajes

En el fragmento del Código 4 en caso de ser privado se busca la posición del chat al que se ha cambiado mediante su identificador único, en la lista que le corresponde, es decir la lista de chats privados y, en caso de estar vacío el listado se realizaría la petición de mensajes, en caso de darse esta situación porque no se haya mandado ningún mensaje todavía no devolvería ningún mensaje y no afectaría al curso de la aplicación aunque no se podría evitar que esta petición se repitiera si el usuario cambiará de chat varias veces sin haber mandado un mensaje. De igual manera sucede para los chats grupales.

Aprovechando esta implementación se facilitó la integración de la recepción de mensajes de distintos chats, incluyendo los chats que no estuvieran abiertos, pudiendo recibir mensajes de distintos chats mientras un chat estuviera abierto, una vez se accediera a ese chat poder acceder a estos mensajes recibidos en el momento. Una de las funciones más relevantes a la hora de realizar las tareas de introducción de mensajes es la función encargada de encontrar la posición de un chat en la lista. Esta función es llamada cuando se recibe un mensaje de cualquier tipo para poder encontrar la posición correcta del chat del que proviene el mensaje. Dado que tiene una gran cantidad de intervenciones, veo pertinente la inclusión de esta, aunque su complejidad es reducida.

```
findChatInList(listChats: any[], chatId: String) {
  return listChats.findIndex(x => x.chatId == chatId);
}
```

Esta función pasada una lista como parámetro, siendo la lista de mensajes privada o grupal busca por el parámetro del identificador del chat la posición en la que el valor sea igual que el identificador del primer miembro de la pareja chat y lista de mensajes.

ii. Problemas encontrados en el desarrollo

Como es habitual en todo desarrollo surgen problemas por los que se tiene que tomar o bien decisiones de diseño, cambios en el código, refactorizaciones, pero sobre todo lo común a todas estas situaciones es la dedicación de tiempo a encontrar una solución. Este proyecto no iba a estar alejado de este tema. Por ello a continuación se van a explicar las diversas situaciones problemáticas por las que el proyecto ha pasado y se ha desarrollado solucionando los problemas.

El primer problema grave encontrado en la aplicación fue el funcionamiento incorrecto de la comunicación vía Web Sockets entre la parte trasera en NestJs y la parte frontal en Angular. Por la naturaleza de cualquier aplicación que implemente Web Sockets mínimo siempre van a existir dos funciones, conexión de un WebSocket al servidor de Web Sockets y desconexión del servidor. Al ser el proceso habitual de desarrollo es lógico que estas funciones sean las primeras en ser comprobadas si funcionan correctamente.

La situación que sucedió fue la siguiente, habiéndose finalizado el desarrollo de la parte frontal incluyendo la conexión de Web Sockets. En el lado del servidor se aprovechaba de la implementación de Web Sockets ya integrada en el **`@WebSocketGateway`** de Nestjs. Una vez configuradas las opciones de conexión y el CORS solo quedaba comprobar que la conexión funcionaba correctamente. Y funcionó, o al menos eso daba a entender, el WebSocket de la parte frontal mandaba el mensaje de conexión al servidor de Web Sockets y este le respondía posteriormente a haber guardado el valor del identificador del WebSocket.

Es decir, el funcionamiento de la aplicación era el esperado, llegados a este punto se comenzó el desarrollo de diversas funciones en la parte frontal para poder comenzar con el envío de mensajes entre usuarios. De manera complementaria se implementaron las funciones en el servidor que reaccionaran a estas peticiones vía Web Sockets. A la hora de ejecutar estas funciones no llegaban a conectar. Al ser funciones con distinto formato que el de la conexión y la desconexión se planteó que se habían programado incorrectamente. Tras revisar la documentación y probar distintos métodos de envío de mensajes tanto para el “backend” como para el “frontend” se llegó a la conclusión de que podría ser problema de la librería por ello se probó a usar una librería distinta en la parte frontal, usando la librería nativa Socket.io aprovechando que el lenguaje era TypeScript y permitía el uso de JavaScript en la aplicación. Se cambió la librería y tras una implementación del código haciendo uso de la última librería se testeo el funcionamiento de la aplicación y todo conectaba con corrección y se desconectaba correctamente pero el resto de las peticiones seguían fallando. Esto quería decir, por lo tanto, que el problema no eran las funciones diseñadas en primera instancia. Sino que el problema por el contrario provenía de la utilización de la librería de Web Sockets de NestJs.

Una vez investigada la traza del problema se descubrió que el problema partía de la integración de Web Sockets de manera nativa por parte de NestJs. El problema provenía de la versión que se encontraba en la tecnología de la parte trasera, al ser una tecnología en pleno crecimiento, la comunidad no estaba tan involucrada como con tecnologías como Angular. Es por ello, que no había mucha información al respecto de integraciones de tecnologías como Angular y NestJs de manera conjunta, y más aun incluyendo en el uso de estas tecnologías el uso de web Sockets. Se decidió hacer un cambio de ordenador, al realizar este cambio, se instalaron las dependencias necesarias y se valoró la posibilidad de que el problema proviniera de las dependencias. Y la realidad, era que era así, la librería

utilizada en el desarrollo de la parte frontal era ngx-Socket-io una librería que encapsulaba la implementación de Socket.io de cara a su uso en la parte frontal. Esta librería estaba en su versión 3.3.0, la versión proporcionada en la parte trasera para la implementación de los Web Sockets era la versión 7.6.0, tanto para la implementación del servidor de Web Sockets como para la librería de implementación de los Sockets. Esta versión en la parte trasera estaba por debajo de la compatibilidad con la versión de la parte frontal. Esto se debe a que el proceso de desarrollo de la tecnología Angular está todavía por encima del desarrollo de la tecnología NestJs, sobre todo para la utilización de las librerías de npm. Por esta razón se cambió la versión de la librería de la parte frontal, ya que la librería de la parte trasera no se podía modificar actualizando su versión, y se conseguiría degradando la versión de la parte frontal a la versión 3.2.0. De esta manera, y re implementando el código previo al cambio de librerías, debido a que la facilidad para programar con la primera librería era mayor, se cambió de nuevo a la librería ngx-Socket-io y ya con la dependencia degradada el funcionamiento de las funciones creadas en primera instancia fue el correcto, solucionando así el problema encontrado con los web Sockets.

