UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

APP FULL STACK PARA PLATAFORMA ACADÉMICA CON FRONTEND MÓVIL

TRABAJO FIN DE GRADO

Junio - 2022

AUTOR: Ángel Muñoz Roche

DIRECTOR/ES: Miguel Onofre Martínez Rach

RESUMEN

Este proyecto consiste en el desarrollo full stack de una aplicación móvil que presta servicios de comunicación. El Backend y el Frontend que forman la aplicación han sido desarrollados de manera independiente, el primero consiste en una API construida con Laravel y el segundo se trata de una interfaz de aplicación móvil multiplataforma escrita en React Native.

La plataforma proporcionará un chat en tiempo real a los miembros de un centro académico, tanto profesores como alumnos, sin la necesidad de conocer el teléfono móvil de dichos miembros. La idea es que la aplicación dependa únicamente del centro académico en cuestión y de esta manera no se vea obligado a utilizar tecnologías que dependan de terceros.

Para la completa utilización de la plataforma implementada, los usuarios deberán autenticarse con sus credenciales personales. Una vez un usuario haya iniciado sesión podrá leer los mensajes enviados en las salas de chats a las que pertenece y, evidentemente, enviar nuevos mensajes. También podrá crear salas de chats privadas o grupales con cualquier miembro del centro académico, ya que se permite la libre comunicación entre todos ellos. En caso de que el usuario sea un profesor, tendrá la facilidad de crear un grupo con los miembros de cualquier asignatura que imparta.

PALABRAS CLAVE

Backend, Frontend, Laravel, React Native, HTTP, Websockets, App

ABSTRACT

This project consists of the full stack development of a mobile application that provides communication services. The Backend and Frontend that make up the application have been developed independently, the first is composed of an API built with Laravel and the second is a cross-platform mobile application interface written in React Native.

The platform will provide a chat in real time for the members of an academic center, both teachers and students, without the need to know the mobile phone of said members. The idea is that the application depends only on the academic centre that implements it, and in this way it is not forced to use technologies that depend on third parties.

For the usage of the implemented platform, users must authenticate with their personal credentials. Once a user has logged in, they will be able to read the messages sent in the chat rooms to which they belong and, of course, send new messages. You can also create private or group chat rooms with any member of the academic center, since free communication is allowed between all of them. In case of the users are teachers, they will have the facility to create a group with the members of any subject they teach.

KEYWORDS

Backend, Frontend, Laravel, React Native, HTTP, Websockets, App

ÍNDICE

1. INTRODUCCIÓN 7
1.1. CONTEXTO 7
1.2. OBJETIVOS 8
1.3. ESTRUCTURA DE LA MEMORIA 8
1.4. MENCIÓN AL DESARROLLO 9
2. ESTADO DEL ARTE 10
2.1. SOLUCIONES SIMILARES EN LA INDUSTRIA 10
2.2. TECNOLOGÍAS DE DESARROLLO 11
2.2.1. DESARROLLO DE API REST (BACKEND) 11
2.2.2. DESARROLLO MÓVIL (FRONTEND) 15
3. ANÁLISIS 20
3.1. ANÁLISIS DE REQUISITOS 20
3.2. ANÁLISIS DE LOS CASOS DE USO 21
4. ARQUITECTURA DE LA SOLUCIÓN 24
4.1. ESQUEMA GENERAL DE LA APLICACIÓN 24
4.2. ESTRUCTURA DEL BACKEND 26
4.2.1. ¿QUÉ ES EL BACKEND? 26
4.2.2. ¿QUÉ ES UNA API REST? 27
4.2.3. PHP Y LARAVEL 29
4.2.4. ¿QUÉ ES MYSQL? 32
4.2.5. WEBSOCKETS 35
4.3. ARQUITECTURA DEL FRONTEND 37
4.3.1. ¿QUÉ ES EL FRONTEND? 37
4.3.2. REACT NATIVE 38
4.3.3. CLASES DE COMPONENTES EN REACT NATIVE 42
4.3.4. DEPENDENCIAS EXTERNAS UTILIZADAS 43
4.4. HERRAMIENTAS 46
4.4.1. MYSQL WORKBENCH 46
4.4.2. VISUAL STUDIO CODE 47
4.4.3. EXPO GO 47
4.4.4. POSTMAN 48
5. IMPLEMENTACIÓN 50

5.1. IMPLEMENTACIÓN DEL BACKEND....50
5.1.1. DISEÑO DE LA BASE DE DATOS....50
5.1.2. ESTRUCTURA DEL PROYECTO LARAVEL....58
5.1.3. ELEMENTOS DEL PROYECTO....61
5.2. IMPLEMENTACIÓN DEL FRONTEND....71
5.2.1. ESTRUCTURA DEL PROYECTO REACT NATIVE....71
5.2.2 COMPONENTES DEL PROYECTO....80
6. RESULTADOS....97
7. CONCLUSIONES Y LÍNEAS FUTURAS....100
8. BIBLIOGRAFÍA....101

1. INTRODUCCIÓN

1.1. CONTEXTO

Es innegable que hoy en día las tecnologías de la información y la comunicación han pasado a ocupar un primer plano en la vida de un gran número de personas. Este hecho es debido a la existencia de una gran cantidad de herramientas que proveen a los usuarios de una enorme variedad a la hora de comunicarse a través de estas tecnologías. Poseen además un ámbito de uso muy amplio, albergando desde su empleo en la comunicación con amigos o conocidos hasta la utilización para el establecimiento, incluso, de comunicación militar. Dentro de este abanico se encuentra también, por supuesto, por la comunicación en empresas o en centros docentes.

Profundizando en la utilización de estas tecnologías en el ámbito docente, es evidente que en los últimos años se han incorporado o renovado herramientas con el fin de facilitar las gestiones de profesores y alumnos, así como la comunicación entre ellos. No obstante, la mayoría de las herramientas en los centros dependen de terceros, por lo que el centro no tiene un control total sobre dichas herramientas.

Estos últimos años también se ha visto incrementado el uso de los smartphones de manera exponencial, llegando a ocupar un gran ámbito del día a día de la mayoría de personas. Parece lógico pues que el desarrollo de una herramienta deba tener presencia en el ecosistema de los dispositivos móviles.

Por todos estos motivos, el presente proyecto consistirá en el desarrollo de una aplicación móvil que permitirá la comunicación en tiempo real entre los integrantes de un centro académico, ya sean alumnos o profesores. La idea es que los datos sean almacenados en servidores propios de cada centro.

