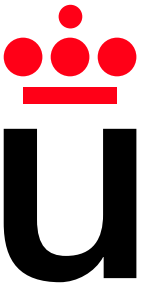


Universidad
Rey Juan Carlos

Escuela Técnica Superior
de Ingeniería Informática

Grado en INGENIERÍA DE SOFTWARE

Curso 2022-2023

Trabajo Fin de Grado

CHAT MEJORADO EN APLICACIÓN DE VIDEOCONFERENCIA

Autor: Jorge Esteban Pérez
Tutor: Micael Gallego Carrillo

Licencia

Agradecimientos

Quiero agradecer tanto a mis profesores del grado superior, como a mis profesores de la carrera y a mi tutor, como a toda la gente que me ha ayudado a interesarme cada vez más por este mundo y que me ha enseñado todo lo que sé hasta este momento.

Le doy las gracias a toda mi familia, amigos y a mi pareja por apoyarme en todo momento y darme los ánimos que necesitaba en los momentos más difíciles.

Resumen

El proyecto realizado trata de mejorar el chat de la aplicación de videoconferencia OpenVidu, a través del TFG "Plataforma de chats en grupo implementada en Angular y Nestjs", realizado por Diego Pascual Ferrer. La idea es que se implemente este TFG junto con el software de OpenVidu Call para ofrecer un chat mejorado, que incluso otras plataformas de videoconferencias del momento, como puede ser Google Meet. Estas mejoras son: la posibilidad de enviar ficheros de cualquier formato, tanto de texto como imágenes; la posibilidad de enviar emoticonos y la identificación por usuario a través del uso de cookies, sin necesidad de inicio de sesión.

El proyecto está desarrollado en su totalidad en Angular con la idea de que se pueda seguir escalando para futuros proyectos. La parte de frontend viene dada por la propia aplicación de OpenVidu, donde hemos incluido la parte de chat que funciona a través de señales propias, otorgadas por la aplicación que nos facilitará la comunicación en tiempo real entre los usuarios y de protocolos HTTP, que se comunican con la parte del backend. En esta parte hemos adaptado los archivos que tiene la aplicación OpenVidu, que están desarrollados en JavaScript, y la que hemos desarrollado para que funcionen conjuntamente en Angular junto con la base de datos, que es una base de datos MongoDB ejecutada a través de un contenedor Docker.

Este documento describirá los siguientes apartados correspondientes a cada capítulo: en el primer capítulo, se relatarán la motivación y los objetivos que se deben alcanzar en el proyecto; el segundo capítulo, se hablará de la metodología utilizada y las distintas tecnologías, lenguajes, librerías y herramientas que se han usado para desarrollar el proyecto en su totalidad; en el tercer capítulo, comentaremos las distintas decisiones de diseño que hemos adaptado, frente a los errores que hemos ido encontrando a lo largo del desarrollo, las distintas opciones de diseño gráfico por el que hemos optado; y por último, en el cuarto capítulo, se realizará la conclusión junto con las posibles futuras mejoras y de los conocimientos aprendidos durante el desarrollo del proyecto. Al final del documento se describirán los comandos necesarios para la ejecución correcta del proyecto.

Índice de contenidos

Índice de tablas

Índice de figuras

1

Capítulo 1: Introducción

En este capítulo se describirán los contextos que rodean al desarrollo de esta aplicación.

Para empezar, nos encontramos con dos aplicaciones totalmente desconocidas, OpenVidu Call y la aplicación de TFG "Plataforma de chats en grupo implementada en Angular y NestJs ", a la cual se referirá como aplicación de chat.

OpenVidu Call es una aplicación de videoconferencia y de comunicación en tiempo real, desarrollado con tecnologías WebRTC, para poder retransmitir audio y vídeo, y con Socket.io, para la comunicación en tiempo real. En cuanto a su desarrollo, la parte de *frontend* está desarrollada en Angular y la parte de *backend* en Node.js.

La aplicación de chat es una aplicación de mensajería instantánea en tiempo real, que permite a los usuarios establecer conversación en cualquier momento de manera simultánea. En esta aplicación se ha desarrollado la parte de *frontend* con Angular y la parte de *backend* en NestJs. Además, también utiliza la tecnología de Socket.io para la transmisión de información en tiempo real implementada con una base de datos MongoDB Atlas para el almacenamiento de información en la nube.

La aplicación de OpenVidu Call, es una aplicación de videoconferencia que cuenta con numerosas librerías, que están integradas y ocultas en su propia aplicación, de las cuales apenas se pueden modificar sus comportamientos, pero si se pueden hacer uso de ellas en su desarrollo. Entre las características, se encuentra con un sistema de autenticación básica que permite a los usuarios unirse a una sesión simplemente con un clic. Antes de unirte a la sesión te encuentras con una

página, que se ve en la figura 1.1, en la que te permite editar tanto el nombre, como la cámara web o el micrófono que quieres utilizar durante la sesión.

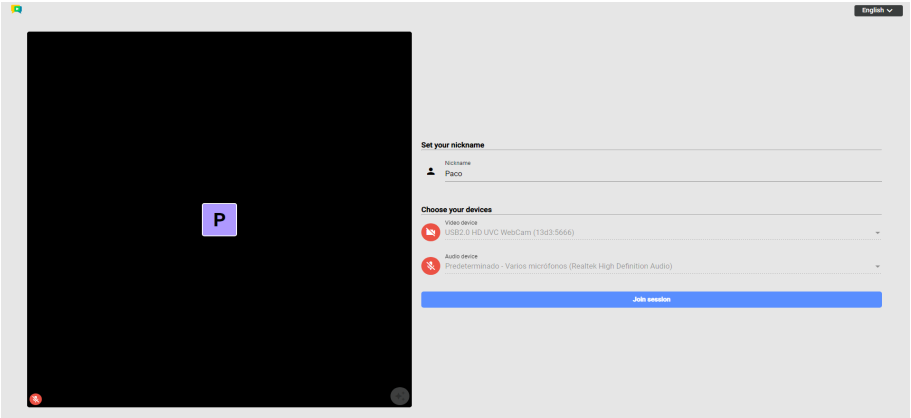


Figura 1.1: Página de configuración del usuario

OpenVidu Call cuenta con numerosas funcionalidades que hacen que la aplicación sea bastante versátil a su modificación, esto permite que se pueda desarrollar y ampliar la aplicación al gusto del programador. La interfaz principal del *software* es bastante simple como se puede observar en la figura 1.2, cuenta con numerosas funcionalidades como: poner la conferencia en pantalla completa. empezar a grabar la conferencia, la posibilidad de aplicar efectos de fondo de la cámara sin que tape al participante y la capacidad de poner subtítulos durante la conferencia. Esta última es una de las funcionalidades más exclusivas que se han encontrado en una aplicación de este estilo, ya que ni Google Meets cuenta con una funcionalidad que esté a la altura.

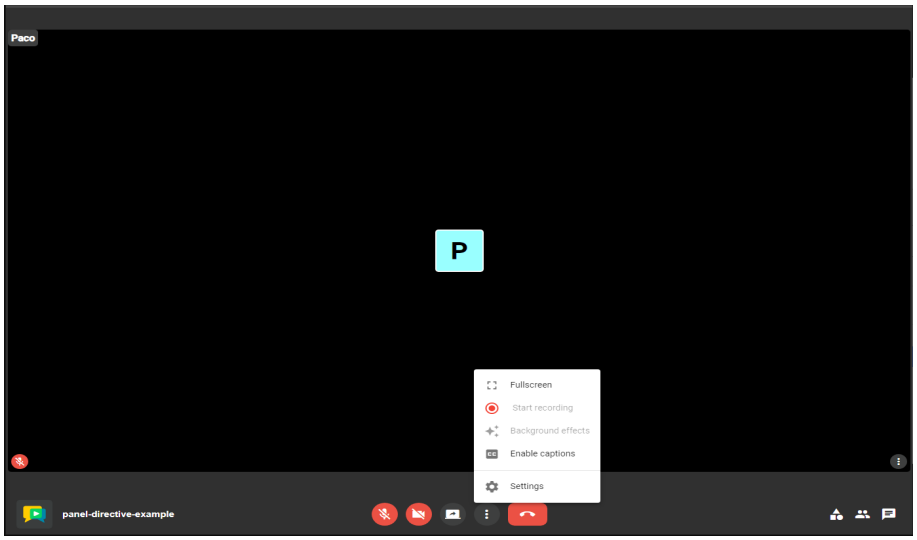


Figura 1.2: Interfaz de OpenVidu Call

Por otra parte, la aplicación de chat está desarrollada como una librería con dos componentes, ChatList y ChatBox, que juntas forman la aplicación. Esta aplicación es una aplicación web de mensajería, muy similar a Whatsapp Web. La parte de la librería de ChatList y de la aplicación cuentan con una gran cantidad de funcionalidades, como se observan en la figura 1.3, que son: la posibilidad de modificación de tu usuario, la creación de un chat privado con un usuario, la creación de un chat en grupo en el que puedes incluir varios usuarios al mismo tiempo con una imagen y nombre de grupo, la opción de establecer un modo nocturno o diurno. También se puede observar cómo en la parte derecha, se encuentran los chats del usuario y la opción para cambiar entre los chat privados y los chat grupales.

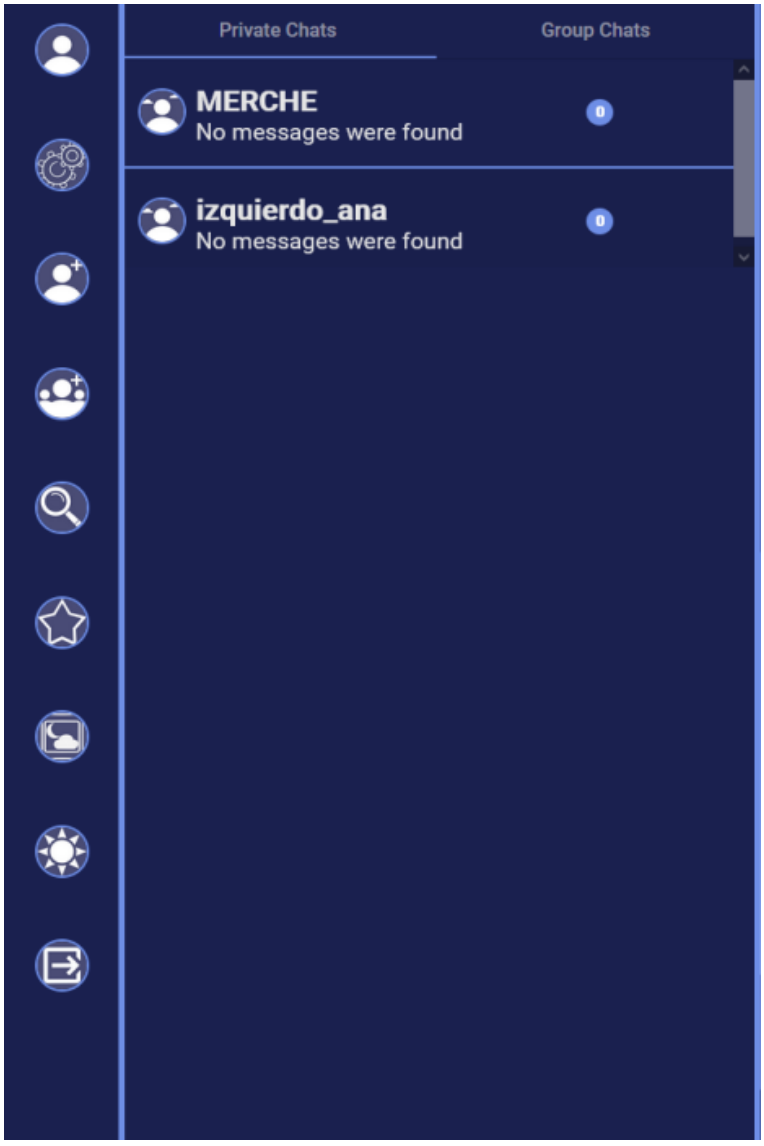


Figura 1.3: Interfaz de la librería ChatList

Y luego contamos con la parte referente a ChatBox, la que será la parte más útil para el desarrollo del proyecto y observamos en la figura 1.4, nos encontramos con que se divide en tres áreas: la parte superior, donde se encuentra el nombre del usuario con su imagen; la parte del medio, donde se encuentran los mensajes tanto nuestros como los del otro usuario; y en la parte inferior, donde encontramos las funciones de envío de ficheros, lista de emoticonos y el área de texto donde escribimos el mensaje junto con el botón necesario para enviarlo.

Figura 1.4: Interfaz de la librería ChatBox

Gracias a la documentación de ambas aplicaciones, podemos conocer qué paquetes se han utilizado en la aplicación de chat y qué distintas funciones podemos utilizar que nos otorga OpenVidu. Entre estas, los denominados *signals*, que son señales que se envían a todos los usuarios que están en una sesión. En este caso, por ejemplo, son los mensajes enviados por el chat como si fueran los Web Sockets utilizados en la aplicación de chat.

En la actualidad, se utilizan a diario aplicaciones de videoconferencias con chat incorporado, tanto para impartir clases online como para reuniones de trabajo o reuniones privadas entre amigos. Por lo que la idea de realizar una implementación de una aplicación de videoconferencia con una aplicación de chat en vivo permite aprender cómo funcionan internamente el resto de las aplicaciones comerciales que existen hoy en día, como son Skype o Microsoft Teams.

Existen una gran variedad de aplicaciones de conferencia y de chat que son entre ellas muy parecidas pero que cuentan con funciones o características que las hacen únicas. y que hacen que tengan un uso más presente en grupos específicos de usuario. Por ejemplo, Microsoft Teams es un programa que se utiliza mayoritariamente para entornos de trabajo o escolares, su interfaz se puede observar en la figura 1.5, porque cuenta con funciones como: la administración de usuarios, inte-

gración con otras aplicaciones empresariales y diferentes funciones de seguridad. Previamente para asuntos empresariales se usaba bastante Skype Empresarial, pero con el surgimiento de Microsoft Team, la mayoría de empresas, debido a las características anteriores, han preferido hacer uso de esta última. También existen otras como Discord, que es más utilizado para un uso más privado y personal.

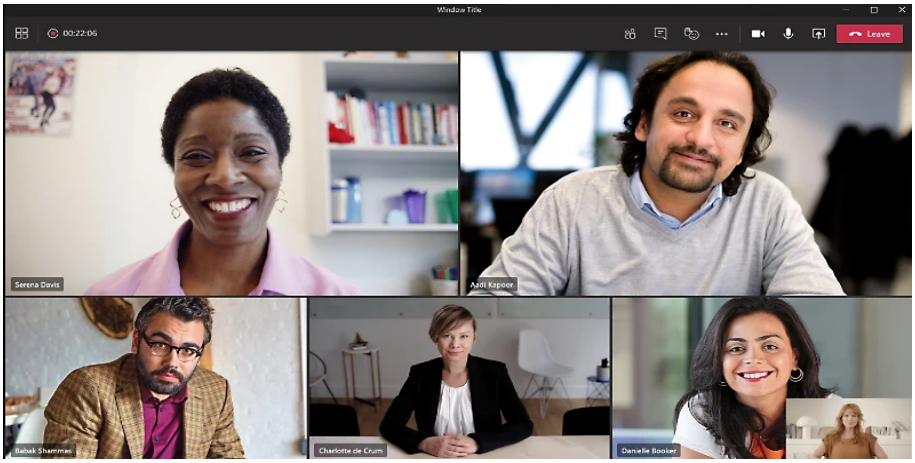


Figura 1.5: Interfaz de Microsoft Teams

En cuanto a aplicaciones de Chat, la que destaca por encima de las demás es WhatsApp Web, la cual cuenta con una interfaz parecida a la de la aplicación de chat como se puede observar en la figura 1.6. Existen otras como son Telegram o incluso Discord, que cuenta con su propia parte de chat tanto grupal como privado.