El siguiente problema encontrado en el desarrollo de la aplicación fue la visualización de las imágenes en el chat. Para explicar este problema es necesario detallar como se realizaba el guardado de las imágenes en la base de datos y como se reinterpretaban en la parte frontal. El proceso seguido es el siguiente: un usuario sube la imagen en la parte frontal y esta imagen se envía a la parte trasera y se recoge gracias a la librería mencionada previamente, llamada Multer. Una vez procesada esta imagen se guarda en base de datos el buffer de datos que produce la imagen, este proceso es igual que el seguido para subir un archivo de cualquier formato. Al hacer una petición el usuario esta imagen se devuelve a la parte frontal y se interpreta procesando el array de bytes de información y transformando estos a una URL que se integra en el atributo **`img`** del HTML. Para realizar esto se hacía uso del siguiente fragmento de Código 5.

```
formatImage(img: any): any {
  // Converts arraybuffer to typed array object
  const TYPED_ARRAY = new Uint8Array(img.buffer.data);
  // converts the typed array to string of characters
  const STRING_CHAR = String.fromCharCode.apply(null, TYPED_ARRAY);
  //converts string of characters to base64String
  let base64String = btoa(STRING_CHAR);
  //sanitize the url that is passed as a value to image src attribute
  return
this.domSanitizer.bypassSecurityTrustUrl('data:'+img.type+';base64, ' +
base64String);
}
```

Código 5 Método que da formato a una imagen

Una vez procesado el dato el usuario podía decidir realizar la siguiente animación para visualizar la imagen en pantalla completa, ampliándose a pantalla completa como se puede visualizar en la Ilustración 13.

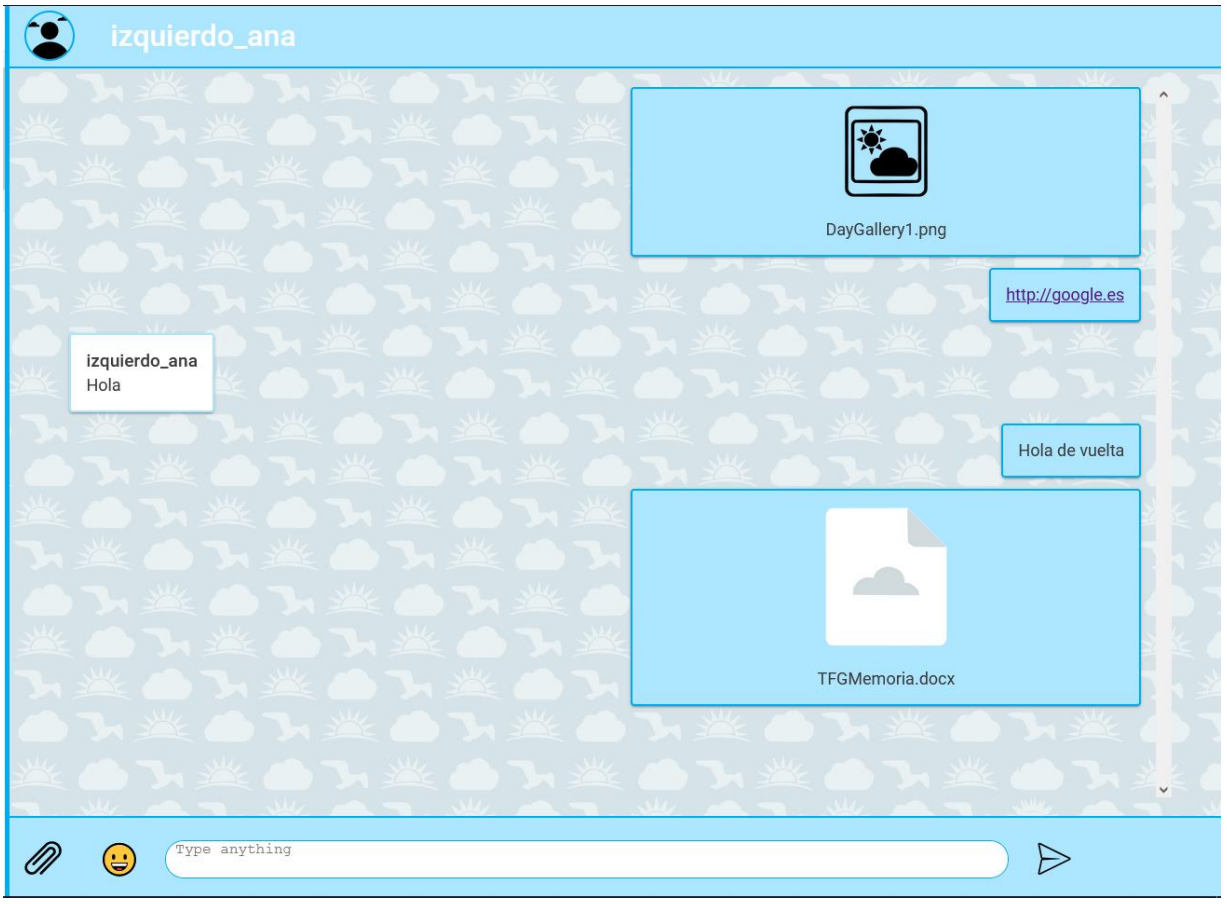


Ilustración 13 Chat con imagen sin ampliar

Pasando una vez clicada la imagen a lo visualizado en la Ilustración 14:

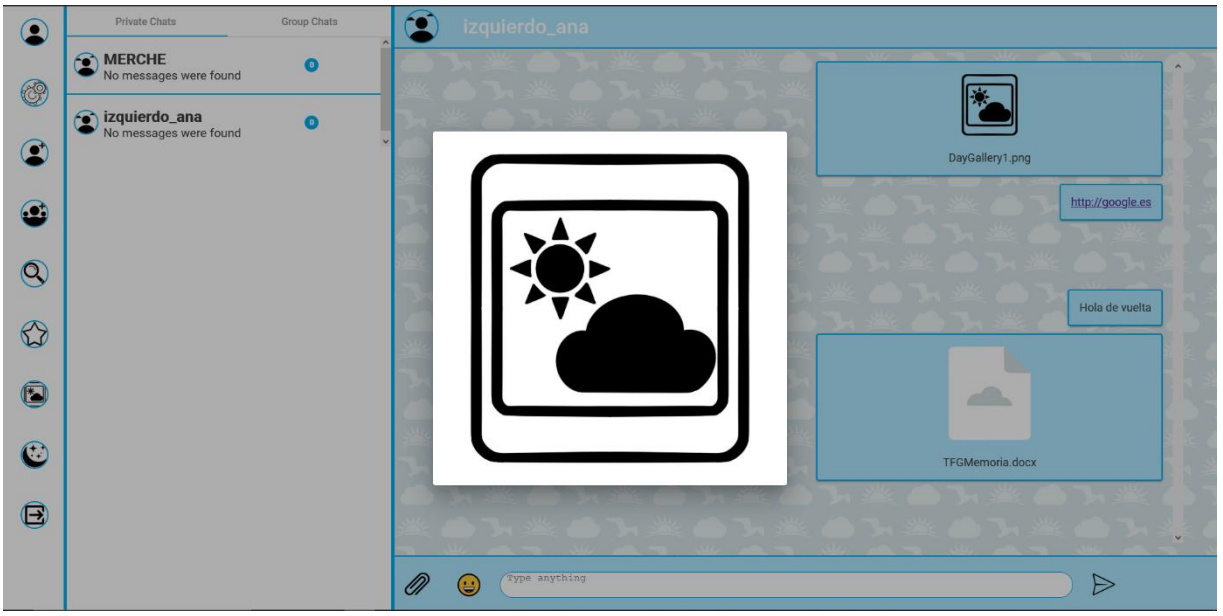


Ilustración 14 Chat con imagen ampliada

Esta función que ampliaba la imagen invocaba a la función primera para poder obtenerla en el formato completo y adaptándola correctamente al tamaño. Este proceso funcionaba correctamente, pero de cara a la integración de esto en la librería comenzó a funcionar de manera incorrecta, no permitiendo que se visualizaran las imágenes ni en pantalla completa y ralentizando la aplicación por completo. Se llegó a la conclusión de que el procesado era el correcto para las imágenes de pocos KB, pero cuando la imagen era de grandes dimensiones esta función ralentizaba todo el proceso bloqueando la carga de más datos. La siguiente función se implementó para reducir la carga de datos y aligerar este proceso. Esta función fue una reimplementación de la primera, pero aprovechándose de ventajas como las de la función **`reduce`** (Kotvas, 2019) que se puede visualizar en el Código 6.

```
formatImage(img: any) {
  if (img.type == "image/jpeg" || img.type == "image/jpg" || img.type ==
"image/png") {
    const base64String = btoa(new Uint8Array(img.buffer.data).reduce((data,
byte) => {
      return data + String.fromCharCode(byte);
    }, ''));
    img.image = this.domSanitizer.bypassSecurityTrustUrl('data:' + img.type
+ ';base64, ' + base64String);
  } else {
    img.image = null;
  }
}
```

Código 6 Prueba de envío de un mensaje con comprobación de link

Con esta función se podían mandar correctamente imágenes sin suponer problemas de rendimiento, pero no permitiendo que se ampliaran estas imágenes a pantalla completa. Se solucionó este problema transformándolo en un archivo enviable y que se pudiera abrir en otra pestaña nueva en caso de ser demasiado grande para ser procesado, solución que se aplica también para los vídeos debido a su tamaño.

iii. Diseño gráfico de iconos y fondos

De cara a ofrecer una interfaz de usuario agradable, se decidió diseñar mediante herramientas como Procreate y Photoshop. Una de las razones también de diseño de los fondos e iconos en vez de obtener plantillas gratuitas fue de cara a ofrecer una visión más personalizada que el de otras librerías del mercado, que proporcionaban una interfaz solo marcada por los colores e iconos por defecto, de las librerías con las que se desarrollaban. Este proceso de modelado y diseño conllevó un apartado e iteración del proyecto para su integración.

A continuación, se mostrarán los iconos y fondos más relevantes y su uso dentro de la aplicación.

Ilustración 15 Fondo Diurno para el chat sin desplegar

El fondo de la Ilustración 15 es el usado para el recuadro del chat sin desplegar a modo de acompañamiento de la interfaz. A continuación, en la Ilustración 16 su versión nocturna:

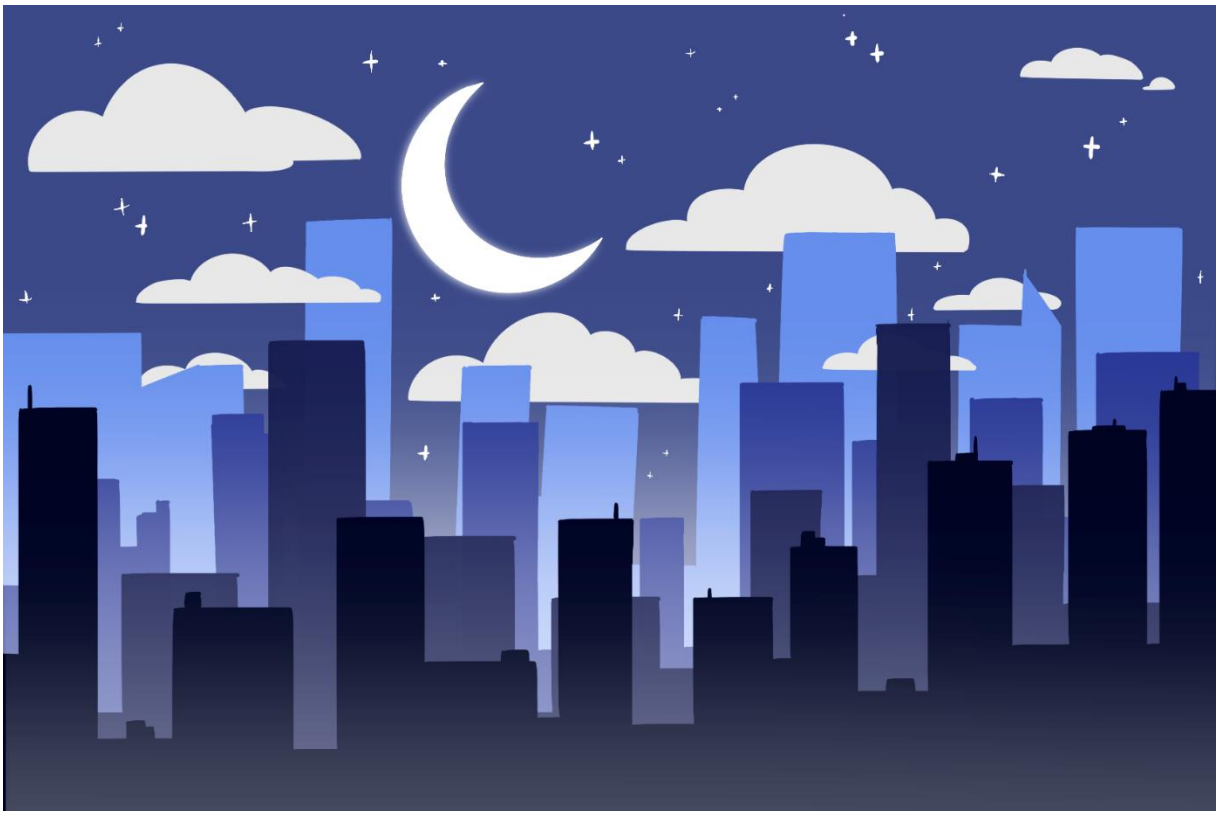


Ilustración 16 Fondo Nocturno para el chat sin desplegar

Para dar una sensación de relieve a los mensajes del chat se diseñó también un fondo siguiendo con la temática de los fondos mostrados. La diferencia con estos es que debería no ser muy invasivo para no desconcentrar al usuario de lo relevante de la aplicación, los mensajes. Es por esto por lo que se diseñó un fondo con estampado de iconos sencillos pero que siguieran la temática de la aplicación como se puede ver en la Ilustración 17.

Ilustración 17 Fondo Diurno chat desplegado

Y también su versión para el modo nocturno como se puede ver en la Ilustración 18:

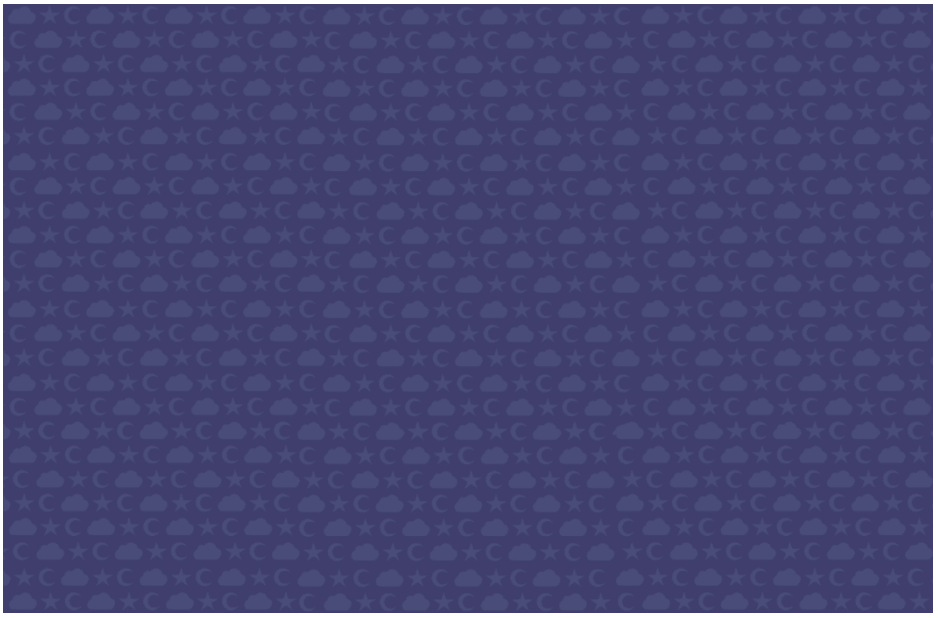


Ilustración 18 Fondo Nocturnos para el chat desplegado

Finalmente, se mostrarán los iconos correspondientes a los archivos que corresponden con la Ilustración 19, al botón de cambio de modo diurno a nocturno y viceversa y finalmente los iconos de los usuarios que no cambiaron su foto de perfil, que son los iconos más relevantes.

Ilustración 19 Iconos de archivos enviados

Estos tres iconos se alternan de cara a mostrar los archivos enviados o recibidos y a su vez mostrar el modo diurno y el modo nocturno. Siendo el segundo y el tercero usados en el modo nocturno, uno para el archivo que el usuario envía y otro si es el usuario el que lo recibe. En el caso de tener habilitado el modo diurno la combinación usada es la del tercero y el primer icono. Cabe destacar que estos iconos solo aparecen en caso de haberse enviado un archivo en formato distinto a image/jpeg, image/jpg o image/png, es decir el formato de las imágenes.