En la actualidad la forma más popular de desarrollar aplicaciones es creando el Frontend y el Backend de forma independiente. Más concretamente el Backend se encarga de ofrecer un API al Frontend para que éste pueda obtener los datos y la lógica presentes en el lado del servidor y mostrarlos al usuario mediante su interfaz. Para poder realizar este proceso el Frontend debe realizar peticiones HTTP a la mencionada API y convertir la respuesta a un formato que pueda representar por pantalla. Construyendo una API se deja abierta la posibilidad de conectar diversos Frontend a esa misma API, pudiendo ser estos nuevos Frontends de diferente naturaleza, ya que podrán ser webs, aplicaciones o incluso dispositivos electrónicos, en definitiva cualquier cosa que pueda ejercer de cliente y realizar peticiones a un servidor.

1.2. OBJETIVOS

El principal objetivo del presente proyecto es el desarrollo de una aplicación móvil multiplataforma que implementará un prototipo de chat en el que los integrantes de un centro docente puedan comunicarse entre sí en tiempo real. Para una mayor claridad separaremos el objetivo principal en dos objetivos más concretos. Éstos serán: el desarrollo del Frontend y la construcción del Backend.

El primer objetivo, equivalente al desarrollo del Frontend, es la realización de un correcto diseño de la interfaz de las pantallas que formarán la aplicación. Cada pantalla, además, deberá ser capaz de representar los datos provenientes del servidor.

El segundo objetivo, correspondiente a la construcción del Backend, es la construcción de una API a la que el Frontend pueda pedirle los datos que necesite en cada momento. Por eso en la API deben desarrollarse las diferentes funcionalidades con las que contará el chat, como puede ser la creación de las salas de chat o la comunicación en tiempo real entre los usuarios de una sala. La API deberá ser desarrollada de tal forma que sea posible acceder a ella desde nuevos Frontends en un futuro.

1.3. ESTRUCTURA DE LA MEMORIA

- **Introducción**: primer apartado en el cual se da un punto de vista general de la aplicación explicando los objetivos que persigue el desarrollo del proyecto.

- **Estado del arte**: apartado en el que se detallan las tecnologías más populares con las que se realiza un proyecto de estas características en la industria. También se mencionan aplicaciones similares a la desarrollada en el proyecto.

- **Análisis**: apartado que agrupa los diferentes requisitos y casos de uso de los que deberá constar la aplicación.

- **Arquitectura**: apartado en el cual se explica cómo está formada la aplicación. Incluye la arquitectura general de la aplicación explicando cómo se conectan el Frontend y el Backend, y también la arquitectura de la constan cada una de las dos partes.

- **Implementación**: apartado en el que se explica el código implementado tanto para el Frontend como para el Backend.

- **Resultados**: apartado en el que se mostrarán las principales funcionalidades del proyecto.

- **Conclusiones**: apartado que recoge las conclusiones sacadas tras la elaboración del proyecto y las posibles mejoras que podrían implementarse en un futuro.

- **Bibliografía**: último apartado en el que se facilitan las referencias que se han utilizado para la elaboración del proyecto.

1.4. MENCIÓN AL DESARROLLO

El Backend de este proyecto ha sido desarrollado conjuntamente con un compañero del Grado en Ingeniería en Tecnologías de Telecomunicación, que ha desarrollado el TFG “Aplicación web con React y Laravel para plataforma académica en tiempo real”. En cuanto al Frontend, está formado por dos interfaces (web y móvil) que han sido diseñadas e implementadas por separado. Él ha desarrollado la aplicación web y yo la aplicación móvil.

De esta forma, la aplicación quedaría como un Backend compartido por una aplicación móvil y una página web. Por lo tanto, un mismo usuario tendría la posibilidad de acceder a la plataforma por cualquiera de los dos medios, conservando siempre toda su información. Es decir, el usuario podría enviar un mensaje iniciando sesión desde la aplicación móvil y si entrase a la web con los mismo credenciales vería que también se le ve reflejado el mensaje enviado.

2. ESTADO DEL ARTE

En esta sección se mencionarán las diferentes aplicaciones móviles más conocidas similares a la desarrollada en el presente proyecto, así como las tecnologías de desarrollo más utilizadas actualmente con las que también habría sido posible desarrollar la aplicación. Se explicarán por separado las tecnologías para el desarrollo del Backend y las del desarrollo del Frontend. La información recopilada en este apartado será utilizada en una futura sección de la memoria para justificar por qué han sido seleccionadas las herramientas para el desarrollo de la aplicación.

2.1. SOLUCIONES SIMILARES EN LA INDUSTRIA

En la actualidad existen distintas aplicaciones que podrían solventar el problema de la comunicación en tiempo real entre el alumnado y el profesorado, entre las más conocidas destacan Whatsapp, Telegram o Discord.

Logotipos de Whatsapp, Telegram y Discord

La principal ventaja que tendría nuestra aplicación sobre las mencionadas es la comodidad a la hora de establecer la comunicación con los profesores o alumnos deseados, ya que se permite contactar con cualquier persona registrada en la base de datos, que en un principio serían directamente todos los alumnos y profesores del centro académico. Además, a diferencia de las soluciones a las que se hace mención en el párrafo anterior, el centro

académico en cuestión tendría el total control de la aplicación, ya que los administradores designados por el centro serían los únicos que tendrían la capacidad de gestionar la aplicación. Por último cabe destacar que los únicos datos personales necesarios para el registro son el nombre y un correo electrónico, datos que ya tendría previamente el centro académico.

2.2. TECNOLOGÍAS DE DESARROLLO

Actualmente existen numerosas tecnologías que permitirían desarrollar una solución como la que se propone en este proyecto. Para explicar las más populares separaremos este subapartado dos partes. El primero tratará de explicar las tecnologías más eficientes para el desarrollo de la parte del Backend de la aplicación, centrándose siempre en el desarrollo de una API REST. El segundo, por su parte, mostrará las tecnologías actuales para desarrollar la parte del Frontend, centrándose en la creación de Frontends de aplicaciones móviles.

2.2.1. DESARROLLO DE API REST (BACKEND)

Para la implementación de una API REST pueden utilizarse diversos lenguajes de programación como JavaScript, Python o PHP. Pero existen diversos software que utilizan esos lenguajes para implementar una API REST. Por eso, este apartado se centrará en recolectar información sobre los softwares más populares de ese tipo. Algunos de los más relevantes hoy en día serían: Express JS, Flask, Django y Laravel.