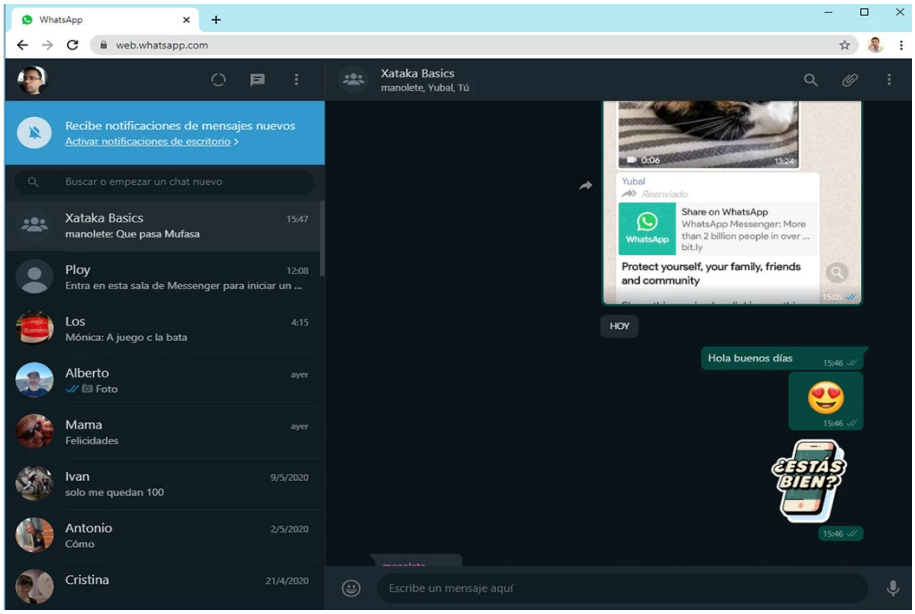


Figura 1.6: Interfaz de WhatsApp Web

Gracias a la implementación de una aplicación de mensajería instantánea junto con otra de videoconferencias, es que puede contar con la posibilidad de enviar ficheros por el chat. Esta función es bastante útil a la hora de tratar de transmitir información sin necesidad de utilizar aplicaciones de terceros o que se tengan que enviar de forma más rebuscada, como puede ser el caso de Google Meet, en el cual el fichero que quieres enviar tiene que ser subido previamente a Google Drive o a otra aplicación de almacenamiento en la nube y ser compartida mediante un enlace web.

Una de las ideas que a veces no se tiene en cuenta a la hora de crear este tipo de aplicaciones, que van a ser utilizadas por miles de personas, es la opinión del usuario. No utilizar o realizar estas pruebas de usuario común, son uno de los errores más llamativos que cometen la mayoría de programadores y las que suelen conducir al desastre e inutilización de la aplicación. La opinión del usuario tiene que ser crucial para la creación de la aplicación, ya que va a ser él, quien utilice de forma continuada esta aplicación y tiene que ser sencilla a la par que cómoda para el usuario.

Nuestro principal objetivo es implementar una sola aplicación fusionando las dos aplicaciones anteriormente descritas, que están desarrolladas por separado, para que funcionen conjuntamente y se puedan apoyar la una a la otra. La idea es, además de fusionarlas, tratar de mejorarlas y poder crear una aplicación que cuente con todas las funciones útiles de las dos aplicaciones.

2

Capítulo 2: Objetivos

Este apartado explicará los objetivos que se pretenden conseguir en este proyecto. Los objetivos que se nombrarán a continuación reflejan el desarrollo punto por punto que tuvo el proyecto, aunque algunos de ellos se han estado desarrollando de manera simultánea por decisiones de diseño:

1. **Unificación de las aplicaciones**, este es el principal objetivo en el que se tiene que conseguir que las dos aplicaciones funcionen como si fueran una única. También se pretende fusionar las funciones de ambas para que estén lo más interconectados posibles y poder hacer uso de la mayoría de características que nos aportan las aplicaciones. Contando con la aplicación OpenVidu Call como base, que será la que aporte servicios para las funciones de la aplicación de chat.

2. **Satisfacer al usuario**, como se ha explicado en el apartado de motivación, uno de los objetivos es que la aplicación satisfaga las necesidades del usuario en cuanto al diseño visual. La aplicación tiene que ser cómoda para el uso de cualquier usuario y no sea innecesariamente compleja.

3

Capítulo 3: Tecnologías, Herramientas y Metodologías

En este capítulo se desarrollará en detalle las tecnologías, herramientas y metodologías que se han implementado durante la realización del proyecto. Se empezará explicando cada apartado en el orden descrito en el título del capítulo.

3.1. Tecnologías

En este apartado se explicarán las tecnologías que hemos necesitado y utilizado para la implementación y el desarrollo de la aplicación. Se empezará explicando la tecnología Angular, utilizada para el desarrollo del *frontend*, seguido de NestJs que se ha utilizado para la implementación del servidor en el *backend*, junto con MongoDB para la base de datos almacenada en local y por último, Docker que nos permite la ejecución tanto del servidor de MongoDB, el servidor de OpenVidu Call y para el despliegue al final de la aplicación en un contenedor.

3.1.1. Angular

Angular es un *framework* de código abierto desarrollado por Google, para la fabricación de aplicaciones mantenibles y robustas. La primera versión surgió en el año 2009, desarrollada por Miško Hevery, esta versión estuvo basada en JavaScript. Y en 2016 se lanzó Angular 2, lo que se conoce hoy en día por Angular,

que esta creado desde cero y basado en TypeScript (Vanessa Marely Aristizabal Angel, 2021). Actualmente, Angular se encuentra en la versión 16.0.3.

Como se ha especificado anteriormente, Angular está basado en el lenguaje de programación TypeScript, este lenguaje es un superconjunto de JavaScript por lo que cualquier navegador web que admita JavaScript, podrá ejecutar de forma exacta cualquier código TypeScript. Debido a ello cuenta con numerosas características propias de JavaScript, como el tipado estático y la orientación a objetos. Además, este lenguaje añade ciertas funcionalidades basadas en ECMAScript, como pueden ser las promesas.

Angular se basa en una arquitectura de componentes, por lo que una aplicación Angular está dividida en componentes que se combinan entre sí y que son reutilizables en todo momento. Cada componente encapsula su lógica, su plantilla y sus propios estilos lo que hace que Angular tenga una estructura desacoplada, que junto con los enrutamientos y la inyección de dependencias hace que Angular sea un *framework* bastante potente para la creación de páginas y aplicaciones web.

Siguiendo con el párrafo anterior, el enrutamiento de Angular permite manejar diferentes rutas y enlazar componentes a ellas en una aplicación web. Es una característica bastante útil, ya que permite la navegación entre varias vistas. También contamos con la inyección de dependencias, esta característica mejora la reusabilidad y modularidad del código debido a que la inyección se hace sobre interfaces y no sobre implementaciones concretas, también ayuda al desarrollo ágil y al desarrollo de pruebas unitarias.

Todo esto, junto con la comunidad con la que cuenta Angular, permite que tenga una gran cantidad de recursos y servicios que pueden ser útiles, para el desarrollo de nuestra aplicación. También cuenta con soporte activo por parte de Google, por lo que recibe actualizaciones periódicas que incluyen: mejoras de rendimiento, correcciones de errores y nuevas funcionalidades.

Angular es uno de los *frameworks* más utilizados a nivel mundial, pero no cuenta con el monopolio, existe un competidor muy fuerte llamado ReactJs.

ReactJs es una librería de JavaScript de código abierto, desarrollado por Facebook, que últimamente ha obtenido una gran fama en el desarrollo web. Esta librería se centra sobre todo en la reutilización de componentes, y cuenta con una gran característica que la hace notable: la actualización eficiente del DOM.

Esta característica se enfoca en comparar el estado actual del DOM con los cambios realizados dentro del componente para realizar solo los cambios necesarios, consiguiendo que se minimice el coste de rendimiento de la aplicación.

React cuenta con una sintaxis denominada JSX, que nos permite mezclar HTML y JavaScript en un mismo fichero, lo que facilita la creación de los com-

ponentes y su renderizado en el navegador.

Como conclusión, con toda la información y con el conocimiento previo que se tenía sobre Angular, se optó a utilizar esta tecnología frente a ReactJs, porque a la hora del desarrollo se consideraba mejor tener una experiencia previa para poder trabajar más rápido y poder utilizar los conocimientos para solucionar errores. Además de que las dos aplicaciones, OpenVidu Call y la del chat, están desarrolladas con Angular, lo que supone un gran avance en el desarrollo de la aplicación.

3.1.2. NestJs

NestJs es un *framework* de desarrollo de *backend* basado en TypeScript. Este *framework* se desarrolló en 2017 y se lanzó de forma definitiva en 2018, se creó con la finalidad de que existiera una tecnología que se pudiera utilizar para el desarrollo del *backend*, aprovechando otras tecnologías como Angular u otros *frameworks*.

Al estar basado en TypeScript significa que destaca en la programación orientada a objetos y en la arquitectura de capas. También cuenta con la inyección de dependencias, que permite el intercambio de componentes, y con gestiones de solicitudes HTTP, que nos permitirá comunicarnos con el *frontend* y con la base de datos.

NestJs cuenta con los famosos decoradores, una característica que se utiliza para anotar clases, métodos e incluso propiedades. Estos decoradores nos permiten definir controladores, inyectar dependencias y definir autenticación y autorización. Estos decoradores se definen utilizando el símbolo '@' y seguido del nombre del decorador. Siguen una sintaxis parecida a la que podemos encontrar con los decoradores del *framework* Spring de Java.

Se eligió utilizar NestJs por encima del resto, debido a que la parte del *frontend* iba a estar desarrollado en Angular por ello, se optó por unificar ambas partes utilizando el mismo *framework*, ya que iba a ser más cómodo a la hora de ejecutar y desarrollar.

3.1.3. MongoDB

MongoDB es una base de datos NoSQL, que está orientada a documentos y de código abierto, esta base de datos nos permite almacenar, recuperar y administrar grandes volúmenes de datos, por lo que para nuestra aplicación es bastante acertado.

Esta base de datos carece de esquema por lo que los documentos, que se

guardan en las colecciones, no tienen por qué tener los mismos datos, esto permite mucha más flexibilidad que cualquier otra base de datos.

Además, tiene mecanismos de tolerancia a fallos que, a pesar de errores en los servidores, siempre se tiene la disponibilidad continua de los datos. Por lo que es muy utilizado en aplicaciones web y móviles, es porque cuenta con numerosos controladores y librerías para diversos lenguajes de programación, lo que facilita enormemente su integración en muchas aplicaciones.

Tiene una versión de servicio *online* llamada MongoDB Atlas, que se está volviendo muy popular en los últimos años, ya que ofrece mantenimiento constante sin necesidad de que el usuario tenga que mantener la infraestructura por su cuenta y la posibilidad de integración con otros servicios *online*, como AWS y Azure. MongoDB Atlas cuenta con la posibilidad de acceder y consultar los datos a través de una API unificada y con diferentes prácticas integradas que aseguren que los datos estén siempre disponibles y todo funciona de forma adecuada. ([Implemente una base de datos multicloud], s.f.)

3.1.4. Docker

Docker es una plataforma de contenedores de código abierto, que permite almacenar y desplegar aplicaciones sin necesidad de instalar todas librerías y tecnologías en nuestro dispositivo, simplemente con el código, librerías, dependencias y configuraciones.

Esta plataforma destaca sobre todo en la portabilidad, ya que permite ejecutar cualquier aplicación en cualquier sistema operativo que tenga Docker. Esto libra de errores en diferentes dispositivos, debido a que realmente no se está ejecutando en ellos si no en el contenedor. Entre otras características, se encuentra el aislamiento, esto significa que, al estar ejecutándose la aplicación en el contenedor, esta no afecta a otras aplicaciones o servicios.

Como conclusión, Docker es una herramienta muy potente en cualquier desarrollo y despliegue de una aplicación, además, de la posibilidad de la implementación en cualquier entorno.

3.2. Lenguajes

En esta sección se comentarán los diferentes lenguajes que se han empleado para el desarrollo de la aplicación. Como se puede observar en la figura 3.1, el lenguaje que más destaca en el proyecto es TypeScript.

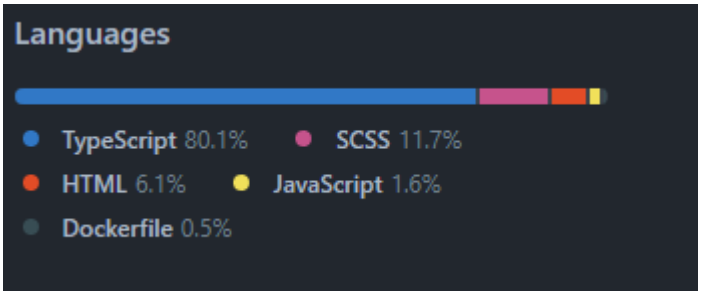


Figura 3.1: Lenguajes más utilizados

3.2.1. TypeScript

TypeScript es un lenguaje de código abierto desarrollado por Microsoft, creado como un superconjunto de JavaScript, ya mencionado anteriormente, lo que proporciona todas las características de esta y, además, añade funcionalidades nuevas.

Este lenguaje añade un sistema de tipos estáticos, esta característica consiste en que se puede declarar y especificar los tipos de datos de manera explícita en el código, lo que otorga una serie de beneficios como verificar los tipos de datos en tiempo de compilación, la detección temprana de errores y una mejor comprensión del código. Esto, en un desarrollo de una aplicación grande, permite que sea más robusto y mantenible.

Como hemos mencionado anteriormente, al ser un superconjunto de JavaScript significa que utiliza el mismo compilador, esto es una gran ventaja ya que cualquier navegador web es capaz de leer el código desarrollado en este lenguaje (José Luis Chacón,2021). Además, también puede utilizar cualquier código o librería desarrollada en JavaScript por lo que es un lenguaje bastante potente y versátil al mismo tiempo.

En la figura 3.1, que se ha mostrado al inicio del apartado, TypeScript es el lenguaje que más destaca en el proyecto debido a que iba a ser el lenguaje en el que se desarrollaría el *frontend*, por lo que para que fuera monótono, y no mezclar varios lenguajes y distintos programas, se decidió unificar en usar TypeScript en ambas partes. Con la idea de usar Angular para el *frontend* y NestJs para el *backend*. Además, se contaba con que la aplicación de chat estaba desarrollada también en su mayoría con TypeScript por lo que haría más fácil la implementación en la nueva aplicación.

3.2.2. SCSS

SCSS es una extensión del lenguaje CSS, que permite escribir los estilos más legibles, reutilizables y modulares, además de permitir el anidamiento de selectores. Como surge de una extensión de CSS, este lenguaje se permite compilar y ser interpretado por cualquier navegador web y aplicarse a cualquier elemento HTML.

3.3. Librerías

En este apartado se explicarán las librerías que se han utilizado y que son más relevantes en la aplicación.

3.3.1. Angular Material

Angular Material es una biblioteca de componentes de interfaz de usuario que permite hacer uso de componentes estilizados que siguen el diseño de Material Design de Google. Estos componentes incluyen numerosos elementos de diseño como botones, formularios y más. Además, proporciona herramientas que permiten modificar los componentes al gusto del desarrollador ([¿Qué es Angular Material?], s.f.)