Los iconos de la ilustración 20 se corresponden con los utilizados en el botón de cambio de modo diurno a modo nocturno y de modo nocturno a modo diurno, respectivamente.

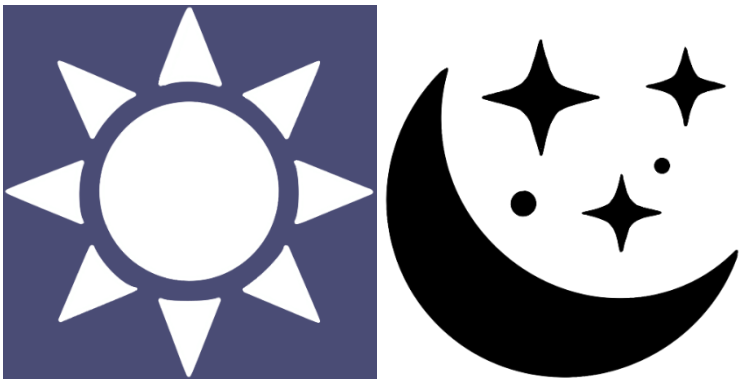


Ilustración 20 Iconos cambio modo diurno y modo nocturno

Finalmente, los iconos de perfil establecidos por defecto que corresponden con la Ilustración 21, y que todo usuario porta hasta que decida cambiarse por su foto de perfil personalizada .

Ilustración 21 Iconos del perfil de usuario por defecto

iv. Pruebas

En este apartado se tratarán la pruebas que se han realizado para la librería. Debido al tiempo reducido al abarcar distintos aspectos en cada iteración, las pruebas realizadas para el proyecto solo se realizaron en el apartado concreto de la librería. Al ser una gran cantidad de pruebas realizadas se hará un breve repaso de algunos aspectos importantes, en vez de un análisis completo.

Para el desarrollo de las pruebas de los componentes se han implementado mocks de los distintos **`@Input()`** que el componente tiene, de forma que se pudiera generar de manera correcta sin indicar que hay campos nulos y falle su creación. El componente ChatList está cubierto por completo y el componente ChatBox está cubierto en un 65%. Esto último se debe a la realización de pruebas sobre la recepción de peticiones de Web Sockets, con mayor complejidad que el envío, que sí fue comprobado. Las pruebas realizadas sobre los métodos abarcan todas las ramas del método, probando de esta manera que el método funciona correctamente en las diversas situaciones contempladas. Un ejemplo de esto es el visualizado en el fragmento de Código 7.

```
it('should format Image', ()=>{
  let data = messageImage;
  componentChatbox.formatImage(data);
  expect(data['image']).toBeDefined();
});


it('should not format Image', ()=>{

componentChatbox.formatImage(componentChatbox.listChatsPrivate[0].messageLi
st[0]);

expect(componentChatbox.listChatsPrivate[0].messageList[0].image).toBeNull(
);
});
```

Código 7 Pruebas relacionadas con formatear la imagen

En las dos pruebas del Código 7 se puede visualizar lo comentado anteriormente, por un lado, se contempla que se le ha ofrecido una imagen para cambiar y por lo tanto le da un formato que se guarda en la propiedad **`image`**, generada en el momento y de ahí su acceso mediante corchetes para comprobar si está definida, completando así la comprobación con una imagen. Por otro lado, se prueba a darle formato a un mensaje de texto, para lo que el método devolvería el valor **`null`** en el campo y se aseguraría de que el funcionamiento fuera el apropiado.

En cuanto a la introducción de los mensajes recibidos, se realizaron pruebas tanto para un mensaje de grupo como para un mensaje de chat privado, con este objetivo, previamente y mediante mocks se establecieron las listas de chats con mensajes también de prueba, para probar el comportamiento de los nuevos mensajes introducidos. Esta prueba corresponde con el fragmento de Código 8. En este fragmento se obtiene el tamaño de la lista de chats privados y tras introducir un mensaje en dicha lista se comprueba que haya aumentado en una unidad. El comportamiento para la prueba de adición de un mensaje en un chat de grupo es similar, solo se cambia la lista sobre la que se evalúa y los datos del mensaje, que tienen que cuadrar con el chat al que se envía.

```
it('should add a message to private list',()=>{
  let valueBefore =
componentChatbox.listChatsPrivate[0].messageList.length;

componentChatbox.addMessageToList(normalMessagePrivate.chatId,normalMessage
Private,true);
  let valueAfter = componentChatbox.listChatsPrivate[0].messageList.length;
  expect(valueBefore+1).toEqual(valueAfter);
});
```

Código 8 Prueba de adición de mensajes a la lista privada

De cara a controlar el envío de mensajes se realizó una prueba reflejada en el fragmento de Código 9, en esta prueba se asigna un valor al área de texto, pero con un link incluido para poder comprobar también que el envío de links funciona correctamente. Se realiza la llamada al método pertinente y se comprueba que el campo se ha vaciado, proceso que se realiza posteriormente al envío del mensaje, comprobando de esta manera que todo funciona según lo esperado.

```
it('should submit a message', ()=>{
  componentChatbox.textArea = "Esto es un test con link incluido
http://google.es"
  componentChatbox.onSubmit();
  expect(componentChatbox.textArea).toEqual("");
});
```

Código 9 Prueba de envío de un mensaje con comprobación de link

También se realizaron pruebas para el control del tipo de archivo que se enviaba, para el cambio del chat principal reflejado en ambos componentes mediante distintas pruebas, una enfocada al establecimiento del valor del chat principal y otra a las solicitudes pertinentes al recibir la notificación de un cambio de chat; la comprobación de la maximización de las imágenes y finalmente el cambio de animaciones para mostrar información sobre el chat abierto.