Express JS

Para poder entender en qué consiste Express JS antes es necesario entender qué es Node JS. Y es que resulta que Node JS es un entorno de ejecución para JavaScript que está orientado a eventos asíncronos y que está diseñado para crear aplicaciones escalables. De esta forma, posee la capacidad de gestionar más de una conexión de forma simultánea, activándose para devolver una llamada para cada conexión; en el caso de no existir ninguna conexión se dormirá. No requiere el uso de hilos del Sistema Operativo, y en consecuencia pasa a ser más eficiente y sencillo de utilizar. También es importante destacar que permite la ejecución de un proyecto en tiempo real. [1]

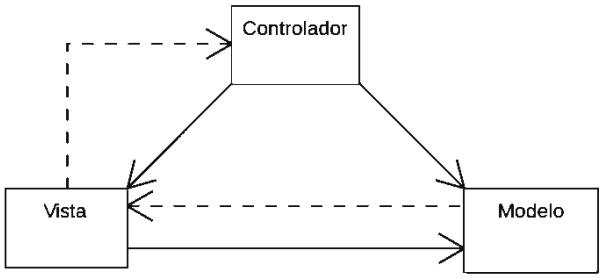
Logotipo de Node JS

Express, por su parte, es el framework más popular de Node, de hecho ha sido designado el marco de servidor estándar de facto para Node JS. Express es básicamente un marco de desarrollo minimalista de aplicación web de Backend que permite estructurar una aplicación de una manera ágil, nos proporciona funcionalidades como el enrutamiento o la gestión de sesiones y cookies [2]. Es gratuito y de código abierto. Además de su alta escalabilidad posee un gran rendimiento y velocidad, lo que lo hace perfecto para para tu desarrollo de aplicaciones de negocio para empresas. Obviamente está basado en JavaScript, por lo que es rentable a las empresas por ser un lenguaje altamente utilizado para el desarrollo Frontend y consecuentemente un desarrollador con conocimientos de JavaScript podría trabajar en el Backend y en el Frontend de una misma aplicación. Es utilizado por grandes empresas como pueden ser IBM, PayPal o Walmart. [3]

Flask

Flask nos permite crear de una manera muy sencilla aplicaciones web y API REST con Python. Es un Framework escrito en Python y concebido para facilitar el desarrollo bajo el patrón MVC (modelo-vista-controlador). [4]

El patrón de arquitectura de Software MVC se basa en la separación del código en tres capas diferentes, acotadas por su responsabilidad, denominadas Modelos, Vistas y Controladores.



Patrón MVC

En la capa Modelo es donde se trabaja con los datos. Posee pues la capacidad de acceder a la base de datos y actualizarlos mediante las consultas necesarias como puede ser select, insert, delete, etc.

En la capa vista está contenido el código de la aplicación que permite visualizar la UI (interfaz de usuario) de la aplicación. Las vistas no pueden acceder directamente a la base de datos, sino que tienen que pedirle los datos a la capa de Modelo.

La capa controlador es capaz de proporcionar una respuesta a la acción que ejecute el usuario. Se encarga de enrutar los comandos a las otras capas, es decir, no puede acceder a la base de datos ni visualizar la interfaz, sino enlazar a las capas que se poseen las funciones mencionadas. [5]

Como ya ha sido mencionado, Flask permite la creación de REST APIs, siendo uno de los mejores frameworks de desarrollo para el desarrollo de este tipo de soluciones su sencillez. Para ello se utilizan diferentes extensiones para facilitarle la vida al desarrollador, aunque todas estas extensiones son proporcionadas por la comunidad. Destaca la extensión Flask-RESTful que agrega una compatibilidad para crear APIs de manera rápida trabajando con sus propias librerías; y también la extensión Flask Migrate que permite generar las tablas de la base de datos a partir de ficheros de migración. [6] [7]

Logo Flask

Django

Para que este punto no sea repetitivo se centrará en la comparación de Django y Flask, ya que Django, al igual que Flask, está basado en Python y sigue la arquitectura MVC.

Logo Django

Para la creación de una REST API Django requiere del kit de herramientas Django REST Framework, mientras que en Flask son necesarios varios complementos populares necesarios para conseguir resultados comparables, como se ha explicado en el punto anterior. Un dicho popular utilizado para comparar ambos frameworks es ‘Los piratas usan Flask, La Armada usa Django’. [8]

Tanto Flask como Django son tecnologías que admiten muchas capacidades respecto al diseño de REST APIs como pueden ser la autenticación de usuario, el límite de usuarios registrados o el mapeado de bases de datos relacionales.

Entre las ventajas de Django destacan la inclusión de un control de versiones muy flexible, mientras que en Flask debe ser gestionado por el propio desarrollador. Django incluye también la capacidad de crear páginas para navegar, que permite que el desarrollador pueda ejecutar peticiones HTTP rápidamente, lo cual es sumamente útil a la hora de realizar un test. Como última ventaja destacable de Django mencionaremos la frecuente actualización de Django REST Framework, que se encuentra bajo un desarrollo activo muy fuerte.

Como ventajas de Flask destacan la posibilidad de obtener un mejor rendimiento como consecuencia de su diseño minimalista, y también la mejor integración con bases de datos no relacionales, ya que Django está vinculado a bases de datos relacionales.

Laravel

Laravel es un framework de PHP diseñado para proporcionar una gran productividad al desarrollador. Al igual que Django y Flask, está basado en la arquitectura MVC. Proporciona una gran ayuda al desarrollador, como puede ser ahorrarle ciertos aspectos en el desarrollo como instanciar clases y métodos sin la necesidad de repetir el proceso. Esto hace que el desarrollador quede exento de cometer errores a la hora de modificar estos apartados en el código. [9]

Logo de Laravel

Como es evidente, Laravel puede utilizarse como API para el Backend de una aplicación una aplicación móvil. Permite autenticación y almacenamiento de datos.

Ya que este es el framework elegido para la realización de la API de este proyecto se explicará más detalladamente en un futuro apartado de la memoria.

2.2.2. DESARROLLO MÓVIL (FRONTEND)

De igual manera que para el desarrollo del Backend, existen múltiples lenguajes posibles a la hora de desarrollar el Frontend de una aplicación móvil como el realizado en este proyecto, pero los más relevantes en el mercado actual son: Kotlin, Swift, React Native y Flutter.