Una de las competidoras principales de esta librería es React Material-UI, que también sigue los principios de Material Design para React. Como Angular Material, esta librería otorga una amplia gama de componentes y, ambas librerías se preocupan en proporcionar componentes reutilizables para ayudar al desarrollo de aplicaciones web. La principal diferencia es el *framework* en la que quieras desarrollar tu aplicación web, ya que ambas son dos librerías muy potentes.

Una de las bibliotecas más populares es Bootstrap. Esta librería, a diferencia clara de Angular Material, puede ser utilizada bajo cualquier *framework*, destaca por su sistema de rejilla flexible y por los componentes preestilizados. Bootstrap es una librería de código abierto que fue creada por Twitter y ha sido muy famosa por la gran versatilidad y compatibilidad con todos los navegadores web.

Para el proyecto se utilizó Angular Material ya que se iba a utilizar Angular para el desarrollo del *frontend*, además de que ambas aplicaciones, OpenVidu Call y la aplicación de chat, utilizan esta librería.

3.3.2. OpenVidu-Angular

OpenVidu Angular es una librería que proporciona OpenVidu Platform, sobre la cual está construida OpenVidu Call, que ofrece diferentes componentes y servicios para la comunicación en tiempo real y para la gestión de sesiones y conexiones.

Esta librería es la que nos va a permitir utilizar los *sockets* internos para poder establecer una conexión en tiempo real entre los usuarios que están conectados a la misma sesión. Estos servicios nos van a servir para el desarrollo de diferentes funcionalidades que serán útiles en futuras implementaciones.

Esta biblioteca, como la mayoría de Angular, hacen uso de Angular Material para los componentes gráficos que estos otorgan.

3.3.3. ngx-dropzone

Esta librería consiste en un componente para la subida de archivos en la parte del *frontend*. Permite una zona de arrastre o selección de ficheros, como se puede observar en la figura 3.2, además, permite configurar el límite de tamaño de los archivos e indicar los formatos de archivos que serán aceptados. Entre otras características, ofrece la capacidad de previsualizar el contenido y la posibilidad de eliminar los ficheros de forma individual antes de la subida. Esta biblioteca se encuentra en la aplicación de chat y es útil para su implementación en la aplicación ya que ofrece una funcionalidad que apenas se encuentra en otras aplicaciones de videoconferencias.

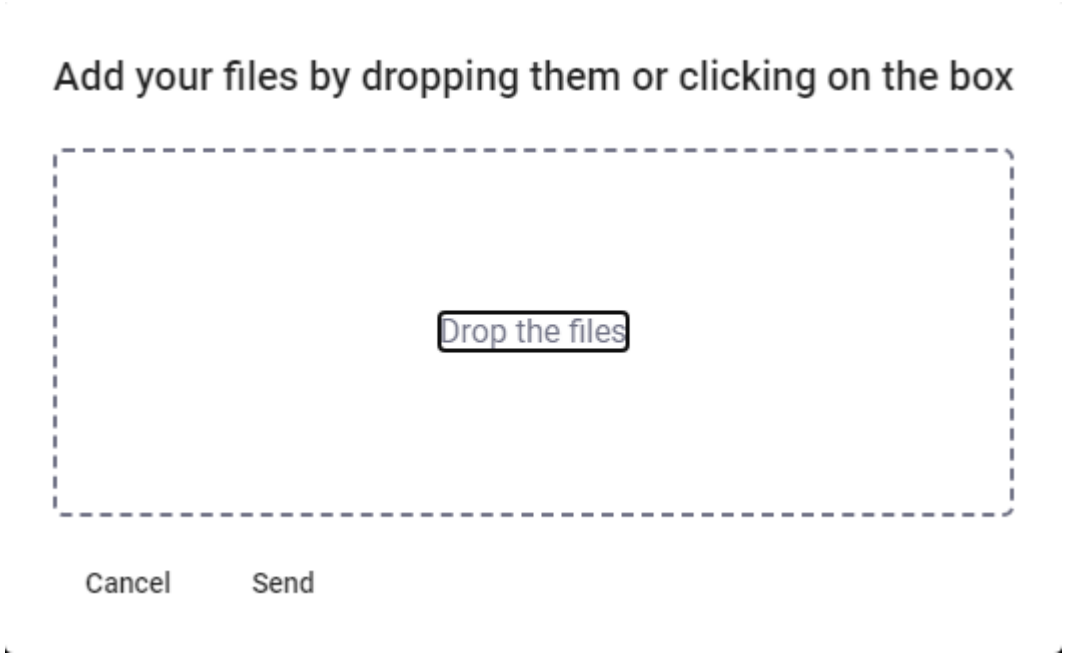


Figura 3.2: Caja de envío de ficheros

3.3.4. ctrl/ngx-emoji-mart

Esta librería implementa el uso de emoticonos en el chat. El componente despliega una caja que contiene una lista de emoticonos donde el usuario podrá seleccionar el que quiera como se pudo observar en la figura 3.3 . Estos emoticonos pertenecen a los sets de emoticonos de *softwares* como Apple, Google y Facebook, aunque también permite la introducción de emoticonos propios.

Esta librería ya estaba implementada en la aplicación del chat por lo que vista la utilidad que ofrece se decidió que era interesante e innovador incluirla en la aplicación.

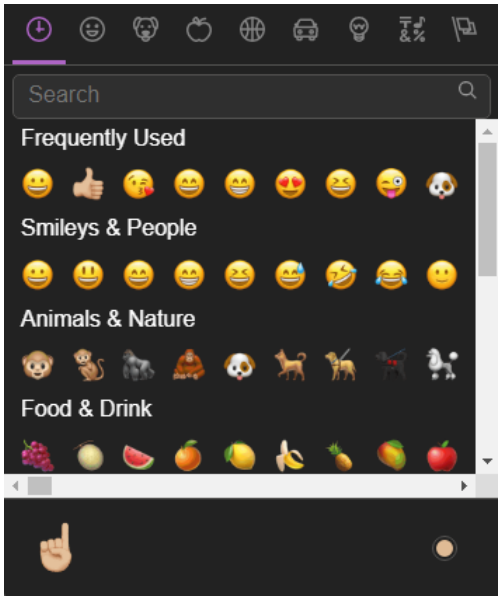


Figura 3.3: Caja de selección de emoticonos

3.3.5. Jasmine

Esta biblioteca ofrece un paquete de funciones y herramientas para la creación y ejecución de las pruebas unitarias en aplicaciones JavaScript, aunque, como se ha comentado anteriormente, también funciona para TypeScript. Esta librería proporciona una sintaxis que permite definir pruebas y especificaciones para poder comprobar que las partes de la aplicación funcionan de la manera esperada.

Aunque Jasmine se pueda utilizar como un framework que proporciona una estructura y sintaxis para poder escribir las pruebas, realmente es una librería.

3.3.6. Playwright

Playwright es una librería que permite realizar tests fiables *end-to-end* para aplicaciones web modernas ([¿Qué es Playwright?], s.f.), esto permite asegurar la funcionalidad completa de la aplicación, tanto las acciones como el flujo de trabajo de un usuario final. Esta biblioteca, además, genera datos sobre los test realizados, lleva a cabo pruebas de rendimiento y un informe sobre los test ejecutados, estas funcionalidades adicionales hacen que el estudio sobre la aplicación sea completa.

La librería Playwright cuenta con un gran soporte en Angular que permite desarrollar y ejecutar los test de una manera sencilla. Además, podemos en su página web la documentación y la API para los lenguajes más utilizados en el desarrollo web.

3.3.7. Mongoose

Como se ha explicado en la parte de tecnologías, para nuestra aplicación se utiliza una base de datos MongoDB y para que se pueda trabajar y establecer conexión con ella se necesita de esta librería que, otorga las herramientas necesarias para el *backend* desarrollado en NestJs.

Esta librería proporciona funciones y herramientas que permitirán poder definir modelos de datos y la realización de todo tipo de consultas a cualquier base de datos MongoDB.

3.4. Herramientas

En el nuevo apartado se detallará qué herramientas se han utilizado para el desarrollo de la aplicación entre ellas podremos encontrar Visual Studio Code para la realización del código de toda la aplicación, MongoDB Compass para la administración de la base de datos utilizada, Git y GitHub para el control de versiones, entre otras aplicaciones que aparecerán a continuación.

3.4.1. Visual Studio Code

Visual Studio Code es un IDE gratuito de código abierto desarrollado por Microsoft, que está disponible en varios sistemas operativos. Este IDE es uno de los más populares en cuanto a desarrollo de código fuente ya que ofrece una interfaz muy intuitiva, como se observa en la figura 3.4. Además, una función de extensiones que permite ampliar las características de esta herramienta dando soporte a multitud de lenguajes y tecnologías. También, existen extensiones que permiten modificar el editor al gusto de cada programador, lo que lo hace un IDE bastante completo teniendo en cuenta que es gratuito. Esta característica hace que los desarrolladores puedan utilizar simplemente un mismo IDE para realizar todo tipo de desarrollo sin tener que estar descargando y utilizando diferentes IDES al mismo tiempo, lo que ahorra en tiempo y carga para el propio ordenador.

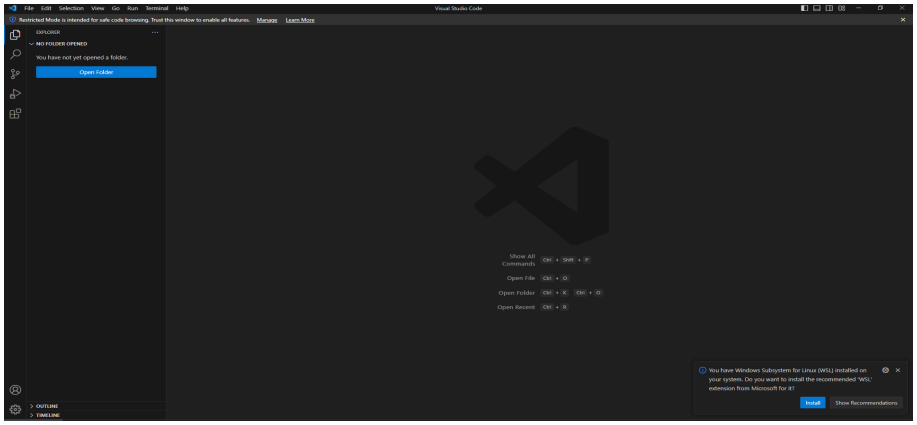


Figura 3.4: Interfaz del IDE Visual Studio Code

Otra de las características que hizo que se decantara por el uso de este IDE, es el recuadro que sale en la figura 3.4 abajo a la derecha. Este recuadro es una notificación que utiliza Visual Studio Code, que, si detecta un archivo con un lenguaje o una tecnología que está instalado en tu sistema, te otorga la posibilidad de que instales su extensión para que puedas trabajar con él desde el mismo IDE, esto te quita de tener que buscar en internet e instalar de forma manual todos los *softwares* necesarios para su ejecución.

Existen otras alternativas a Visual Studio Code que podrían haberse utilizado que comentaremos a continuación:

- IntelliJ IDEA: es un IDE muy interesante para el desarrollo en Kotiln o Java, también ofrece una característica importante como es el autocompletado y las sugerencias inteligentes ([¿Qué es IntelliJ IDEA?], s.f.) algo en lo que no destaca mucho Visual Studio Code e incluso incluye una completa integración con herramientas de control de versiones, como son Git o SVN. A pesar de estas características, se descartó debido a que no se conocía los lenguajes y tecnologías que se iban a implementar en el proyecto por lo que se optó por la versatilidad de Visual Studio Code.

- Atom: es un editor de código gratuito, que cuenta con una gran comunidad de desarrolladores que amplían continuamente a través de complementos. Aunque la comunidad de Visual Studio Code es más extensa y cuenta con una mayor variedad de opciones, también, aunque cuente con la posibilidad de ampliar los lenguajes y tecnologías como Visual, cuenta con una gran integración con Angular y TypeScript por lo que al estar el proyecto enfocado en este lenguaje y esa tecnología hace que sea la principal elección como IDE.

3.4.2. Microsoft Edge

El navegador que se ha utilizado para el desarrollo de la aplicación fue el Microsoft Edge. Se eligió este navegador frente a otros más populares como Firefox o Google Chrome, porque es el navegador inicial de la mayoría de ordenadores de cualquier usuario que utilice Windows como sistema operativo y aunque otros sistemas operativos no cuenten con este navegador, una de las ventajas es que está desarrollado con la tecnología Chromium por lo que lo hace prácticamente idéntico a Google Chrome, como se puede ver en la figura 3.5.

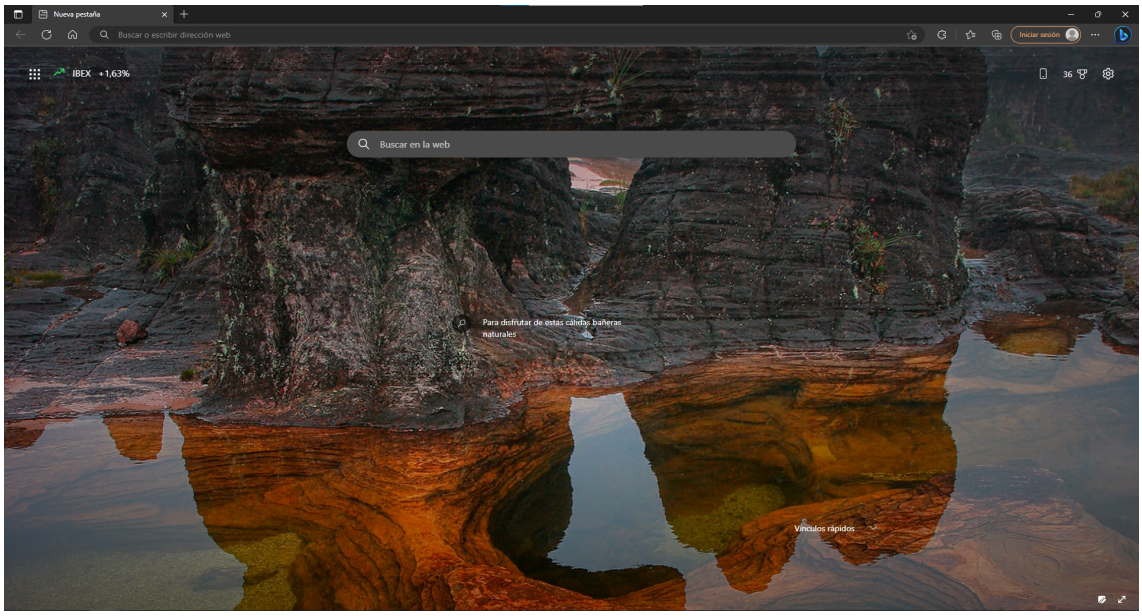


Figura 3.5: Interfaz de Microsoft Edge

También cuenta con las mismas herramientas de desarrollo que el resto de navegadores por lo que la elección del navegador es algo que simplemente queda como una elección personal del desarrollador.

3.4.3. MongoDB Compass

MongoDB Compass es una herramienta que permite acceder a la base de datos y sus colecciones a través de una interfaz gráfica que hace que sea todo más visible, se puede ver en la figura 3.6. Cuenta con diferentes funcionalidades como la posibilidad de observar gráficos y estadísticas sobre los datos almacenados en la base de datos e incluso de cada una de las colecciones.