En cuanto al servicio al que realizar las pruebas, se puede indicar que se ha incluido entre sus dependencias el módulo **`HttpClientTestingModule`** de cara a realizar las peticiones de tipo HTTP y entre las propiedades del servicio se ha generado un Socket para poder emular el comportamiento de las peticiones vía Web Sockets. A continuación, se mostrará un ejemplo de cada uno de los tipos. Cabe destacar con anterioridad, que como ejecución previa de cada test se hace una conexión del socket y con la finalización de cada test se hace una desconexión.

Comenzamos con el de una petición HTTP visualizado en el fragmento de Código 10.

```
it('should send a file ',()=>{
  let url = 'http://localhost:3000/files';
  service.sendFiles(url,formSendFile ).subscribe(
    response=>{
      expect(response).toBeDefined();
    }
  );

  const req = httpMock.expectOne(url);
  expect(req.request.method).toBe("POST");
  req.flush(formSendFile);
});
```

Código 10 Prueba de envío de un archivo mediante el servicio al servidor

Este método envía un archivo y espera que la respuesta este definida, finalmente comprueba también que el tipo de la petición es el correcto y se deshace del mock usado en el **`@Post().`** Una vez introducidas las peticiones HTTP, el siguiente tipo de test a evaluar es una petición vía Web Socket, el planteamiento de esta ha sido un poco más complejo que el de cualquier test, debido a la poca documentación existente para el testing con Web Sockets.

```
it('Socket Send Private Message', ()=>{
  let m = new Message("Test","Diego_pf7", "60747a42c99c1902fe482867",
"message", null)
  spyOn(socket, "emit").and.callFake(function(){ socketWorks = true});
    service.sendMessagePrivate(socket,'sendMessagePrivate', m);
```

```
  expect(socketWorks).toBeTrue();
});
```

Código 11 Prueba de envío de un mensaje mediante WebSockets

La prueba exhibida en el Código 11 crea un mensaje a modo de ejemplo, a su vez coloca un espía en el método emit del socket para poder realizar el cambio del valor de la variable socketWorks en caso de que el socket realice la petición. Después de esto se llama al método a testear del servicio con los parámetros necesarios y se comprueba el valor de socketWorks asegurándose de que sea verdadero. En caso de no serlo querría decir que no se ha hecho la llamada correcta. Al ser una llamada independiente no es posible realizar una prueba completa y asegurarse de que se ha recibido correctamente la petición. En futuras implementaciones se podría incluir una serie de peticiones en el backend de cara al testing con Web Sockets, permitiendo que se pudieran realizar pruebas más completas.

Capítulo 4 Conclusiones y trabajos futuros

En este capítulo final se expondrán los objetivos alcanzados, se harán reflexiones personales sobre el trabajo realizado y que aspectos permiten una futura ampliación del proyecto.

I. Resolución de objetivos

Uno a uno, se han completado los objetivos fijados para este proyecto y que fueron los siguientes:

I. **Desarrollar una aplicación de mensajería vía chats simple:**

   Este objetivo se cumplió abarcando la mayor parte del tiempo del desarrollo del proyecto, debido a la construcción y diseño desde el inicio un proyecto completo. Este objetivo sirvió para sentar las bases del proyecto y para introducirse en las tecnologías nuevas para el autor como fueron NestJs y los Web Sockets.

II. **Mejorar visualmente la aplicación mediante diseño gráfico:**

   Este objetivo supuso un previo estudio de las librerías disponibles, comparando el alcance de la aplicación que se desarrollaba con el proporcionado por otras librerías, y sirvió para desmarcarse del resto de librerías. Se elaboró un diseño que resultó más atractivo para el entorno cercano al autor mediante los diversos iconos y fondos creados. Este apartado permitió que se aprendiera a realizar diseño gráfico a un nivel principiante.

III. **Incluir funcionalidades complejas para mejorar la aplicación:**

   Después de alcanzar los dos objetivos previos el nivel de familiarización con el entorno y las tecnologías era ya suficiente para que este objetivo solo supusiera un desarrollo normal de la aplicación y una integración simple de funcionalidades complejas. Para las que en algún caso se hizo uso de librerías externas que proporcionaban lo esperado por el desarrollador y eran perfectas para el desarrollo como la librería de emoticonos dotando de mayor abstracción del código en algunos puntos.

IV. **Crear una librería en base a los componentes más útiles de la aplicación:**

   Este objetivo estuvo marcado desde el principio por la realización de un estudio de los apartados más importantes de la aplicación en relación con un chat. También supuso la refactorización de la aplicación para obtener la parte útil de los componentes que pudieran integrarse en la librería. Este objetivo ponía al desarrollador en el lado opuesto, en vez de hacer uso de librerías como había ocurrido hasta ahora se iba a crear una y se tenía que buscar los defectos encontrados en otras para elaborarse de la mejor manera posible.

V. **Integrar la librería en la aplicación:**

   Finalmente, el objetivo que cerraba el ciclo de desarrollo del proyecto por el momento. En este objetivo se liberó de la lógica de los componentes de la aplicación

involucrados en la librería para pasar al uso de la librería creada en el anterior objetivo, no sin antes realizarse una serie de correcciones tanto de la librería como de la aplicación, completando finalmente los objetivos planteados para este proyecto. Al completar este proceso se publicó la librería en NPM.

El uso del desarrollo iterativo ha supuesto que se pudieran visualizar los cambios en la aplicación, entregando de una manera muy rápida un prototipo inicial, favoreciendo de esta manera también la adaptación a las nuevas tecnologías que en algunos puntos supusieron algunos retos debido a su nivel de madurez y a la poca comunidad que las soportaba.