Kotlin (Apps nativas de Android)

Como sabemos, si queremos programar aplicaciones para dispositivos Android se requiere de un lenguaje de programación. Actualmente este lenguaje puede ser Java, que fue el primero adoptado por Google para el desarrollo de sus aplicaciones móviles; o Kotlin, que actualmente es el lenguaje recomendado por Google para el desarrollo de aplicaciones nativas en Android. Kotlin es el

lenguaje principal para la creación de este tipo de aplicaciones desde 2019. Salvo sorpresa acabará convirtiéndose en el estándar a la hora de crear esta clase de aplicaciones. Evidentemente también es requerido un entorno de programación para poder desarrollar una app, el entorno que se utiliza es Android Studio. [11]

Logotipos de Android y Kotlin

La utilización de Kotlin provee al desarrollador de grandes beneficios. [10] Entre ellos destaca la posibilidad de desarrollar menor cantidad de código y más legible, lo que conlleva a una menor probabilidad de cometer errores a la hora de escribir código. Además, es un lenguaje fácil de aprender que cuenta con una gran comunidad, cuenta un gran apoyo y está en pleno auge. Por último, es relevante mencionar su interoperabilidad con Java, es decir, es posible utilizarlo conjuntamente con ese lenguaje sin tener que pasar todo su código a Kotlin.

Como punto negativo podemos decir que a pesar de ser multiplataforma y permitir el desarrollo de aplicaciones iOS no resulta nada sencillo trabajar en ambas versiones a la vez, por lo tanto no es muy recomendable utilizar este lenguaje si queremos desarrollar una aplicación móvil multiplataforma.

Swift (Apps nativas de iOS)

Swift es el lenguaje de programación desarrollado por Apple y enfocado en el desarrollo de aplicaciones nativas de iOS. Según Apple es un lenguaje potente y abierto a todos para crear apps increíbles. [12] Está pensado con la intención de dar a los desarrolladores más libertad que nunca y para que cualquiera tenga la capacidad de plasmar sus ideas, ya que es de código abierto y muy sencillo de utilizar. Es un lenguaje rápido y eficaz capaz de proporcionar información en tiempo real. Al igual que pasaba con Kotlin y Java, Swift puede integrarse a la perfección con código escrito en Objective-C (antiguo lenguaje de programación de aplicaciones iOS). Para hacernos a la idea de la diferencia en términos de potencia y rendimiento de Swift diremos que es 2,6 veces más

rápido que Objetive-C, esto se debe a que está mucho más cerca del hardware y está más orientado a la CPU por lo que contiene menos capas y una optimización muy superior a su predecesor. También requiere de un entorno de desarrollo para poder programar las aplicaciones, que en este caso será Xcode.

Logotipos de iOS y Swift

La parte mala de utilizar este lenguaje es la imposibilidad de crear una aplicación multiplataforma con él, es decir, si quisiéramos realizar una aplicación que sirviese tanto para iOS como para Android tendríamos que hacer una app nativa de iOS con Swift y otra app nativa de Android con Kotlin.

Flutter (Dart)

Flutter es un SDK (Kit de desarrollo de software) de código abierto desarrollado por Google con la capacidad de desarrollar y compilar de forma nativa aplicaciones móviles para iOS y para Android. Fluter se basa en la combinación de diferentes elementos gráficos denominados Widgets para la creación de interfaces gráficas (vistas) muy versátiles, los mencionados Widgets pueden ser simplemente botones, texto o imágenes, y también pueden ser Widgets formados por otros Widgets, es decir, los Widgets creados son reutilizables haciendo así a Flutter un framework muy potente y sencillo de utilizar. [13]

Logotipo de Flutter

El lenguaje de programación que utiliza es Dart, que también es de código abierto y desarrollado por Google. Su principal objetivo es permitir a los

desarrolladores utilizar un lenguaje orientado a objetos y con análisis estático de tipo. Además, se desarrolló también con el propósito de hacer el proceso de desarrollo lo más cómodo y rápido posible para los desarrolladores. No requiere de un entorno de desarrollo especial para la compilación y ejecución de los programas realizados ya que se podrá compilar de forma nativa. Es un lenguaje relativamente sencillo de aprender si se tienen conocimientos previos de Java, C++ o JavaScript debido a que Dart tiene una sintaxis parecida a estos lenguajes.

Además, Dart consta de un gran apoyo para la asincronía, y trabajar con generadores e iterables es extremadamente sencillo.

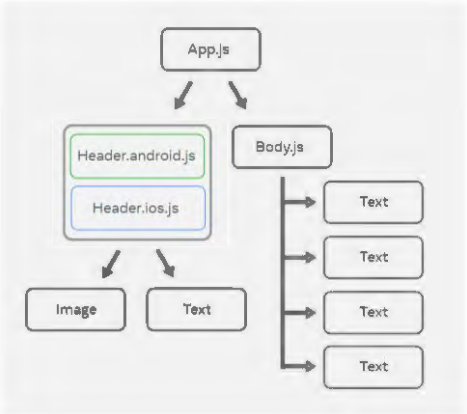
React Native

React Native es un framework basado en React que permite crear aplicaciones reales nativas para iOS y Android. React, a su vez, es una librería de JavaScript desarrollada por Facebook. El principio de funcionamiento de React Native es muy similar al de Flutter, solo que los componentes visuales creados se denominan componentes en lugar de Widgets. A los componentes creados se le pueden pasar ciertas propiedades (denominadas props), por ejemplo podríamos crear un botón con cierta forma pero queremos reutilizarlo en otra vista con otro color, sólo sería necesario llamar al componente del botón creado y pasarle por props el color deseado, de esta forma se conseguiría reutilizar el mismo botón en dos vistas pero de diferentes colores.

Logotipo de React Native junto los de iOS y Android

Entre las características más relevantes de React Native destacamos la compatibilidad Cross-Platform permitiendo crear aplicaciones de manera simultánea y con un único código para iOS y Android, esto implica además una funcionalidad nativa como una aplicación creada en Swift y otra creada en Kotlin para Android. También es importante destacar lo sencillo que es de

aprender, especialmente si se tienen conocimientos previos de JavaScript. Y por último, cabe mencionar que permite agregar código nuevo a una aplicación en ejecución, lo que reduce el riesgo de perder algunas funcionalidades durante una recarga completa o la reconstrucción de la aplicación. [15] [16]



Multiplataforma en React native

React Native es el framework seleccionado para el desarrollo de la aplicación móvil a la que se refiere esta memoria, por lo tanto, se comentará en mayor profundidad en futuros apartados.