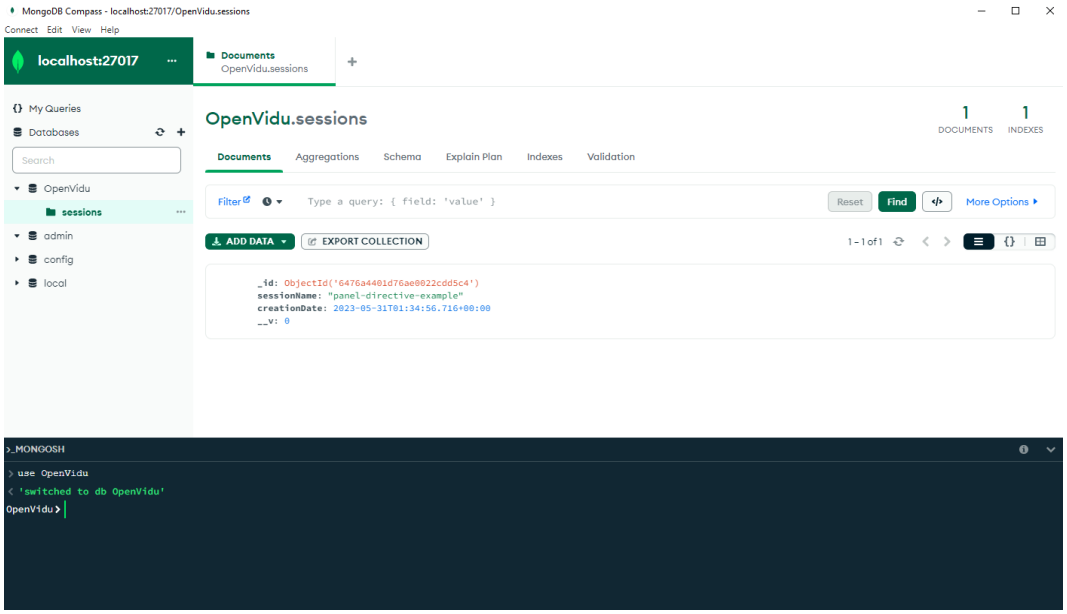


Figura 3.6: Interfaz de MongoDB Compass

Esta herramienta cuenta con su propia línea de comandos, donde se pueden realizar todo tipo de operaciones que también se pueden realizar desde la parte gráfica de la herramienta. Esta nos ha sido de gran utilidad para poder revisar de qué forma se estaban almacenando los documentos y si seguían lo descrito en el código de la aplicación.

3.4.4. Git y Github

Git es un sistema de control de versiones que es muy popularmente utilizado en el desarrollo de software. Como su definición nos indica, se utiliza para poder controlar las diferentes versiones de nuestra aplicación, para llevar a cabo esta funcionalidad se utiliza lo que se conoce como ramas, que permite dividir el software en varias versiones con la capacidad de poder unificarlas en la rama principal llamada *master* o *main*.

Github es la plataforma en la que se almacena el código fuente de las diferentes versiones de la aplicación siempre y cuando sea código abierto, aunque existe una versión de pago para poder alojar proyectos de forma privada (Yúbal Fernández, 2019). En esta plataforma, que se muestra en la figura 3.7, se puede observar de forma gráfica las diferentes ramas que se han creado a partir de la rama principal y los diferentes *commits* que se han realizado en cada una de las ramas, indicando quién la ha realizado y quién ha hecho esta operación. Como se ha indicado, esta plataforma almacena las diferentes versiones de la aplicación de manera de gráfica, lo que nos permite también acceder a estas versiones de nuestro código para poder revisar qué elementos se han eliminado en versiones futuras, también

se cuenta con la posibilidad de poder volver a esa versión de la aplicación en cualquier momento.

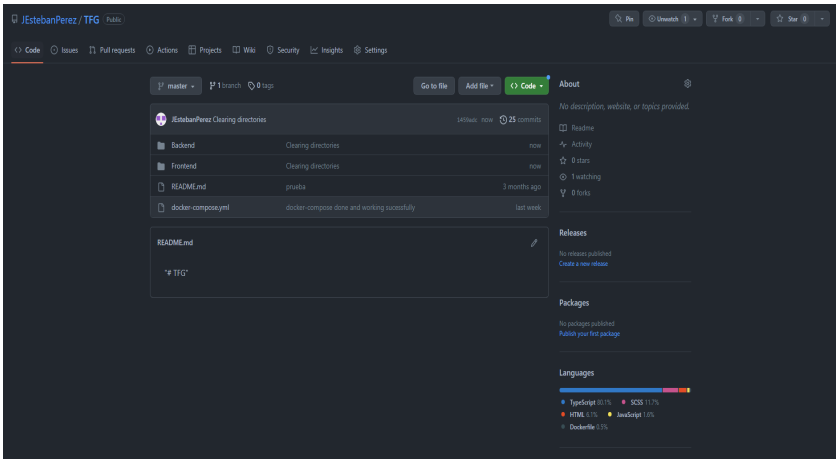


Figura 3.7: Repositorio Github del proyecto

3.4.5. Sourcetree

Esta herramienta viene acorde con la anterior, Git y Github. Sourcetree es una herramienta que permite realizar y visualizar los cambios que se están produciendo en el repositorio, muestra todos los *commits* con sus comentarios, todas las ramas que se han creado y podemos situarnos en cualquiera de estas, como se puede observar en la figura 3.8.

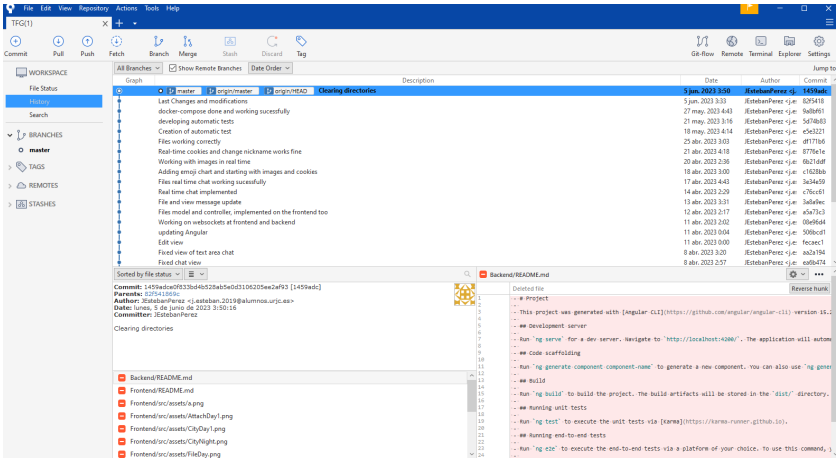


Figura 3.8: Interfaz de Sourcetree

Es una herramienta que se centra mucho en visualizar de manera gráfica todas estas funciones que en la plataforma de Github son difíciles de ver de manera tan

detallada. Nos ayuda mucho a saber en qué punto del desarrollo de la aplicación nos encontramos, además, otra de las funciones que me parecen claves en esta herramienta, es que automáticamente cuando se haya modificado un fichero, ya sea el contenido o si se ha creado/eliminado, esta herramienta te muestra que ese fichero puede ser subido ya que es distinto al de la última versión del repositorio.

3.4.6. Postman

Esta herramienta es muy útil para la creación y prueba de las diferentes peticiones HTTP. Esta herramienta te permite realizar todo tipo de peticiones HTTP, además de poder almacenarlas, y poder realizar envíos de datos en formato JSON para realizar POST y comprobar siempre la respuesta que devuelve el servidor, como se puede observar en la figura 3.9.

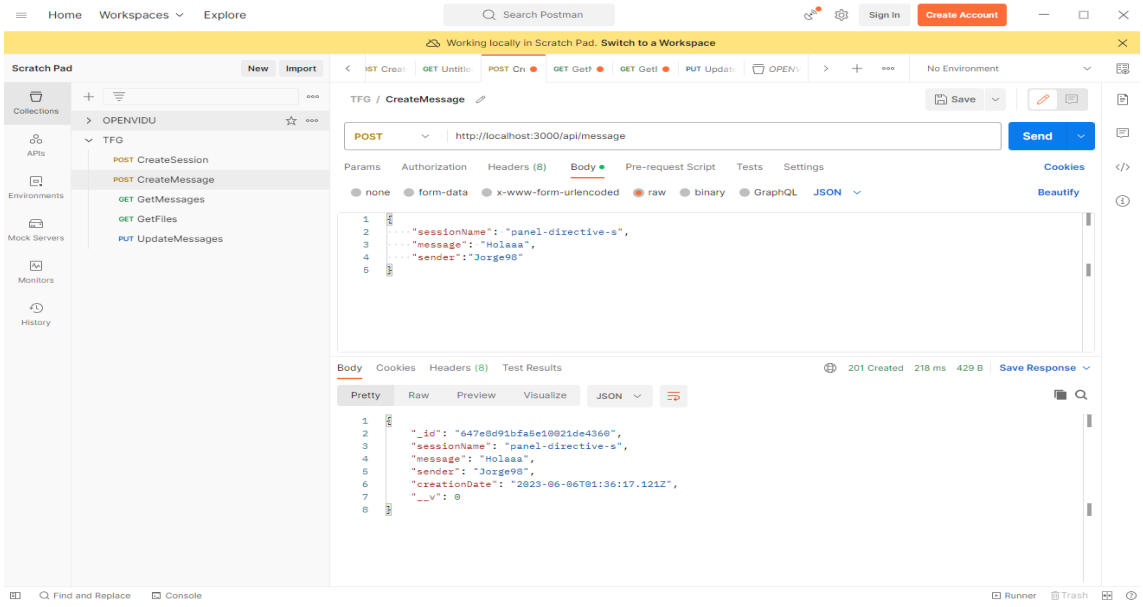


Figura 3.9: Interfaz de Postman

Esta herramienta ha sido esencial para poder determinar qué URLs son correctas, y qué tipo de peticiones hacía falta realizar para realizar las distintas funciones que queramos utilizar.

3.4.7. Docker Desktop

Docker Desktop es una herramienta que simplifica la forma de desplegar y probar las aplicaciones que están almacenadas en contenedores. Esta herramienta

nos permite poder modificar tanto la memoria y el uso de CPU límite, que van a hacer uso los contenedores que se ejecuten en el sistema.

En esta herramienta podemos observar qué imagen y qué puertos están haciendo uso los distintos contenedores que se están ejecutando, como se ve en la figura 3.10. Esta herramienta es necesaria si queremos dockerizar nuestra aplicación a la vez que poder ejecutar otros contenedores que nos pueden ser útiles durante el desarrollo de la misma.

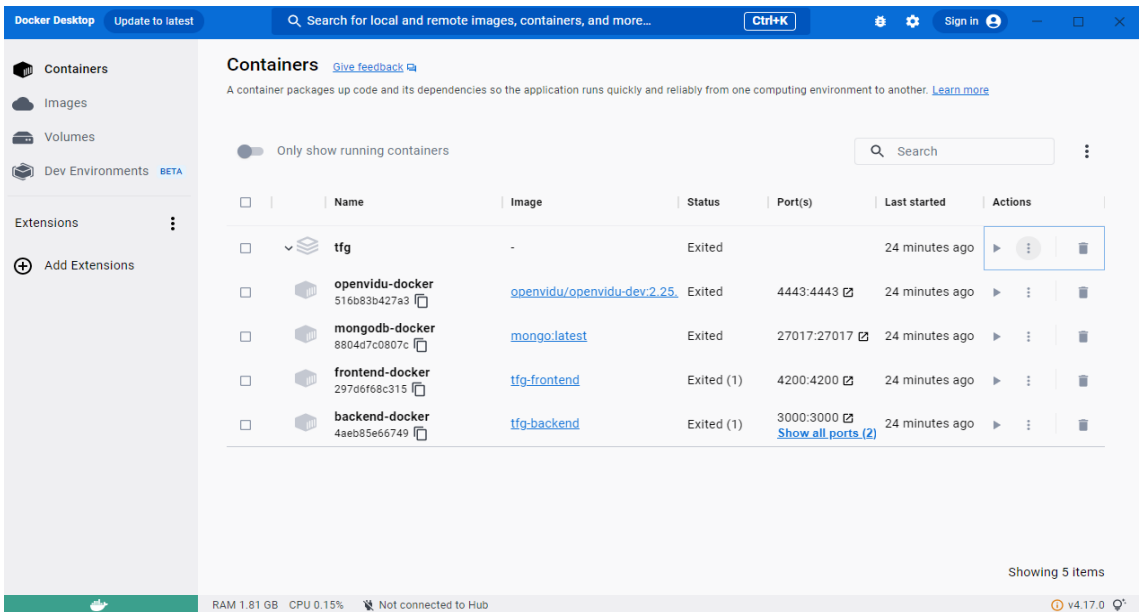


Figura 3.10: Interfaz de Docker Dekstop

3.5. Metodología

En este apartado se explicará el proceso que se ha utilizado en el desarrollo del proyecto. La aplicación se desarrolló con un modelo iterativo e incremental, partiendo de una base principal que sería lo que otorga la aplicación de OpenVidu Call. A medida que se fue desarrollado el proyecto se iban creando prototipos que iban aumentando de funcionalidades a medida que se seguía desarrollando. Empezando por la parte más interna como es el *backend* y asegurándose de que no se continuaba el desarrollo hasta completar y comprobar las funcionalidades que se iban introduciendo de manera incremental.

De manera natural se usó una metodología ágil que fue Scrum. El uso de una metodología ágil es muy popular en el desarrollo de aplicaciones, esta se caracteriza por la flexibilidad y adaptabilidad. .Este enfoque está mostrando su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad"(José,

Patricio, Mª Carmen, 2003). Dentro de las metodologías ágiles se encuentran varios tipos como XP, Kanban y Scrum. Este último, que es el que hemos aplicado para el desarrollo de nuestra aplicación, se basa en la entrega incremental por lo que se divide el trabajo en ciclos llamados *sprints* que deben tener una duración de entre 1 a 4 semanas. Se utilizó esta metodología ya que su objetivo es la obtención rápida de resultados y la facilidad que ofrece para adaptarse a cambios (Javier Garzas, 2012), esto era importante ya que podría haber cambios de requisitos con cada revisión con el tutor, lo que hacía que el proyecto estuviera en constante cambio.

Con la utilización de esta metodología se consiguió un desarrollo más fluido, a la vez que se comprendían nuevas tecnologías y herramientas que se habían utilizado en la aplicación del chat. Durante el desarrollo se utilizaron *sprints* que contenían cada uno de ellos unos objetivos que se tenían que cumplir en un plazo más o menos relativo debido a otras tareas externas. Estos *sprints* se pueden agrupar en diferentes fases que son las siguientes:

- Fase 1: Comprensión de las aplicaciones (25 días)

    1. Se estudió la documentación de las aplicaciones de las que se iba a partir para poder determinar qué funcionalidades serían útiles para el desarrollo de la nueva aplicación, qué elementos son esenciales para poder implementar ciertas funcionalidades y cómo funcionan internamente ambas aplicaciones.
    2. Se probaron ambas aplicaciones por separador para concretar qué versiones de NPM y de Angular estaban utilizando, además de probar las interfaces.

- Fase 2: Desarrollo de la aplicación (70 días)

    1. Se empezó desarrollando el *backend* en NestJs, se tuvo que transcribir los ficheros que utilizan en el *backend* de OpenVidu Call que estaban desarrollados en JavaScript a TypeScript para que, fijándonos en el servidor de la aplicación de chat, pudiéramos desarrollar un servidor unificado de las dos aplicaciones con lo esencial para que funcionara correctamente.
    2. Se implemento la base de datos MongoDB, pero en local a diferencia de la base de datos de la aplicación de chat, por lo que se tuvo que hacer uso de un contenedor Docker para su implementación y de la librería Mongoose.
    3. Utilizando como base para el *frontend* los archivos de Angular que otorga la aplicación de OpenVidu Call, se implementó las conexiones con el *backend* para que estuvieran comunicadas en todo momento, y se implementó las funcionalidades importantes de la aplicación del

chat, se adaptarón todas las funcionalidades para que se utilizarán servicios que ya tenían implementadas OpenVidu Call.