II. Futuras implementaciones

Al haberse cumplido todos los objetivos las futuras implementaciones tienen relación más con funcionalidades diferentes a las expuestas hasta ahora o con la introducción de nuevas tecnologías. Por lo tanto, las propuestas para extender y mejorar este proyecto son las siguientes:

- Realizar la integración del envío de Stickers y del envío de audios, funcionalidades muy utilizadas en la mayoría de aplicaciones de chat moderno. Esta funcionalidad ofrecería al usuario más opciones de comunicación.

- Implementar una galería de archivos, esta funcionalidad aportaría al usuario una forma de acceder de manera única a los archivos sin tener que hacer una búsqueda entre los diversos chats. Pudiendo acceder al completo a todos los archivos recibidos, agrupando estos por chats o en su totalidad. Esta funcionalidad no es habitual entre las aplicaciones de chat y sobre todo con los archivos de imágenes supone una facilidad en el momento de su gestión.

- Integrar un sistema de pruebas automáticas mediante el uso de integración continua y generar pruebas que interactuen en mayor medida con el servidor, ofreciendo una mayor fiabilidad.

- Integrar un servicio de videollamadas con compartición de pantalla. El tamaño de esta integración es considerablemente mayor que el de las anteriores propuestas. Esta funcionalidad elevaría mucho el nivel de la aplicación e incluso se podría implementar esta funcionalidad en la librería haciendo de la librería una versión mucho más completa.

- Crear una aplicación móvil haciendo uso de la librería, en el caso de ser necesario realizar ajustes de la librería actualizar la librería a una nueva versión con su nuevo manual de uso (Vandehey, 2020).

III. Conclusiones personales

El proyecto realizado ha supuesto uno de los mayores retos de mi carrera, esto se debe a que ha sido el más grande en cuanto al tamaño, siempre en relación con la temática del desarrollo web, especialidad en la que me enfoco. A su vez ha combinado distintos aspectos que hasta ahora no se habían explorado en la carrera: el uso de Web Sockets, la utilización de bases de datos no relacionales (siendo más extensa la investigación de este punto al crear una base de datos alojada en la nube), el uso de tecnología nuevas como lo es NestJs usada para el desarrollo del servidor, ampliando así el conocimiento de este campo centrado hasta el momento en el uso de Spring Boot y la introducción al mundo del diseño gráfico para la creación de fondos e iconos propios para la aplicación, haciendo uso de herramientas como Procreate y Photoshop, campo del que tenía completo desconocimiento. Cabe destacar que era el primer proyecto de estas dimensiones que documentaba a tanto nivel.

En general, ya solo por el hecho de abarcar nuevas tecnologías y aprender a utilizar nuevas herramientas supuso un reto. Pero el proyecto no terminaba ahí, se tenía que realizar algo opuesto a lo que se había realizado hasta ahora en el grado. Hasta el momento se había hecho uso de distintas librerías en diversos proyectos, tanto personales como enfocados a las diversas asignaturas del grado, pero nunca se me había planteado el objetivo de diseñar una librería para aportar al resto de desarrolladores, en vez de realizar una aplicación que contuviera las funcionalidades esperadas, crear una librería que empaquetara las funcionalidades y que pudiera ser reutilizada por la comunidad. Este aspecto me introdujo en un mundo nuevo, aportando un grano de arena a la comunidad de desarrolladores. Esto último ha supuesto que valore la idea de seguir proporcionando a la comunidad, tanto de mejoras para la librería realizada en este proyecto como creando nuevas librerías en el futuro. Por estas razones este proyecto ha supuesto una mejora de mis habilidades en tecnologías como Angular o Docker, una mayor especialización en el campo del desarrollo web, que forma parte de la ingeniería de software y el descubrimiento de nuevos aspectos, que, aunque no estén completamente relacionados con este campo, suponen un añadido a mi formación de cara al futuro pudiendo señalar las herramientas Photoshop y Procreate.

Bibliografía

Angular Team. (21 de Octubre de 2020). *Angular*. Obtenido de v11.Angular.io: https://v11.angular.io/docs

Borody, D. (20 de Junio de 2018). *Medium*. Obtenido de Medium: https://medium.com/@dmitriy.borodiy/easy-color-theming-with-scss-bc38fd5734d1

Delgado, L. (29 de Diciembre de 2020). *freeCodeCamp.org*. Obtenido de freeCodeCamp.org: https://www.freecodecamp.org/espanol/news/angular-vs-react-cual-elegir-para-su-aplicacion/

Garzas, J. (24 de Julio de 2015). *Javier Garzas*. Obtenido de Javier Garzas: https://www.javiergarzas.com/2015/07/que-es-docker-sencillo.html

Gwartney, S. (10 de Mayo de 2021). *Digital Ocean*. Obtenido de Digital Ocean: https://www.digitalocean.com/community/tutorials/angular-socket-io

Hernández Chillón, A., Feliciano Morales, S., García Molina, J., & Sevilla Ruiz, D. (2017). Visualización de Esquemas en Bases de Datos. *Conferencia JISBD 2017*, 14.

Kniberg, H. (2015). *Scrum and XP from the Trenches.* Estocolmo: C4Media.