3. ANÁLISIS

En este apartado se evaluarán los requisitos que deberá cumplir la aplicación, lo cual resulta es un proceso indispensable para la definición de la funcionalidad del proyecto y para la posterior evaluación de su calidad. También serán expuestos los diferentes casos de uso de los que constará la aplicación.

3.1. ANÁLISIS DE REQUISITOS

El proyecto tiene como objetivo la implementación de un chat que permita la comunicación en tiempo real entre diferentes usuarios. Por tanto, el primer requisito fundamental es la posibilidad de que un usuario tenga unos credenciales personales para poder efectuar un inicio de sesión en la aplicación. En este punto resulta importante destacar que un usuario no poseedor de una cuenta no tiene la posibilidad de registrarse, ya que sólo un administrador es capaz de registrar a un nuevo usuario en la aplicación, restringiendo así el acceso únicamente a las personas que desee el centro académico. Los usuarios tendrán asignado un rol de profesor o de alumno, y en función de este rol puede variar la usabilidad de la aplicación.

En referencia al apartado de usuarios, cada usuario podrá escoger una imagen de perfil propia y modificarla siempre que quiera. También debe tener la opción de cambiar su contraseña a una más segura o que le resulte más cómoda de memorizar, ya que en un primer momento se le asignará una por defecto.

El siguiente requisito sería la creación de las diferentes salas de chat. La idea es que cada usuario tenga, por defecto, una sala de chat para cada asignatura en la que está matriculado. En esa sala de chat estarán todos los alumnos y profesores de la asignatura en cuestión. También será posible para un usuario abrir una conversación con cualquier otro usuario de la aplicación, así como la creación de grupos con los integrantes que él mismo desee. Los usuarios con rol de profesor tendrán la posibilidad de filtrar directamente una asignatura de las que impartan para crear un grupo con integrantes de esta asignatura, esto resulta muy útil por si los profesores quisieran crear un chat grupal para cada uno de los grupos de prácticas en los que es habitual separar a los alumnos de una asignatura. Al momento de crear una sala grupal el creador elegirá, adicionalmente a los usuarios que lo integrarán, un nombre y una imagen; y el propio creador quedará como el administrador de esa sala.

Las salas grupales de chat deben incluir una opción de modificación de los parámetros que lo definen. Los administradores del chat tendrán la capacidad

de cambiar el nombre y la imagen del chat, y además tendrán la capacidad de agregar o eliminar integrantes.

El último requisito importante es, como no puede ser de otra manera, la implementación y establecimiento de una comunicación en tiempo real en las diferentes salas de chat. De no ser así toda la aplicación carecería de sentido, pues en el caso de no existir dicha comunicación en tiempo real y un usuario realizase una pregunta en una sala de chat y continuase dentro de esa misma sala no tendría la posibilidad de ver la respuesta a menos que refrescase la pantalla, mientras que en el caso de existir podría estar indefinidamente hablando con otros usuarios sin tener que moverse de la sala de chat y visualizando instantáneamente las respuestas del resto de integrantes de la sala.

3.2. ANÁLISIS DE LOS CASOS DE USO

A partir de todos los requisitos mencionados en el punto anterior será posible detallar los casos de uso de la aplicación a continuación.

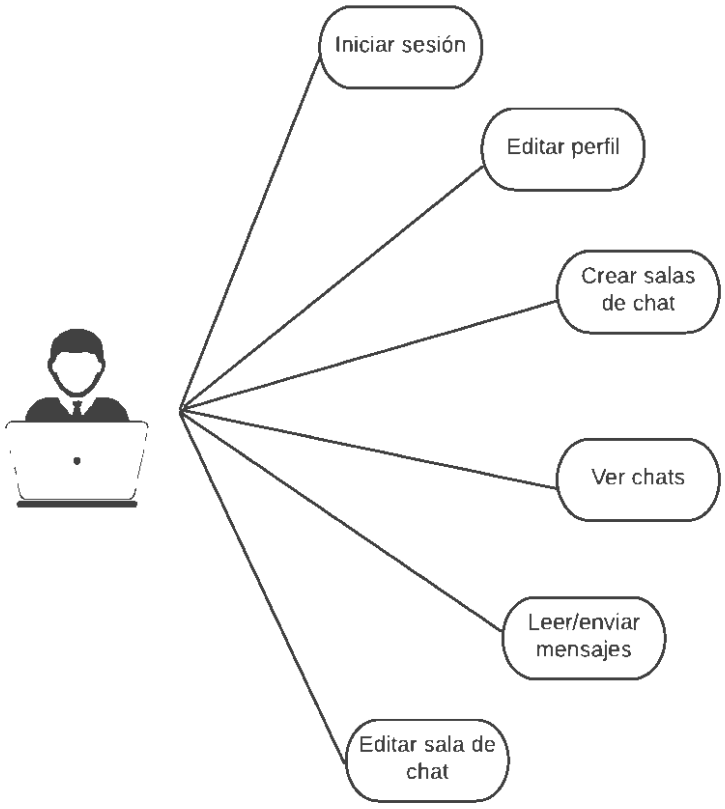


Diagrama de los casos de uso

Inicio de sesión

Como se ha mencionado en el apartado anterior, solamente un administrador designado por el centro académico que haya implementado la aplicación será capaz de registrar usuarios en la aplicación. Una vez el usuario esté registrado, podrá iniciar sesión con el correo con el que haya sido registrado (normalmente debe ser el correo del centro académico) y la contraseña que le haya sido asignada. También se le asignará un rol a cada usuario, bien de alumno o bien de profesor, que evidentemente no podrá ser modificado salvo por un administrador en el caso de que se hubiese producido un error en el momento de asignarlo.

Edición de perfil

El usuario tiene la opción de editar su imagen de perfil accediendo a la galería de imágenes de su dispositivo móvil y seleccionando la imagen deseada. También podrá cambiar la contraseña a la que desee.