4. Se desplegó la aplicación con todas sus partes, *backend*, *frontend*, MongoDB. coordinados por un mismo Docker Compose para que simplemente con un comando se pueda ejecutar toda la aplicación sin necesidad de instalar diferentes herramientas y tecnologías que con Docker no son necesarias.

- Fase 3: Creación de nuevas funcionalidades (20 días)

  1. Como la aplicación de chat cuenta con unas funcionalidades que la aplicación de OpenVidu Call no tiene, se tuvo que investigar distintas funcionalidades que nos podrían ser útiles y que se podían implementar en la parte de *frontend* para poder suplir distintos casos que no era viables en la aplicación.

- Fase 4: Correcciones visuales (15 días)

  1. Se ajustaron distintos elementos del HTML, y sus respectivas hojas de estilos, para que estuvieran en sintonía con toda la aplicación y no fueran molestas para los usuarios. También se modificaron tamaños y tamaños de la tipografía acorde a la opinión de los usuarios.

4

Capítulo 4: Descripción Informática

En este nuevo capítulo se detallará todo lo realizado en el proyecto empezando por los requisitos establecidos, la arquitectura utilizada, su diseño e implementación, la interfaz del usuario, los test y, por último, la distribución y despliegue.

4.1. Requisitos

Como hemos explicado antes, en el apartado de metodología, nuestra aplicación se divide en fases por lo que vamos a explicar los requisitos dividiéndolos por las distintas fases. Empezando por la segunda fase, desarrollo de la aplicación, ya que la primera fase simplemente es un estudio de las dos aplicaciones de las que partimos.

Los requisitos se pueden dividir en funcionales y no funcionales. Los requisitos funcionales representan lo que el sistema debe ser y lo que debe hacer, y los requisitos no funcionales describen los límites y las restricciones que deben tener el sistema. ([¿Qué son los requisitos funcionales y no funcionales?], s.f.)

4.1.1. Fase 2: Desarrollo de la aplicación

Identificador	Nombre	Descripción
RNF-001	Selección de puertos	Se determinan los puertos por los que se van a comunicar las aplicaciones, tanto la BBDD, el *frontend* y el *backend*.
RF-001	Almacenamiento BBDD	Enlazamos la base de datos al servidor para poder almacenar los datos.
RF-002	Creación sesión servidor	La primera vez que un usuario se conecta a una nueva sesión, esta se envía a la base de datos para guardar esta sesión
RF-003	Envio de mensajes servidor	Cada vez que un usuario envía un mensaje por el chat, este mensaje se transmite a la base de datos para su almacenamiento.
RF-004	Recepción de mensajes servidor	Cuando un participante se conecta a la sesión, se pedirá al servidor todos los mensajes anteriores a su conexión.
RF-005	Botones del chat	En el apartado del chat, aparecerán un botón que será para adjuntar ficheros, otro botón que será para insertar algún emoticono, y el último, que sirve para enviar el mensaje escrito.
RF-006	Área de texto	En el apartado del chat, entre el botón de emoticonos y de envío de mensaje, aparecerá un área de texto donde podremos escribir el mensaje que enviaremos en el chat.

RF-007	Posición de mensajes	En el apartado de chat, se mostrarán en el lado izquierdo los mensajes enviados por otros usuarios y a la derecha los mensajes enviados por el usuario.
RF-008	Envío de ficheros	La aplicación permitirá el envío de cualquier tipo de ficheros, se enviará a todos los participantes y se guardará en la base de datos
RF-009	Abrir ficheros	Se permitirá clicar cualquier fichero que haya sido enviado, dependiendo del tipo del archivo, se permitirá visualizar en una nueva ventana o requerirá su descarga.
RF-010	Menú emoticonos	La aplicación permite el envío de emoticonos, que se podrán elegir desde una ventana.
RF-011	Último mensaje	La primera vez que se entre al chat, este enfocará directamente al último mensaje que se ha enviado en el chat.

Tabla 4.1: Tabla de requisitos de la fase desarrollo

4.1.2. Fase 3: Creación de nuevas funcionalidades

Identificador	Nombre	Descripción
RNF-001	Identificación mediante cookies	La aplicación carece de un sistema de autenticación por lo que no se puede identificar al usuario que entra en la aplicación, entonces se ha decidido utilizar la tecnología de cookies para la identificación en las siguientes visitas.

RNF-002	Uso de *Signals*	la aplicación de OpenVidu Call cuenta con su propia gestión de *sockets* mediante el uso de *signals*, que nos permite enviar información a los usuarios conectados a la aplicación. En la aplicación se utilizará para el envío de información en tiempo real.
RF-001	Cambio de nombre	En la aplicación de OpenVidu Call se permite el cambio de nombre, por lo que se ha implementado una función pensando en el chat. Cuando un usuario cambia de nombre, todos los mensajes tanto en el chat como en la base de datos también cambian el nombre del emisor.

Tabla 4.2: Tabla de requisitos de la creación de nuevas funcionalidades

4.1.3. Fase 4: Correcciones visuales

Identificador	Nombre	Descripción
RF-001	Colores de los mensajes	En el chat, se realiza una diferenciación de los colores entre los mensajes de otros participantes y de los mensajes del usuario.
RF-002	Tamaño del texto de los mensajes	Se han ajustado el tamaño del texto de los mensajes para cumplir con el gusto del usuario.
RF-003	Tamaño del cuadro de mensaje	Se ha ajustado el tamaño de la caja para que se ajuste adecuadamente al espacio del chat.

Tabla 4.3: Tabla de requisitos de correcciones visuales

4.2. Arquitectura y Análisis

La arquitectura es una parte fundamental de cualquier desarrollo, ya que permite diseñar la estructura que va a tener el *software* y las herramientas que vamos a utilizar antes de que se empiece a desarrollar. (Pablo Huet, 2022)

En este apartado se desarrollarán los detalles de alto nivel de la aplicación a través del uso de diagramas. El primer diagrama, que veremos en la figura 4.1, representa la arquitectura de la aplicación y cómo se relaciona el *frontend*, que es Angular Client, con el *backend*, que se representa con NestJs Server. Este a su vez se relaciona directamente con el servidor de OpenVidu y con la base de datos.

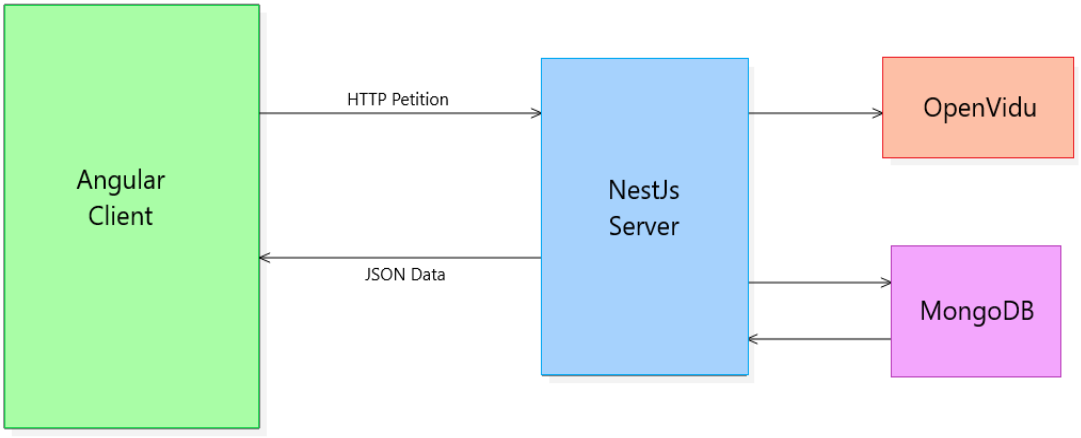


Figura 4.1: Arquitectura de la aplicación

El diagrama 4.2 representa las clases y vistas del *front* de la aplicación. Como esta desarrollado en Angular, implementa un patrón parecido a MVC, aunque si comparte algunos conceptos y principios. En este caso la vista se relaciona con los distintos componentes de la aplicación.

En el *frontend* de la aplicación hay componentes ocultos internamente, que son proporcionados la propia aplicación de OpenVidu Call, por lo que no se mostrará en el diagrama, pero realizan una función importante dentro de la aplicación.

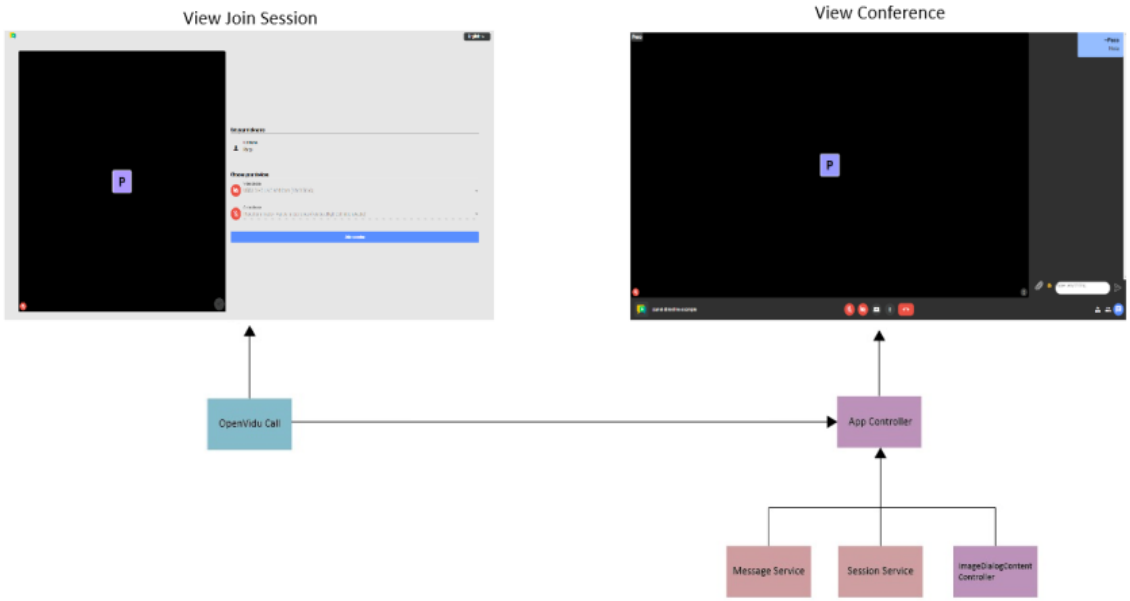


Figura 4.2: Diagrama de clases y vistas

Como se ve en el diagrama, el componente de OpenVidu Call cuenta con vistas propias que utiliza sirve también de base para el controlador App. En nuestra aplicación el controlador principal es App donde se realizan todas las funciones y todo lo necesario para que el *software* cuente con todas las características.

Uno de los problemas de lo comentado en el párrafo anterior es que, por ejemplo, la vista Join Session, no puede ser modificada, para ello habría que mirar en los ficheros internos de la aplicación de OpenVidu Call, pero no existe documentación para este tipo de archivos por lo que lo hace demasiado complicado poder modificar algo de esa vista, lo mismo sucede prácticamente en la vista Conference, aunque esta es más editable, hay varios elementos que no se pueden modificar.

El siguiente diagrama que se explicará será el de clases de la parte de *backend*, como se observa en la figura 4.3. En este diagrama contamos con los controladores, los servicios y los modelos que hemos utilizado para que haya una comunicación completa, tanto con el *frontend* como con las aplicaciones de OpenVidu y de MongoDB. Se explicará cada uno de los controladores junto con las peticiones que contienen.

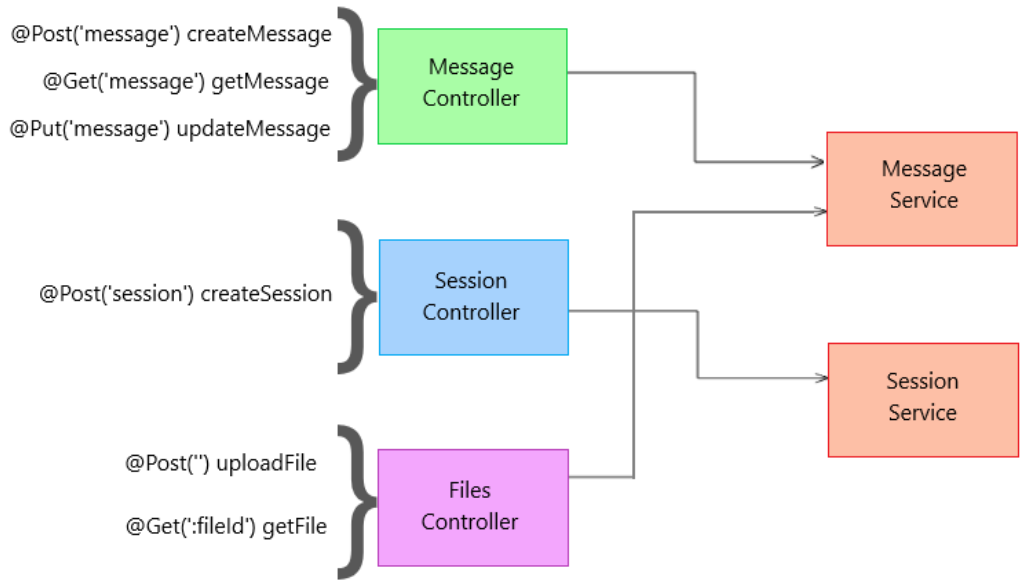


Figura 4.3: Arquitectura del servidor

En el diagrama se reflejan las peticiones que se realizan al servidor para la transmisión de información, estas peticiones permiten un flujo constante de datos desde la base de datos hasta el *frontend*.

Toda la arquitectura de la parte del servidor se fundamenta en los servicios, que son los que nos permiten la comunicación. Estos servicios nos permiten conectar con las colecciones de la base de datos donde almacenaremos los datos.

Como se ve en el diagrama el controlador de mensajes, es el que relaciona directamente los mensajes con la colección *messages*. Este controlador es el que manejará todo lo relacionado con el chat en vivo de la aplicación. Como no existe un sistema de gestión de usuarios, la petición @Put nos permite gestionar los cambios de nombre, además cuenta con otras peticiones que son: la petición de tipo @Post que nos va a permitir almacenar el dato en la base de datos y el tipo @Get que nos permitirá recoger todos los mensajes de una sesión. Este controlador hace uso de datos relacionados de la sesiones, pero no es necesario la utilización del servicio.

El servicio Messages, es el que más importancia tiene en la aplicación, ya que, al ser una aplicación de videoconferencia con chat incluido, es una parte fundamental de esta. Este servicio a la vez es el más utilizado ya que tiene constante comunicación con la base de datos.

El controlador Files maneja todo lo relacionado con ficheros. Aunque esté en otro controlador distinto al de mensajería, este hace uso del servicio Messages,

al tratarse también de un mensaje. Por lo que los ficheros se almacenarán en la base de datos dentro de la colección *messages*. Este controlador cuenta con dos peticiones HTTP: la primera, de tipo @Post nos permitirá almacenar el fichero como un tipo Message y también en la misma colección de la base de datos y la segunda, de tipo @Get, nos permitirá recoger un fichero específico de la base de datos.

Por último, se encuentra el controlador de sesiones, que hace uso del servicio Session. Este controlador maneja lo relacionado con la sesión, en este caso, la creación de ésta. De momento no cuenta con mucha funcionalidad debido a que faltan algunas características, pero se vio importante su utilización para futuras implementaciones. Este controlador solo cuenta con una petición de tipo @Post que nos permitirá almacenar en la base de datos una nueva sesión.