Kotvas, T. (13 de Mayo de 2019). *Medium*. Obtenido de Medium: https://medium.com/@tin.kotvas/this-solution-gives-me-type-errors-on-typed-array-since-were-giving-it-a-string-700cd9597fc2

N., R. (20 de Noviembre de 2019). *DEV Community*. Obtenido de DEV Community: https://dev.to/nickraphael/ngonchanges-best-practice-always-use-simplechanges-always-1feg

portaltic. (25 de Marzo de 2020). *Europa Press*. Obtenido de Europa Press: https://www.europapress.es/portaltic/internet/noticia-uso-apps-mensajeria-crece-50-ultimo-mes-coronavirus-20200325122914.html

Requeridos Blog. (29 de Noviembre de 2018). *Medium*. Obtenido de Medium: https://medium.com/@requeridosblog/requerimientos-funcionales-y-no-funcionales-ejemplos-y-tips-aa31cb59b22a

Stack Overflow. (28 de Febrero de 2020). *Stack OverFlow*. Obtenido de Stack OverFlow: https://insights.stackoverflow.com/survey/2020#technology-programming-scripting-and-markup-languages-professional-developers

Vandehey, S. (10 de Febrero de 2020). *Cloud Four*. Obtenido de Cloud Four: https://cloudfour.com/thinks/how-to-publish-an-updated-version-of-an-npm-package/

Anexos

En los siguientes anexos se proporcionará información sobre la aplicación y la librería. Esta documentación se puede encontrar en el repositorio de GitHub. Para la aplicación se adjunta el README del directorio principal y que corresponde con el anexo I, para la documentación sobre la librería se adjunta el README encontrado dentro del directorio de la librería customizable-chat y que corresponde con el anexo II. Cabe destacar que estos anexos mencionados están documentados en inglés para facilitar la comprensión de los desarrolladores.

I. Anexo I

2020-ng-customizablechat

This project is a chat application fully developed using Angular for the frontend and NestJS for the backend, the database corresponds to a cloud MongoDB database. It includes the development of the library customizable-chat uploaded to NPM, which allows the user to introduce completely a frontend chat library using HTTP and WebSockets.

Starting

The next instructions will allow you to install the project, deploy the different sections and allow you to start developing.

For the backend directory:

```
npm install

npm run start
```

For the frontend directory:

```
npm install

ng serve -o
```

For the library directory:

```
npm install
```

In case you want to modify the library in the proper directory for development

```
ng build customizable-chat
```

For the upload of a version compile it for production

```
ng build customizable-chat --prod
```

Executing the tests ⚙

In the library directory

```
ng test customizable-chat --code-coverage
```

Build with

- Angular - Frontend framework
- NestJs - Backend framework
- MongoDB Atlas - Database
- Docker - Deployment in containers

Author ✍

Diego Pascual Ferrer - Full Development - Diegopasfer1909

Licencia

This project is under the license Apache, version 2.0 - check the file LICENSE.md for more details.

With love ❤ from Diegopasfer1909

II. Anexo II

Ng-Customizable-Chat

This library was generated with Angular CLI version 11.2.11. It is built in the base of Angular Material version 11.0.3

Introduction

This library's purpose is the visual implementation of a chat. This library is composed of two components at least for the moment. Both components help in the making of a chat, but they have different purposes. On the one hand ChatList helps to manage a list of chats unopened. On the other hand, ChatBox helps to manage the principal interaction with an opened chat, including file upload, text messages, an emoji menu, an information menu of the chat that is displayed.

Chat List

As mentioned before, this component's purpose is to display a list of chats, with information about two different types of chats, private and group chats, In both of them the information that could be displayed is an image that describes the chat, the name of the chat and if it is a private chat, the name of the person involved; the last message sent to the chat and finally the quantity of messages unseen.

The inputs of this component are the following:

- [user]: This represents the user of application. The main fields are `idSettings` that references 1 for Day Theme and 2 for Night Theme, `userName` that represents the name of the user (it is also used to detect in private chats who the user is talking to).
- [chatChange]: This represents the info of the displayed chat. This variable is only to treat the display of the main chat and return to the component in which you are using this component of the library. It is a Subject you can subscribe to gather information of the chat the user wants to display, as you display the chat the moment the user clicks in it.
- [privateChats] || [groupChats] : This represents an array of chats one for each type, the info that both have to have in common is: the `image` field, a chat `name` , chat `participants` in which in the case of group chats is an array of strings with the names of the users and in the case of the private chats is a string with the other participant, and the name corresponds to the creator of the chat also the name.

> Note: The marked words correspond to the name of the inner variable of the input.

ChatBox

As mentioned before, this component's purpose is to display a chat, with information about the chat, the description of the chat, the image, a dropzone for file messages, a form of image display, an emoji menu, a list of messages with info of the sender.

The inputs of this component are the following:

- [user]: This represents the user of application. The main fields are `idSettings` that

references 1 for Day Theme and 2 for Night Theme, `userName` that represents the name of the user (it is also used to detect in private chats who the user is talking to).

- [chatObs]: This represents the info of the displayed chat. The main fields of the chat are the following: the `image` of the chat in the case of group chats, in the case of private chats you should include an image of the user whose chat is opened, the `name` and `participants` with the same function as in Chat list component, `isPrivate` field is used to detect if the chat is of private or group type.

- [listUrls] : This represents an array of strings corresponding with the URLs or socket messages that the component is going to use to send messages and interact with the API, listUrls[0] --> where to send files via http request, listUrls[1] --> socket message for a private message, listUrls[2] --> socket message for a group message, listUrls[3] --> socket message to indicate to the users that a file is saved in database and was sent to the chat, is indicates the users that they have to do an http request to obtain the file, this is cause sockets have size limits in messaging. listUrls[4] --> indicates the base URL used to ask for messages, listUrls[5] --> corresponds with the message of the socket when receiving a message and listUrls[6] --> when a file is sent.

- [listChatsGroup] || [listChatsPrivate]: It is an array of chats, one of private chats other of group chats. This contains in chatId the id of the chat and in messageList the list of messages of the chat, the format of the messages is the following:

> Note: `class Message {message: String; date: Date; sender: String; chatId: String; type: String; buffer: Buffer}`

- [socket]: This is user's socket that will be used for communication with the WebSocket's service of the API (currently using ngx-socket-io for the implementation of the sockets)

> Note: The marked words correspond to the name of the inner variable of the input.
>
> Note: For text messages use type: 'message', for the rest of the formats include the explicit format, f.e. 'image/png'

The chat box depends on ngx-socket-io for the socket implementation, the emojis depend on the PickerModule of @ctrl/ngx-emoji-mart, the file dropzone depends on the NgxDropzoneModule of ngx-dropzone.