Creación de salas de chat

Existe la posibilidad de crear nuevas salas de chats. Estas salas pueden ser privadas o grupales, o en el caso de los profesores salas de tipo asignatura, que serán subgrupos de una asignatura y aceptarán únicamente integrantes pertenecientes a la asignatura en cuestión. Los chats grupales exigirán un mínimo de selección de dos usuarios adicionales al creador del chat, y tendrán que tener un nombre asignado. En el caso de los chats privados, evidentemente, solo será posible seleccionar a un usuario.

Visualización de los chats

El usuario verá una lista de todos los chats a los que pertenece y cada uno de los ítems de esta lista será clickable y llevará a la pantalla del chat correspondiente.

Lectura y envío de los mensajes de una sala de chat

Una vez dentro de la pantalla de un chat podrán leerse todos los mensajes que se hayan enviado en ese chat, y recibir nuevos mensajes en directo gracias a la comunicación en tiempo real mencionada en el punto de requisitos. También será posible enviar mensajes, que serán añadidos al instante a la lista de mensajes visibles de la pantalla de chat. Los mensajes pueden ser también

archivos de imágenes que el usuario serán seleccionará de la galería de imágenes de su dispositivo móvil. Por último, debe existir la posibilidad de responder a un mensaje en concreto, y a su vez poder visualizar las respuestas que ha recibido un mensaje.

Edición de los parámetros de la sala de chat

Únicamente en las salas grupales podrá modificarse el nombre el nombre y la imagen del chat. Para la imagen se sigue el mismo proceso que para enviar las imágenes en el chat. Además un administrador del chat puede añadir integrantes o eliminarlos del chat. En el caso de los chats privados solo se podrá ver la imagen del usuario, pero no modificar ningún dato ya que estos son propios de ese usuario.

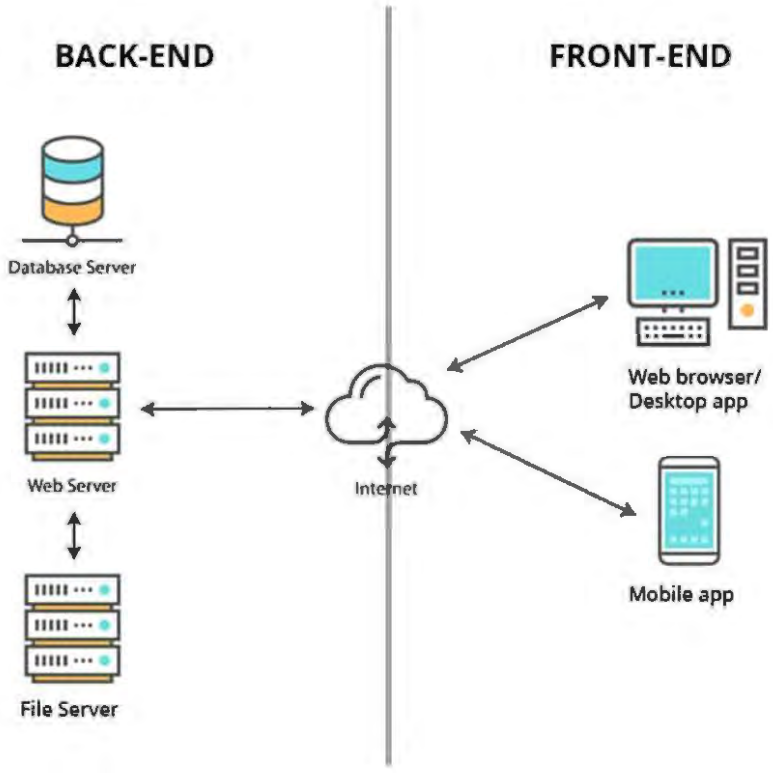
4. ARQUITECTURA DE LA SOLUCIÓN

Este apartado tiene como objetivo justificar todo aquello que ha sido necesario en el proyecto para poder realizar una correcta implementación del mismo. Es decir, se detallarán los frameworks, lenguajes de programación o entornos utilizados, entre otras cosas. También se explicará la estructura general de la aplicación.

4.1. ESQUEMA GENERAL DE LA APLICACIÓN

Ya ha sido mencionado que este proyecto consta de un Frontend y un Backend, donde el primero realiza peticiones al segundo. Esto que significa que el usuario realizará una acción desde la parte visible de la aplicación (cliente), como por ejemplo pulsar abrir un chat para leer los mensajes, y el cliente realizará una petición a la API (servidor) que devolverá los mensajes para que el usuario pueda visualizarlos en la pantalla de su dispositivo con los estilos correspondientes.

A continuación, se muestra un esquema general de una conexión Frontend y Backend, en el que se puede apreciar como un Backend puede ser llamado desde diferentes tipos de dispositivos o aplicaciones.

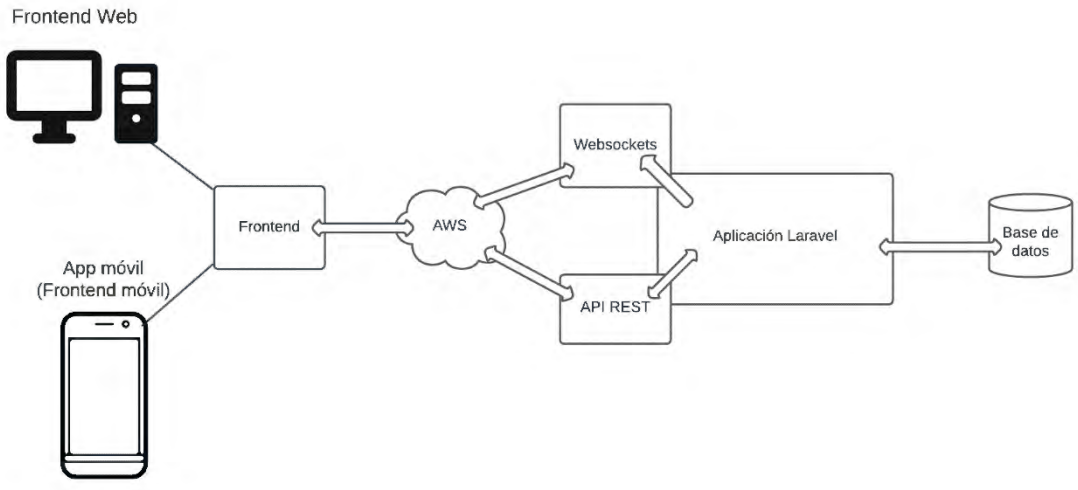


Esquema clásico de Front-Back conexión

La aplicación no guarda ningún dato en memoria a excepción de un token se seguridad que el servidor devuelve al iniciar sesión, es decir, todos los datos tienen que pedirse al servidor para ser renderizados. Los datos se pedirán mediante peticiones HTTP a la API desde el Frontend creado con React Native. Es importante destacar que a una misma API es posible realizarle peticiones desde programas diferentes, por ejemplo desde una web y una app móvil.