A continuación, se explicará la secuencia que sigue un mensaje en la aplicación en detalle. En el diagrama 4.4 se refleja el funcionamiento de la comunicación desde el *frontend* hasta la base de datos. Primero el mensaje sin salir del *frontend* se envía a todos los participantes que estén activos, mientras, al mismo tiempo, se envía al *backend* mediante la petición HTTP de tipo @Post. Con esta petición se comunica con el controlador Message y éste a su vez, envía el mensaje a la base de datos para ser almacenado siguiendo el Schema de Message, que se explicará posteriormente. Todos estos procesos se realizan mediante el servicio Message.

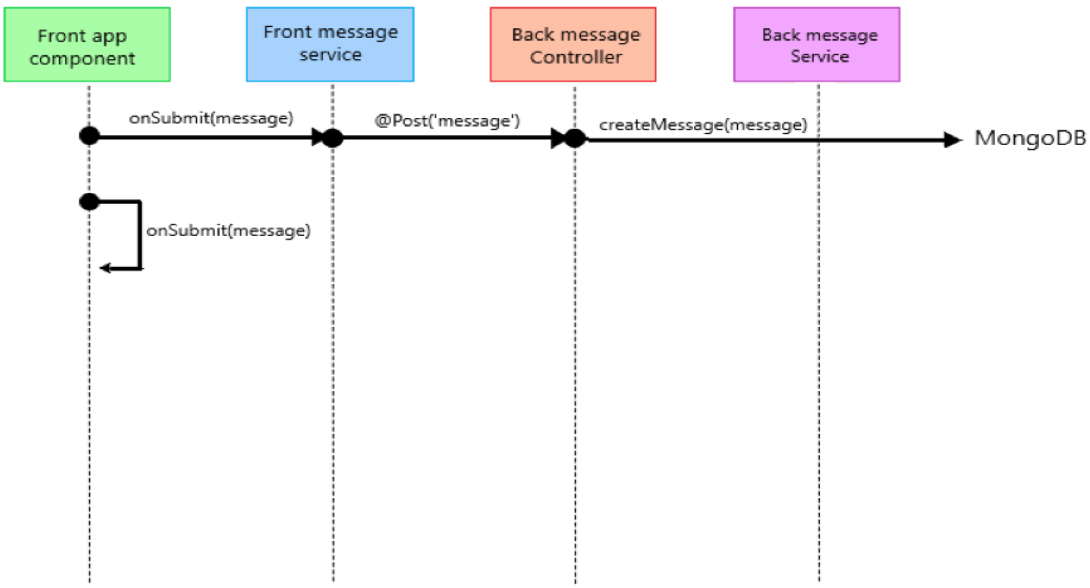


Figura 4.4: Secuencia del envío de un mensaje

Explicando el diagrama, el mensaje sale desde el componente app del *frontend*,

utilizando el servicio de Message del *front*, mediante una petición HTTP de tipo @Post que recogerá el controlador de Message del *Backend*. Este controlador al recibir una petición de tipo @Post lo que hace es, usando el servicio de Message del *backend*, un guardado en la base de datos de este mensaje.

Nos podemos dar cuenta, en el diagrama 4.4, como el método onSubmit se retroalimenta, esto es debido a que el *socket* está dentro del mismo *front* por lo que cuando se ejecuta ese método, se está enviado el mensaje a todos los participantes de la videoconferencia. Esto nos permite reducir en gastos de recursos ya que OpenVidu Call cuenta con esta función y hace el trabajo internamente por lo que no afecta al resto de la aplicación.

Gracias a este tipo de funciones, OpenVidu Call nos facilita una gran cantidad de tareas que nos harían ampliar nuestro software y hacer que, fuera más dependiente a otras librerías o tecnologías y que pudiera ocasionar que haya problemas, con desfases de versiones o incluso problemas con otras librerías. Aunque, solo puede ser desarrollado en el *frontend* por lo que no permite mucha versatilidad y la modificación es muy limitada, así que no contaríamos con mucha escalabilidad. Otro de los grandes problemas es que no cuenta con documentación lo que hace que sea más complicado conocerla al cien por cien.

El diagrama es un ejemplo de un envío de mensaje, también parecido al diagrama que sería el envío de ficheros, por lo que con este diagrama es válido para ejemplificar la funcionalidad de la aplicación entera. Este diagrama abarca desde el envío en el *frontend* hasta su almacenamiento en la base de datos.

Con esta implementación ninguno de los componentes cuenta con una gran complejidad por lo que la trazabilidad del código es bastante simple e intuitiva.

A continuación, se hablará del diagrama 4.5 que representa las entidades y relaciones que contiene la aplicación.

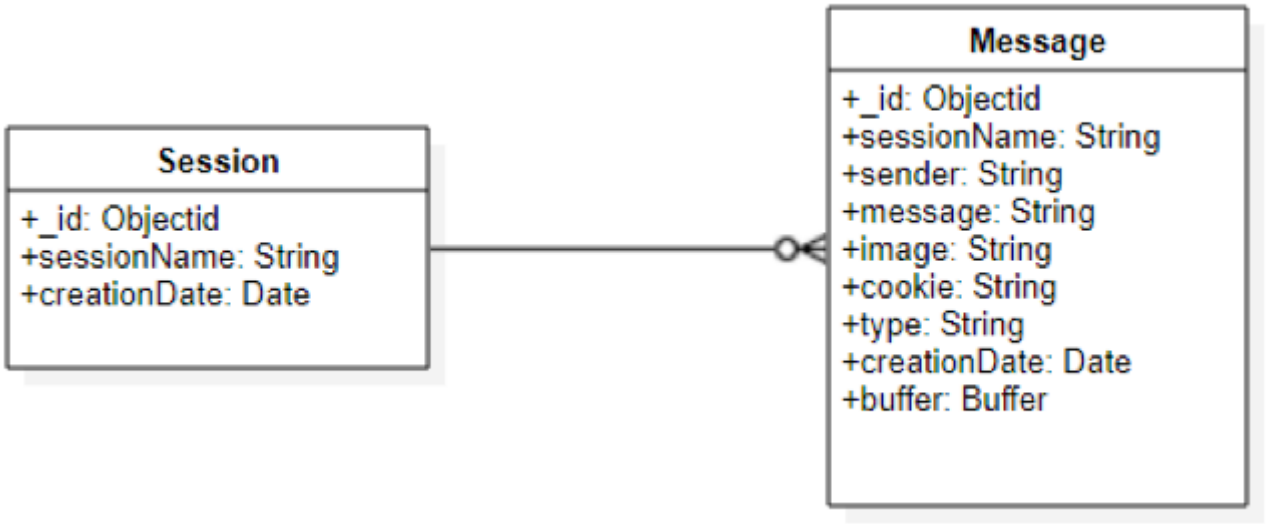


Figura 4.5: Diagrama Entidad Relación Base de datos

La única relación que existe en este diagrama y que se puede observar en el Schema, es de Session-Message. Esta relación es de 1 Session a n Message opcional, esto permite que exista una sesión, pero puede no tener mensajes, sin embargo, no puede existir un mensaje si no existe una sesión. Uno de los problemas es que, al ser una base de datos no relacional, no se pueden hacer borrados en cascada, por lo que, si se borra la sesión, seguirían existiendo los mensajes, lo que no tendría sentido y quedaría una base de datos descuadrada. Algo que se debería solucionar en futuras evoluciones de la aplicación.

Gracias al uso de una base de datos no relacional, esta nos permite contener estructuras polimórficas incluso dentro de la misma colección. Esta característica nos ha servido de utilidad para nuestra aplicación debido a que, tenemos un caso en el que los mensajes alteran este Schema, este caso es el de los envíos de ficheros.

El Schema que tienen los mensajes es el que aparece en el diagrama 4.5, pero hay diferencia a la información que se envía a la base de datos ya que hay cierto tipo de mensajes que no usa algún campo específicamente, por lo que no se envía. Los mensajes de texto normal utilizan todos los campos salvo el de buffer, que se usa exclusivamente para el envío de ficheros, por lo que con esta característica podemos guardar todos los documentos bajo la misma colección sin tener que forzar a rellenar campos o incluso a tener que usar varias tablas distintas para el mismo tipo de información.

Tanto los Schemas como las colecciones están en una fase temprana. Por lo que usando este tipo de base de datos nos permite que, en futuras implementaciones, no sea necesario tener que estar cambiando los esquemas de la base de datos, esto

hace que sea mucho más versátil.

4.3. Diseño e implementación

Aquí se explicará los algoritmos y decisiones de diseño que se han tomado durante el desarrollo de la aplicación.

4.3.1. Decisiones de diseño

Vamos a empezar hablando del apartado gráfico de la web de la aplicación. Aquí surgen varios puntos que se tratarán a continuación:

- El primer punto y más importante, es que al emplearse de base la aplicación OpenVidu Call, tenemos bastante limitado las modificaciones que podemos realizar. Porque, para empezar, hay elementos que no tenemos posibilidad de modificación, ya sea por falta de documentación o porque directamente no podemos acceder porque son elementos internos con su propia hoja de estilos.
- El segundo punto, es que, al realizar la unión de la aplicación de chat, se vio que su apartado gráfico es bastante completo y útil por lo que no fue necesario realizar algo desde cero, además, se suma que realmente estamos haciendo una implementación del chat en la aplicación de OpenVidu Call, por lo que realmente tenía sentido mantener su estilo propio, salvo la realización de pequeñas modificaciones para ajustarse al resto de la aplicación y que fuera más homogéneo.
- El tercer punto, dentro de las pequeñas modificaciones para que fuera una aplicación homogénea, surge la intención de que los usuarios de distintas aplicaciones de videoconferencias tuvieran un peso importante en el diseño de la aplicación. Con esta intención se realizó una encuesta para que, sobre los resultados, se pueda hacer una interfaz más atractiva e intuitiva para ellos.

Empezando por el primer punto, y como se puede observar en la figura 4.2. Los elementos principales de las vistas con las que cuenta la aplicación, y directamente la vista de Join Session, no pueden ser alteradas de prácticamente ninguna forma por lo que tenemos que trabajar sobre ellas. Los elementos que podemos modificar son escasos y uno de ellos es el apartado de chat, accesible desde el botón de abajo a la derecha que se observa en la figura 1.2, que nos despliega un espacio que es donde se representará el chat. Aquí entra justo el segundo punto que hemos

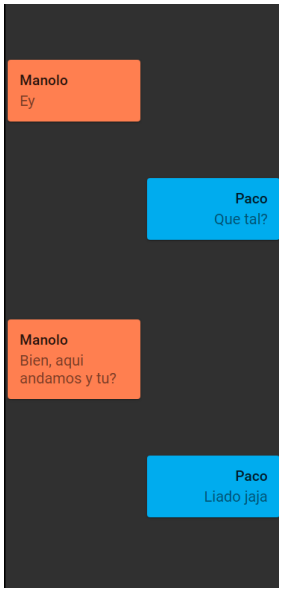
tratado, como se ha comentado, la intencionalidad de la aplicación es la unión de las distintas aplicaciones en una, como la base es OpenVidu Call y contando que no se puede modificar apenas, la parte de la aplicación de chat ha tenido que ser modificada para mantener un estilo unificado. Partiendo de lo que se tenía ya previamente en la aplicación de chat, se ha ido adaptando con el resto de elementos, por ello se ha utilizado la misma hoja de estilos pero depurándola, ya que contiene estilos para los elementos que no se utilizan, a la vez que se iba ajustando para la aplicación. Básicamente los elementos más modificados son los que se mostrarán en las figuras 4.6

Figura 4.6: Elementos de la hoja de estilos

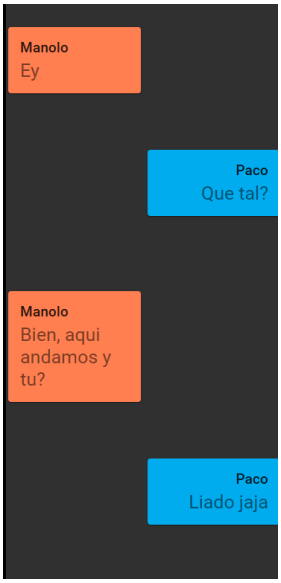
Algunas de las modificaciones en estos elementos vienen de una encuesta realizada a diferentes usuarios, como se ha comentado en el tercer punto. La encuesta fue compartida a través de grupos de WhatsApp y por Instagram para lograr un mayor número de respuestas. En la encuesta se tratan los siguientes puntos: tamaño y tipografía del texto en los mensajes y los colores utilizados para la distinción de los mensajes entre los del usuario y los del resto de participantes.

La primera pregunta se ofreció una serie de imágenes con diferentes tamaño de letra y su visualización, donde cada uno de los encuestados tenía que elegir entre que tamaño les parecía el adecuado y si tenían alguna aportación en cuanto a tipografía o alguna observación respecto a la letra que no se representaba en las imágenes, que se muestran en la figura 4.7. De esta última cuestión, apenas existen respuestas y ninguna compartía la misma opinión por lo que apenas se

tuvo en cuenta para su alteración en la aplicación.



(a) Primera opción



(b) Segunda opción

(c) Tercera opción

(d) Cuarta opción

Figura 4.7: Pregunta del tamaño de la letra de la encuesta

La segunda y última cuestión es relacionada con el color utilizado en el chat para distinguir los mensajes correctamente, en esta cuestión hubo una gran variedad de opiniones, aunque destacó más una de ellas y por ende se tuvo en cuenta para la versión final de la aplicación. La opinión fue que los colores utilizados en un principio eran demasiados fuertes o saturados y también llamativos, por lo que desentonaba demasiado con el resto de la aplicación además de ser molestos

para la vista, y que se optará por bajar la tonalidad o utilizar colores pastel que son más agradables a la vista y hacían que visualmente fuera más homogénea la interfaz.

En la encuesta participaron un total de 35 personas. En la primera cuestión respondieron todos los participantes dando como resultado el gráfico que aparece en la figura 4.8. Hubo dos respuestas que obtuvieron el mismo porcentaje, que al tener un tamaño de letra bastante parecido se optó por utilizar una cantidad intermedia.

Figura 4.8: Resultados primera pregunta de la encuesta

Además, se realizó un cambio adicional para los nombres de los usuarios que envían un mensaje, a estos se les añadiría un símbolo para que se pueda diferenciar del texto que se envía. Este símbolo, es utilizado también por la aplicación WhatsApp, es -". Con este cambio, junto con otras modificaciones, se haría más fácil poder distinguir el texto, del usuario que lo manda.

A la hora de implementar la parte del *frontend*, la aplicación de chat contaba con un sistema que utilizaba Web Sockets, desarrollado de forma propia, que tuvimos que adaptar para su implementación con OpenVidu Call. Por lo que mientras se probaban las aplicaciones por separado y se miraba la documentación, se descubrió que OpenVidu Call contaba con un sistema interno de Web Sockets que nos podría servir, llamado *signals*. Además, al estar ya integrado en la aplicación no era necesario desarrollarlo por nuestra parte, también servía para realizar otras funcionalidades que serían útiles, tanto para el desarrollo como para futuras implementaciones.