Imagen de un API conectada a varios dispositivos

Además de todo lo mencionado, el Backend debe estar conectado mediante un protocolo Websockets, que permitirá enviar información a los diferentes usuarios que se encuentren escuchando en el canal de difusión. Asimismo, la estructura general de la aplicación quedaría de la siguiente manera:



Esquema general de la aplicación

En él se puede apreciar un Frontend formado por dispositivos de diferente índole. Dicho Frontend se conecta a un servidor ubicado en la nube, en el que se encuentran tanto la API como el servicio Websockets manejados por la aplicación implementada en Laravel. Por último, esta aplicación se encuentra en comunicación con la base de datos del proyecto.

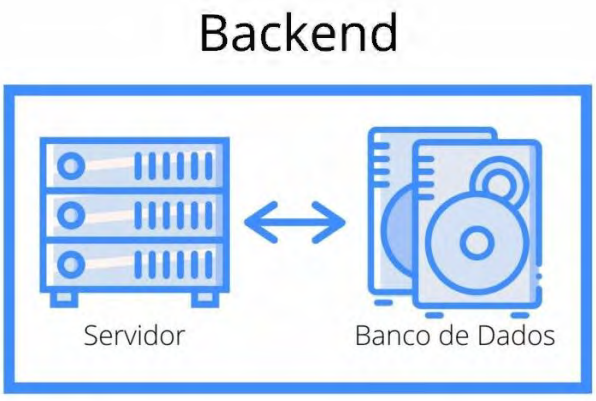
4.2. ESTRUCTURA DEL BACKEND

Si anteriormente se ha explicado que la aplicación consta de un Backend y un Frontend, en este momento pasaremos a explicar qué es el Backend y cómo está estructurado en el proyecto, es decir, con qué elementos se ha construido y qué son cada uno de esos elementos.

4.2.1. ¿QUÉ ES EL BACKEND?

El Backend de una aplicación está formada por la parte responsable del correcto funcionamiento de la lógica de la aplicación, es la parte no visible [17]. Del mismo modo, tiene la función de responder a las acciones o solicitudes del usuario, así como almacenar y gestionar la información proporcionada por él, siendo así el encargado de realizar conexiones a la base de datos. También será el gestor de librerías del servidor para, por ejemplo, comprimir imágenes.

Es, por tanto, el motor que hace que la aplicación funcione y sea segura y con una buena optimización. Una aplicación que no tenga un buen Backend puede sufrir la aparición de problemas o incluso llegar a dejar de funcionar en algunos casos, de ahí la gran relevancia del Backend en el desarrollo de aplicaciones. [18]



Representación de un Backend

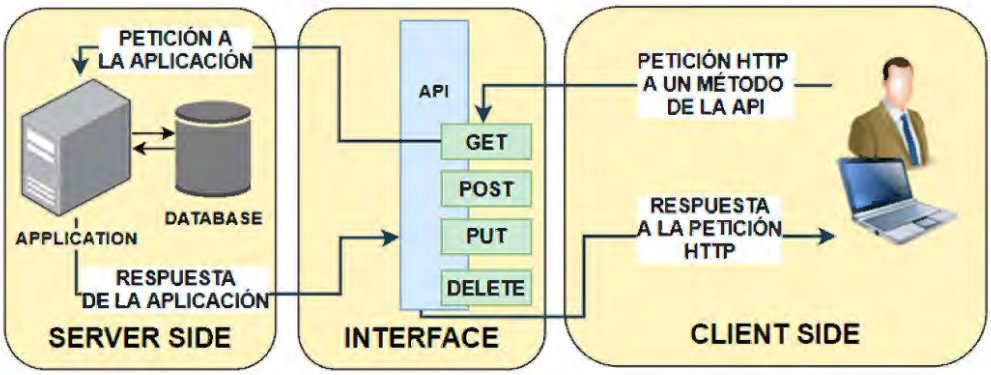
Para esta aplicación se ha desarrollado un servicio backend construyendo un API REST, de tal forma que puede utilizarse desde cualquier tipo de sistema y no solo desde una página web.

4.2.2. ¿QUÉ ES UNA API REST?

Una API REST es un Backend que posee la capacidad de responder a peticiones realizadas sobre una serie de URLs en formato JSON (un lenguaje muy sencillo de entender para máquinas y personas) y que además puede recibir información en JSON y gestionarla de manera eficiente, Puede decirse que es una interfaz de programación de aplicaciones que se ajusta a los límites de la arquitectura REST y permite la interacción con los servicios web de RESTful. Pero, ¿qué se es REST?

REST (transferencia de estado representacional) es un estilo de arquitectura de software utilizada para describir cualquier interfaz entre diferentes sistemas que se comunican mediante HTTP. Ya que el estado es representacional, cada petición que le llega al servidor es independiente de las demás, no se guardan datos entre dos llamadas cualesquiera. En esta arquitectura el cliente no necesita saber cómo está implementado el servidor ni el servidor necesita conocer cómo se están utilizando los datos desde el servidor. De este modo, si un usuario se autentificase en una petición, los datos de autentificación se perderán en la siguiente llamada que se realice, pero esto puede solucionarse mediante un token que el servidor le enviará al cliente y que el cliente tendrá que enviar al servidor en cada petición que requiera de una autenticación. Cabe también destacar que los diferentes servicios que puede realizar requieren de una dirección única para cada uno. [19]

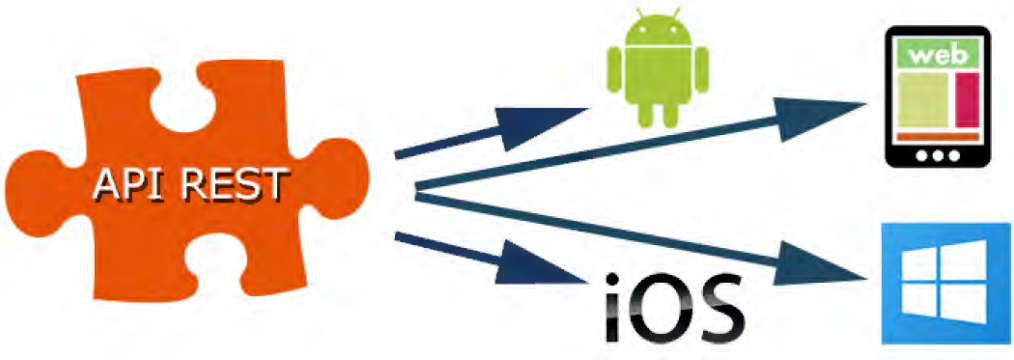
Las peticiones HTTP pueden ser de diferentes tipos según la acción que se desee realizar en el servidor, de modo que se utiliza GET para únicamente pedirle al servidor, POST para enviarle datos (también es posible recibir una respuesta a los datos enviados), PUT para actualizarlos y DELETE para borrarlos.