Los canales de los *signals* se inicializan cuando se conecta el usuario a la sesión como aparece en la figura 4.9. En ella se puede observar cómo cada vez

que se conecta a una sesión, el usuario se conecta a los canales de CambioNombre, CHAT y File. Esto funciona como los *subscribe*, cuando llegue una notificación nueva entrará a realizar el código, que depende de qué tipo de señal llegue.

```
onSessionCreated(session: Session) {
    this.session = session;


    this.session.on(`signal:CambioNombre`, (event: any) => {
        const e = JSON.parse(event.data);
        this.messages.forEach(m=>{
            if(m.cookie== e.cookie){
                m.sender= e.nickname;
            }
        })

    });

    this.session.on(`signal:${Signal.CHAT}`, (event: any) => {
        const msg = JSON.parse(event.data);
        if(msg.sender != this.user && msg.type=="message"){
            this.messages.push(msg);
        }
    });

    this.session.on(`signal:File`, (event: any) => {
        const msg = JSON.parse(event.data);
        this.messageService.getFile(msg.id).subscribe(f => {

            if(f.message.cookie != this.cookie){
                this.formatImage(f.message);
                this.messages.push(f.message);
            }
        },
        error => console.log(error))


    });
```

Figura 4.9: Inicializar los distintos canales Signal

La señal de CambioNombre envía una notificación utilizando este canal, cuando un usuario ha realizado un cambio en su nombre, se cambiará el nombre de todos los mensajes que ha enviado el usuario. Esta notificación se realiza mediante el código que aparece en el método **changeUserName**. En este método además de enviar la notificación, envía una petición a la base de datos, como se ve en la figura 4.10, para cambiar el usuario en los mensajes almacenados en él.

```
changeUserName(){

  let data ={cookie: this.cookie, sender: this.user! };
  this.messageService.updateMessage(data).subscribe(
      message => console.log("Éxito") ,
  error => console.error(error)
  )

  const signalOptions: SignalOptions = {
      data: JSON.stringify({ nickname: this.user, cookie:this.cookie}),
      type: "CambioNombre",
      to: undefined,
  };
  this.session.signal(signalOptions);
}
```

Figura 4.10: Código que maneja los cambios de nombre

La señal CHAT, como su nombre indica, sirve para cuando un usuario ha enviado un mensaje al chat, este se envíe a todos los usuarios conectados, para que les aparezca en su respectivo chat. Este proceso se realiza en el método **onSubmit**, que también realiza el envío del mensaje a la base de datos.

La señal File, se utiliza cuando el usuario envía un fichero, este envía a todos los usuarios el dato con el mismo formato que se envía a la base de datos, para después ser incluido en el chat.

Como la aplicación no cuenta con un sistema de gestión de usuarios, se optó por el uso de cookies de navegador, para identificar al mismo usuario si este sufre algún percance o para identificar cuando cambia de nombre. Aunque no es una solución final, sirve para poder solucionar diferentes conflictos que estaban surgiendo a medida que avanzaba el desarrollo.

```
setCookie(name: string, value: string, days: number) {
    let expires = '';
    if (days) {
      const date = new Date();
      date.setTime(date.getTime() + (days * 24 * 60 * 60 * 1000));
      expires = '; expires=' + date.toUTCString();
    }
    document.cookie = name + '=' + value + expires + '; path=/';
  }

  getCookie(name: string) {
    const cookies = document.cookie.split(';');
    for (let i = 0; i < cookies.length; i++) {
      const cookie = cookies[i].trim();
      if (cookie.startsWith(name + '=')) {
        return cookie.substring(name.length + 1);
      }
    }
    return "";
  }

checkCookie(){
    //Creador de Cookies
    this.cookie =this.getCookie("id");
    if(this.cookie==""){
        this.setCookie("id",Math.floor(Math.random() * 100000).toString(),2)
        this.cookie =this.getCookie("id");
    }
}

async ngOnInit() {
    this.tokens = {
        webcam: await this.getToken(),
        screen: await this.getToken(),
    };


    this.checkCookie();


}
```

Figura 4.11: Creador y verificador de cookies

Siempre que un usuario entre en la aplicación se comprueba si el usuario ya cuenta con una cookie en su navegador con el método **checkCookie** y **getCookie**, que aparece en 4.11, y si no cuenta con ella, se crea una desde cero utilizando el método **setCookie** que nos permitirá insertar una cookie en el navegador de cada usuario.

La aplicación cuenta con la característica de que al momento en el que un usuario se conecta al chat de una sesión, que ya estaba iniciada, se cargan todos los mensajes anteriores a su conexión. Esto puede dar lugar a un inconveniente y es que, si se han enviado muchos mensajes, el usuario tendrá que bajar hasta el último mensaje para seguir la conversación, por lo que se ha realizado una funcionalidad en el método **startChat**, como se ve en la figura 4.12. Este método lo que hace es ir directamente al último mensaje de chat.

```
startChat(){
  //Para focusear siempre el ultimo mensaje en el chat
  let upperElement = document.querySelector(".upper") as HTMLElement;
  let lastChild = upperElement.lastElementChild as HTMLElement;
  lastChild.scrollIntoView();
}
```

Figura 4.12: Código para enfocar el último mensaje del chat

En la parte del *backend*, partíamos de los archivos con los que cuenta OpenVidu Call y con la idea que uso la aplicación de chat, el problema era que OpenVidu Call utilizaba archivos desarrollados en JavaScript, por lo que se decidió realizar el *backend* de la misma forma que el *frontend,* usando Angular y TypeScript por lo que siguiendo la estructura del *backend* de la aplicación de chat, se transcribió los ficheros de OpenVidu Call y se desarrolló todo en TypeScript. Así no tendríamos que mezclar ni lenguajes ni tecnologías para poder ejecutar la aplicación en su conjunto.

A diferencia con la aplicación de chat, que usa una base de datos MongoDB Atlas que es una base de datos de MongoDB en la nube. En nuestra aplicación decidimos cambiar esta base de datos online a una local, a partir de un contenedor Docker, aunque siguiera siendo una base de datos MongoDB.

4.3.2. Problemas encontrados en el desarrollo

En todo desarrollo de un *software* existen problemas o dificultades que ralentizan su progreso y que requieren de una gran cantidad de tiempo y de investigación. Estos problemas pueden surgir por desarrollar nuevas funcionalidades, refactorizaciones o compatibilidades con otras tecnologías.

El mayor problema que ha surgido durante el desarrollo ha sido durante la creación y configuraciones del Docker Compose. Al hacer Docker Compose se crean contenedores de todas las tecnologías necesarias, que son: *backend*, *frontend*, OpenVidu y MongoDB. Para crear estos contenedores se ejecuta el fichero **docker-compose.yaml**, este fichero contiene el siguiente código 4.13

```yaml
version: "3.9"

services:
  backend:
    build: ./backend
    ports:
      - "3000:3000"
      - "5000:5000"
    container_name: backend-docker
    depends_on:
      - mongodb
      - openvidu
    networks:
      - mynetwork

  mongodb:
    image: mongo:latest
    container_name: mongodb-docker
    ports:
      - "27017:27017"
    networks:
      - mynetwork

  openvidu:
    image: openvidu/openvidu-dev:2.25.0
    container_name: openvidu-docker
    ports:
      - "4443:4443"
    environment:
      - OPENVIDU_SECRET=MY_SECRET
    networks:
      - mynetwork

  frontend:
    build: ./frontend
    ports:
      - "4200:4200"
    container_name: frontend-docker
    networks:
      - mynetwork

networks:
  mynetwork:
    driver: bridge
```

Figura 4.13: Fichero docker-compose

En el fichero se define qué imagen se usará en cada uno de los contenedores junto con sus respectivos nombres y puertos que usarán. El primer problema de esto es que entre ellos se conectan haciendo uso de los nombres del contenedor, pero no hay nada que indique en qué enlaces se deben usar, por ejemplo, para conectar el *backend* con la base de datos se usa un enlace que es **mongodb://mongodb:27017**, donde se usa el nombre del contenedor en este caso **mongodb** en vez de **localhost**, como se usaba previamente. Sin embargo, para otros enlaces esto no afecta, por ejemplo, en el fichero **app.component** del *frontend*, que es donde se almacena toda la funcionalidad de la aplicación, se conecta al *backend* de OpenVidu pero sigue utilizando este enlace **http://localhost:5000/** , y no usa el dominio **openvidu** que es el nombre del contenedor. La falta de documentación y que apenas hubiera otros desarrolladores con el mismo problema hizo que el tiempo demorado para poder solucionar los errores y que funcionará todo fuera mucho mayor del esperado.

Al final, tras muchos intentos y pruebas con diferentes enlaces, se consiguió arreglar los problemas y que la aplicación funcionará correctamente.

Otro de los problemas importantes, fue el poder controlar el cambio de nombre del usuario cada vez que quisiera. Este problema surge porque, como se ha comentado anteriormente, hay algunos elementos a los que no se tiene la capacidad de modificación, entre ellas la vista de Join Session y algunos elementos de la vista Conference, el usuario puede modificar su nombre cada vez que entra en una sesión o cuando quiera una vez este dentro, esto sumado a que la aplicación no cuenta con un sistema de gestión de usuario, hacía que fuera complicado poder

controlar qué usuario cambiaba de nombre, si el usuario que entraba en la sesión era el mismo que había entrado anteriormente o incluso, un problema mayor, que era que cualquier usuario podía cambiarse el nombre y ponerse el mismo que otro usuario, esto modificaba la forma en la que veía el chat, además de que el usuario que tenía el nombre anteriormente vería cómo se envían mensajes que él no ha escrito.

Para solucionar este problema, se llegó a la conclusión de que había que implementar el uso de cookies para poder identificar a los usuarios cada vez que se conectaban, como se mencionó en el apartado anterior de decisiones de diseño, aunque es una solución temporal ya que la gestión de usuario debería realizarse de otra forma, y el sistema de cookies puede ser útil para futuras implementaciones que se comentará más adelante.

A pesar de todos estos problemas, la parte de desarrollo que más ha generado dificultades, aunque no con mucha complejidad, ha sido el apartado gráfico. Como ya se sabe, la parte de desarrollo de HTML y de la hoja de estilos puede ser bastante traicionera. No han sido problemas muy difíciles de solucionar, pero si han sido bastante molestas.

El último problema va relacionado con la siguiente sección que comentaremos, que son las pruebas. El problema tiene origen a la hora de desarrollar y ejecutar, principalmente por la falta de conocimiento relacionado con el desarrollo de pruebas en TypeScript y en Angular. Las pruebas se pueden ejecutar con diferentes entornos de pruebas, en este caso se utilizó Karma, principalmente porque es el mismo que se utilizó en la aplicación de chat, por lo que tendría algunos ejemplos que podrían ser útiles. También se pueden utilizar diferentes *frameworks* que son los que nos proporcionan la sintaxis y las funciones que utilizaremos para escribir estas pruebas.

La falta de experiencia y de documentación en internet, hizo algo complicado poder desarrollar y ejecutar estas pruebas. De primeras se tiene que crear un fichero de configuración llamado **karma.conf**, que con el entorno Karma instalado en el proyecto, se podía crear utilizando el comando **karma init** donde te realizará una serie de preguntas que si no tienes conocimiento no te vas a enterar de nada, por lo que tendrás que estar buscando información constantemente, cosa que se realizó en el proyecto al principio, después de tener este archivo de configuración con alguna prueba creada, para realizar pruebas iniciales y ver cómo funciona, empiezan a saltar una serie de errores como por ejemplo, el siguiente:

"Failed to load module script: Expected a JavaScript module script but the server responded with a MIME type of "video/mp2t". Strict MIME type checking is enforced for module scripts per HTML spec.".

Y como este, más errores que en internet tampoco se encuentra una solución clara, y sabiendo que en la aplicación de chat funcionaban los test correctamente,

se optó por replicar su archivo de configuración y adaptarlo a nuestra aplicación. Y después de una serie de cambios y de otros errores minúsculos, se consiguió arreglar los errores que nos impedían ejecutar los test de la aplicación.

Además, debido al uso de la aplicación de OpenVidu Call, surgen otros errores que no tiene solución a priori, por lo que se decide que los componentes que hagan uso de la aplicación se tendrán que probar de forma distinta. En este caso se optó por la realización de pruebas E2E, a través de la librería Playwright.

4.3.3. Diseño gráfico de iconos

La idea principal de los iconos es que sean agradables para los usuarios y contando con que se trata de la fusión de las dos aplicaciones, se optó por el uso de iconos de ambas aplicaciones para no perder la esencia de ambas.

OpenVidu Call cuenta con una serie de iconos y de interfaz que son utilizados bajo un "modo oscuro" en la aplicación, no cuenta con una función que permita este modo, y tampoco que permita el cambio manual de estos colores como hemos contado anteriormente, por lo que toda la aplicación se debe hacer siguiendo la paleta de colores de la aplicación de OpenVidu Call.

Gracias a que la aplicación de chat contemplaba esta funcionalidad de cambio de modo oscuro y modo claro, y contando con que se quería incluir la esencia de ambas aplicaciones, se han utilizado los siguientes iconos para la aplicación, que se observan en la figura 4.14. Al primer icono se le ha añadido un fondo negro para que se pueda ver, ya que como tal no hay fondo.

Figura 4.14: Iconos utilizados en la aplicación

4.4. Pruebas

En esta sección se hablará de los test automatizados realizados en la aplicación. Se cuenta con una gran cantidad de test por lo que se hablarán de las

más relevantes para la aplicación, aunque sean esenciales que todas se completen correctamente.

Se tiene que empezar explicando que, por temas técnicos, que se han comentado en el apartado de problemas encontrados en el desarrollo, no existen algunos test unitarios de componentes y se ha decidido optar por realizar tests *ent-to-end* para comprobar que estos componentes y que la aplicación en general funcionan como se esperaba.

Para el desarrollo de los test se han utilizado distintos *softwares* debido a que se han tenido que realizar dos tipos de test distintos. Para los test unitarios se ha utilizado un *framework* que proporciona la sintaxis y funciones, llamado Jasmine y para ejecutar estos test, se ha utilizado un entorno de ejecución llamado Karma. Para los test E2E se ha utilizado una librería llamada Playwright, que ofrece una gran compatibilidad con los navegadores y también, una API con un soporte para una multitud de lenguajes, lo que nos permite escribir las pruebas en cualquiera de estos lenguajes.

4.4.1. E2E

En estos tests vamos a comprobar que la aplicación funciona como se esperaba desde un principio y para empezar debe estar la aplicación desplegada. Como se observa en la figura 4.15, las primeras lineas de este código, determinan en que navegador se realizarán estas pruebas y que argumentos se le va a pasar al navegador, en este caso, al ser una aplicación de conferencia se requiere de un dispositivos, la cámara y el micrófono, por lo que se utilizan los argumentos -use-fake-ui-for-media-streamz -use-fake-device-for-media-stream" para indicarle al navegador que se usará un dispositivo virtual para simular estos dispositivos.

```
test('Should connect to the session and put a message and get it back', async () => {
  const browser = await chromium.launch({ headless: true , deviceScaleFactor: 1, // especificar el factor de escala de la página
  userAgent: 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)', // especificar el user agent
  args : ["--use-fake-ui-for-media-stream", "--use-fake-device-for-media-stream"]
});

  const context = await browser.newContext({
    permissions: ['camera', 'microphone'],
  });

  const page1 = await context.newPage();
  await page1.goto('http://127.0.0.1:4200');

  await page1.click('#join-button');

  await page1.click('#chat-panel-btn');

  await page1.type('#textAreaForm', 'Ejemplo');

  await page1.click('.sendButton');

  const page2 = await context.newPage();
  await page2.goto('http://127.0.0.1:4200');

  await page2.click('#join-button');

 const parentElement = await page2.$('.upper');

 const element = await parentElement.$(':last-child');

 const children = await parentElement.$$('mat-card-subtitle');

 const lastChild = children.pop();

 const texto = await lastChild.textContent(); // Obtiene el contenido de texto del elemento

 expect(texto).toContain('Ejemplo'); // Comprueba si el texto específico está presente en el contenido del elemento

 // Cerrar las páginas y el navegador
 await Promise.all([page1.close(), page2.close()]);
 await browser.close();
);
```

Figura 4.15: Código del test E2E

El código trata de hacer dos conexiones distintas a la aplicación web, y unirlos a la sesión. La página uno tratará de conectarse al chat y una vez ahí, enviará un mensaje con un texto de prueba. Entonces la página dos también se conectará al chat y comprobará que la página uno ha enviado este mensaje y que a su vez la página dos recibe estos mensajes en tiempo real, básicamente la función principal de nuestra aplicación.

4.4.2. Test unitarios

Se han realizado los test para los servicios y algún componente que se utiliza durante el funcionamiento de la aplicación, intentando cubrir al máximo todas las funciones de la aplicación.

Comenzando por los servicios, uno de los más utilizados es el servicio Message. De este servicio se comprueba que todos los métodos que ofrece, funcionen de forma esperada, por ejemplo como se observa en la figura 4.16, este test comprueba que el método de creación de mensajes funciona correctamente, creando una

variable con los campos necesarios y enviándolo mediante una petición POST y recibiendo la respuesta esperada.

```
it('should create a message', () => {
  const messageData = {
    sessionName: 'Test Session',
    message: 'Test Message',
    sender: 'Test Sender'
  };

  service.createMessage(messageData).subscribe(message => {
    expect(message.sessionName).toBe(messageData.sessionName);
    expect(message.message).toBe(messageData.message);
    expect(message.sender).toBe(messageData.sender);
  });

  const req = httpMock.expectOne(environment.API_URL + 'api/message');
  expect(req.request.method).toBe('POST');
  expect(req.request.body).toEqual(messageData);
});
```

Figura 4.16: Prueba de creación de mensajes

Otra petición Post a la que se ha realizado un test es la petición de envío de ficheros que se observa en la figura 4.17. Este test comprueba que, con una simulación de datos de un fichero estándar, la petición de POST al *backend* de la aplicación funciona correctamente.

```
it('should send a file', () => {
  const fileData = formSendFile

  service.sendFile(fileData).subscribe(message => {
    expect(message).toBeTruthy();
  });

  const req = httpMock.expectOne(environment.API_URL + 'api/files');
  expect(req.request.method).toBe('POST');
  expect(req.request.body).toEqual(fileData);

});
```

Figura 4.17: Prueba del envío de ficheros

También se ha comprobado que la realización de las peticiones GET funcionan correctamente, que se observan en la figura 4.18. El primer test que se llama

"should get messages.[en] el que se realiza una petición GET para recibir todos los mensajes de una sesión específica. Y el último test llamado "should get a file", que realiza una petición GET para recibir un fichero correctamente.

```
it('should get messages', () => {
  const sessionName = 'panel-directive-example';

  service.getMessage(sessionName).subscribe(messages => {
    expect(messages).toBeTruthy();
    expect(Array.isArray(messages)).toBe(true);
  });

  const req = httpMock.expectOne(environment.API_URL + 'api/message?sessionName=' + sessionName);
  expect(req.request.method).toBe('GET');
});

it('should get a file', () => {
  const fileId = '12345';

  service.getFile(fileId).subscribe(file => {
    expect(file).toBeTruthy();
  });

  const req = httpMock.expectOne(environment.API_URL + 'api/files/' + fileId);
  expect(req.request.method).toBe('GET');

});
```

Figura 4.18: Prueba de las peticiones GET del servicio Message

Para terminar de cubrir todas las funciones que se esperan del servicio Message. Nos falta comentar la última función que es el test llamado "should update a message", como se observa en la figura 4.19, que comprobará que una vez existiendo un mensaje en la base de datos se puede actualizar, utilidad de esta función principalmente está enfocado para el cambió de nombre de un usuario dentro de la aplicación.

```
it('should update a message', () => {
  const updateData = {
    cookie: 'Test Cookie',
    sender: 'Test Sender'
  };

  service.updateMessage(updateData).subscribe(messages => {
    expect(messages).toBeTruthy();
    expect(Array.isArray(messages)).toBe(true);
  });

  const req = httpMock.expectOne(environment.API_URL + 'api/message');
  expect(req.request.method).toBe('PUT');
  expect(req.request.body).toEqual(updateData);

});
```

Figura 4.19: Prueba del modificación de mensajes

El servicio Session es más simple debido a que no cuenta con muchas funciones, como se observa en la figura 4.20, ya que no está muy desarrollado por la poca necesidad para el proyecto, pero en futuras implementaciones podrá tener un gran peso en la aplicación.

```
it('should be created', () => {
  expect(service).toBeTruthy();
});

it('should create a session', () => {
  const sessionData = { sessionName: 'panel-test-example' };

  service.createSession(sessionData).subscribe(session => {
    expect(session.sessionName).toBe(sessionData.sessionName);
  });

  const req = httpMock.expectOne(environment.API_URL + 'api/session');
  expect(req.request.method).toBe('POST');
  expect(req.request.body).toEqual(sessionData);

  const mockSession = { sessionName: sessionData.sessionName };
  req.flush(mockSession);
});
});
```

Figura 4.20: Pruebas del servicio Session

4.5. Distribución y despliegue

Para desplegar la aplicación y poder distribuirla a todo el mundo, pudiéndose utilizar sin tener problemas de ejecución en cualquier computadora, se ha utilizado la herramienta Docker para poder simplificar todo este proceso.

Primero empezaremos describiendo el fichero de Docker Compose, que se ha comentado anteriormente en el apartado de problemas encontrados en el desarrollo 4.13. En este fichero se puede observar que se cuenta con cuatro contenedores, el primero referente al *backend* de la aplicación por lo que se ha tenido que desarrollar su respectivo Dockerfile, como se observa en la figura 4.21, en el que se indica cómo se instala esta parte de la aplicación y qué puertos van a ser utilizados: en este caso el 3000 y el 5000.

```
FROM node:16-alpine as build-step

RUN mkdir -p /usr/src/app
RUN mkdir -p /usr/src/app/backend

WORKDIR /usr/src/app/backend

COPY package*.json ./

RUN npm install --save --force

COPY . .

EXPOSE 3000
EXPOSE 5000

CMD npm run start
```

Figura 4.21: Dockerfile del *backend*

Siguiendo con el fichero de Docker Compose, el siguiente contenedor que se ha definido es el referente a MongoDB, donde se indica la imagen Docker que se utilizará junto con los puertos que en este caso es el 27017.

El siguiente contenedor que se ha definido es el de la aplicación de OpenVidu Call, donde se indica la imagen junto con el puerto que va a utilizar que es el 4443. Además, se le ha configurado una variable de entorno.

Y por último, se ha definido el contenedor referente al *frontend* de la aplicación por lo que ha tocado desarrollar y configurar el Dockerfile respectivo para este contenedor, que se observa en la figura 4.22. En el fichero se indica el comando necesario para instalar esta parte de la aplicación y el puerto por el que se va a poder conectar a este contenedor.

```
FROM node:16-alpine as build-step

RUN mkdir -p /usr/src/app
RUN mkdir -p /usr/src/app/frontend

WORKDIR /usr/src/app/frontend

COPY package*.json ./

RUN npm install --save --force

COPY . .

EXPOSE 4200

CMD npm run start
```

Figura 4.22: Dockerfile del *frontend*

5

Capítulo 5: Conclusiones y trabajos futuros

En este último capítulo se reflexionará sobre el proyecto, los objetivos que se han completado y futuras ideas para la ampliación del proyecto.

5.1. Resolución de objetivos

Lo primero que comentaremos serán los objetivos que hemos alcanzado, que son los siguientes:

1. **Unificación de las aplicaciones:**

   El objetivo principal de esta aplicación era la fusión de las dos aplicaciones para que funcionaran como una única, con las funcionalidades interconectadas para una mayor integración. Este objetivo sin duda ha sido el más duradero, ya que requirió de un estudio previo de ambas aplicaciones con la identificación de las funcionalidades o características, que iban a ser útiles para la aplicación. Además del estudio por separado, se tuvo que investigar cómo podían ser integradas unas con otras para poder aprovechar al máximo las características de ambas aplicaciones, como por ejemplo, el uso de *Signals*, el *socket* interno con el que cuenta OpenVidu Call, en vez de la librería de Socket.io que es la que utiliza la aplicación de chat. Al final, este objetivo se ha cumplido como se esperaba, incluso más fusionada de lo que

se creía capaz en un principio, lo que hace que sea una aplicación única.

2. **Satisfacer al usuario:**

   El último objetivo que también se ha completado ha sido la de satisfacer al usuario, este objetivo también era uno de los principales para mí, ya que da voz a los usuarios comunes que hacen uso de este tipo de aplicaciones a diario y que algunas veces no están satisfechos con decisiones que se han tomado en el diseño de la mayoría de aplicaciones. Que aquí ha tenido un gran peso en el desarrollo.

Gracias a la metodología de trabajo que se ha aplicado durante el desarrollo de la aplicación, se han podido completar correctamente todos los objetivos que se tenían previstos. Aun con todas las dificultades encontradas durante el desarrollo, ha sido vital la constancia y la investigación de todas las tecnologías implementadas.

5.2. Futuras implementaciones

Durante el desarrollo de los objetivos que se han comentado en el apartado anterior, han ido surgiendo nuevas ideas para futuros desarrollos que permitan a la aplicación evolucionar y mejorar en nuevas versiones.

- La creación de un sistema de gestión de usuarios que permita controlar a los usuarios, a qué sesión están conectados, qué nombre de usuario están utilizando y los mensajes que han enviado.
- La posibilidad de gestión de sesiones, donde se permita crear una sesión y elegir un número limitado de participantes. Que se pueda elegir el nombre que se le da a la sesión e incluso la posibilidad de poder compartir la sesión sin necesidad de estar enviando el enlace uno a uno.
- Que exista la posibilidad de planificar conferencias, con notificaciones automáticas a los usuarios que se han invitado. Además, que se pueda integrar con aplicaciones como Google Calendar o a los calendarios integrados en los dispositivos móviles.
- La inclusión de stickers o gifs en el chat de la aplicación, algo que han implementado algunas aplicaciones de videoconferencia y que sería interesante incluir en esta para completar las posibilidades que se pueden hacer en el chat.
- Como se ha comentado anteriormente, se hizo una encuesta a los usuarios para el tamaño de textos y colores de la aplicación. En esta encuesta hubo diferencias de opiniones por lo que obviamente la aplicación no esta a

gusto de todos. Entonces para evitar estas diferenciaciones se podría incluir una apartado de configuración que permita al usuario editar este tipo de conceptos a su gusto.

- En la vista de Join Session, te da la posibilidad de elegir qué dispositivo puedes utilizar para la cámara como para el micrófono. Con la cámara se puede hacer una previsualización de cómo se vería en la conferencia pero estaría bien la posibilidad de comprobar el sonido del micrófono.

5.3. Conclusiones personales

La realización de este proyecto ha sido una manera de consolidar y darme la confianza necesaria para enfrentarme a cualquier otro desarrollo que se me presente en el futuro. Este proyecto me ha dado la posibilidad de superarme a mí mismo constantemente; de enfrentarme a errores y de sobreponerme a todos ellos. Ha supuesto un gran reto debido al poco conocimiento del que disponía al inicio del desarrollo. Además, la necesidad de enfrentarme a dos aplicaciones que, aunque cuenten con su documentación, son totalmente desconocidas para mí, hace que sea una gran representación de lo que me puede esperar en un futuro en cualquier puesto de trabajo.

Todo esto teniendo en cuenta que el grado solo cuenta con una asignatura en el que se trata este tipo de desarrollo. Aunque fue una asignatura bastante completa en la que se dan bastantes nociones, al ser una única asignatura no se puede contar con todo el conocimiento necesario, por lo que es necesario hacer labores de investigación.

Y, sobre todo, la parte que, personalmente es una de las mayores dificultades de todo el trabajo, ha sido la memoria. Principalmente se debe a que soy una persona a la que le motiva programar y tiene paciencia para dedicarle muchas horas. Sin embargo, a la memoria posiblemente sea la parte en la que más tiempo he invertido de todo el proyecto, y supongo que es por la falta de experiencia en proyectos similares. Durante los cuatro años del grado no he tenido que enfrentarme a trabajos de esta magnitud y calibre, aunque en todos los que he realizado se requería una memoria o documentación. Sin embargo, no son comparables al nivel que se exige en el proyecto actual.

Como conclusión, completar este trabajo ha sido una gran satisfacción para poder demostrar de lo que soy capaz y de lo que he aprendido durante todos estos años.

Bibliografía

Vanessa Marely Aristizabal Angel. (23 de julio de 2020). *¿Qué es Angular?* . Obtenido de ngchallenges.gitbook.io: https://ngchallenges.gitbook.io/metamorfosis-de-angular/nivel-0-oruga/que-es-angular

[Implemente una base de datos multicloud]. (s.f.). Obtenido de mongodb.com: https://www.mongodb.com/es/atlas/database

José Luis Chacón. (25 de octubre de 2021). *TypeScript: qué es, diferencias con JavaScript y por qué aprenderlo*. Obtenido de profile.es: https://profile.es/blog/que-es-typescript-vs-javascript/

[¿Qué es Angular Material?]. (s.f.). Obtenido de material.angular.io:

https://material.angular.io/

[¿Qué es Playwright?]. (s.f.). Obtenido de playwright.dev: https://playwright.dev/

[¿Qué es IntelliJ IDEA?]. (s.f.). *¿Qué es IntelliJ IDEA?*. Obtenido de jetbrains.com: https://www.jetbrains.com/es-es/idea/features/

Yúbal Fernández. (30 de octubre de 2019). *Qué es Github y qué es lo que le ofrece a los desarrolladores*. Obtenido de xataka.com: https://www.xataka.com/basics/que-github-que-que-le-ofrece-a-desarrolladores

Canós, J. H., Letelier, P., Penadés, M. C. (2003). *Metodologías ágiles en el desarrollo de software*. Universidad Politécnica de Valencia, Valencia, 1-8.

Javier Garzas. (23 de mayo de 2012). *Scrum para Dummies (1/2). Las ideas de Scrum en 2 post de 5 min.*. Obtenido de javiergarzas.com:

https://www.javiergarzas.com/2012/05/scrum-dummies-1.html

[¿Qué son los requisitos funcionales y no funcionales?]. (s.f.). Obtenido de visuresolutions.com: https://visuresolutions.com/es/blog/requirements-specification/

Pablo Huet. (24 de agosto de 2022). *Arquitectura de software: Qué es y qué tipos existen*. Obtenido de openwebinars.net: https://openwebinars.net/blog/arquitectura-de-software-que-es-y-que-tipos-existen/

Anexos

En este apartado se proporcionará la información sobre la aplicación y los pasos necesarios para poder ejecutar la aplicación directamente en tu dispositivo. Todo lo que aparece a continuación se encuentra en el README del GitHub y están documentados en inglés para que sea accesible para todos los desarrolladores.

OpenViduCall_Chat

This project is a develop from OpenVidu Call with a developed chat with all functions that are need. The project is fully developed using Angular for frontend and NestJS for the backend with a MongoDB database.

Prerequisites

Requirements for deploy the aplication:

- [Docker]

Installing

A step by step series of instructions that will allow you to install the project and deploy the aplication. For this instructions you have to be on the docker-compose file.

First you have to create the docker's containers:

```
docker-compose build
```

And then, you cant deploy the aplication:

```
docker-compose up
```

If you want to develop for next versions you have to do this instruction before you build the containers:

```
docker-compose down
```

Running the tests

There are two types of test so to execute them, we nned to execute this two commands

First you have to be on the frontend directory and then

To execute the unitary tests:

```
ng test
```

and before to execute the E2E tests, you have to had the application running and then execute this command:

```
npx playwright test
```

Built With

- Angular - Used to the Frontend framework
- NestJS - Used to the Backend framework
- MongoDB - Database
- Docker - For deployment in containers

Authors

- Jorge Esteban Pérez - Full Development - JEstebanPerez

I hope it's useful for developers