Peticiones HTTP a una API REST

En la actualidad el desarrollo de Backend basado en API REST es una corriente que ha ganado un gran protagonismo y es altamente utilizada por las empresas debido a su gran versatilidad, ya que pueden tener como cliente una aplicación móvil de cualquier sistema operativo, un navegador web y en definitiva cualquier dispositivo que tenga la capacidad de utilizar HTTP.

Por el contrario, el desarrollo tradicional del Backend consistía en el diseño de sistemas que devolvían al cliente código HTML para que fuera interpretado únicamente por el navegador. Pero, como es evidente, en la actualidad se consume el servicio web mediante más medios que navegadores, como pueden ser las aplicaciones móviles. Por tanto, si se desea implementar un único Backend para múltiples clientes es importante que el diseño de éste sea la parte menos cambiante, de forma que puedan incluso añadirse nuevos clientes que continuarán consumiendo el mismo Backend que los anteriores dispositivos.



Diversos dispositivos conectados a la API

Para crear una API REST pueden utilizarse diversos lenguajes y frameworks, pero para este proyecto en concreto se ha escogido el lenguaje PHP y el framework Laravel.

4.2.3. PHP Y LARAVEL

PHP (Hypertext Preprocessor) es un lenguaje de programación de código abierto y de uso general, aunque muy bien adaptado para el desarrollo web ya que tiene la posibilidad de ser incrustado en HTML. Es un lenguaje que se enfoca en generar scripts del lado del servidor, aunque con él se puede hacer prácticamente cualquier cosa. Posee una gran potencia y es muy sencillo, pero también ofrece una gran cantidad de opciones avanzadas, de modo que es un lenguaje apto tanto para los más principiantes como para los desarrolladores expertos. [21]

El uso de este lenguaje permite una gran libertad de elección del sistema operativo y del servidor web, ya que es posible utilizarlo en todos los sistemas operativos transcendentales como pueden ser Windows, macOS o las principales variantes de Unix; e igualmente en la mayoría de servidores, como Apache o IIS.

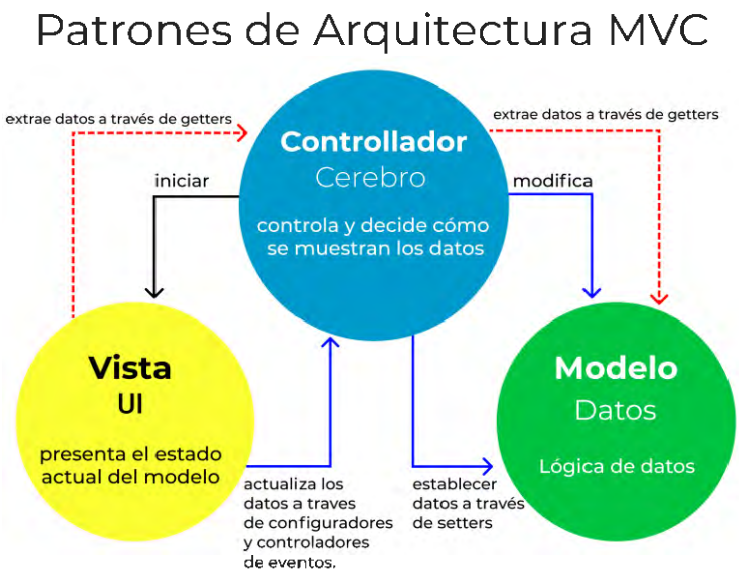
Entre las posibilidades que ofrece PHP también destaca la generación y guardado de ficheros PDF, imágenes, vídeos o archivos XML, entre otros. Del mismo modo permite soporte para una gran cantidad de bases de datos.

Por su parte, Laravel es un framework de PHP que tiene como objetivo facilitarle la vida a los desarrolladores permitiendo instanciar automáticamente clases o métodos que se utilizarán posteriormente en el código. De esta forma no hay necesidad de que el desarrollador tenga que preocuparse por cometer errores a la hora de instanciar estas clases. Es de código abierto y tiene una curva de aprendizaje relativamente sencilla permitiendo además desarrollar PHP de forma muy intuitiva, lo que lo convierte en una buena opción para los desarrolladores menos experimentados. Destaca su facilidad de mantenimiento y escalabilidad, permitiendo el desarrollo de proyectos de toda índole. Se puede usar para tanto para desarrollo full stack como para desarrollo de APIs, que es de la manera en la que se ha utilizado para la elaboración de este proyecto. Como consecuencia de todo lo mencionado, Laravel se ha convertido en uno de los frameworks más utilizados por la comunidad.

En palabras de sus creadores, Laravel se define como: “un marco de aplicación web con una sintaxis elegante y expresiva. Creemos que el desarrollo debe ser una experiencia divertida y creativa para ser verdaderamente satisfactorio. Laravel intenta aliviar el dolor del desarrollo facilitando las tareas comunes que se utilizan en la mayoría de los proyectos web”. [22]

Posee una serie de características muy interesantes permitiendo así infinidad de posibilidades a la hora de desarrollar. A continuación, se comentarán las más relevantes de estas características: [23]

- En primer lugar, debe ser destacada su arquitectura MVC (modelo-vista-controlador) que es un estilo de arquitectura avanzada de carpetas que le permite al desarrollador organizar la aplicación de forma clara separando los datos y la lógica de una aplicación de su representación y de la parte responsable de la gestión. Esto resulta muy útil a la hora de trabajar en equipo, ya que una buena organización es crucial para que otro integrante del equipo pueda continuar con el trabajo.



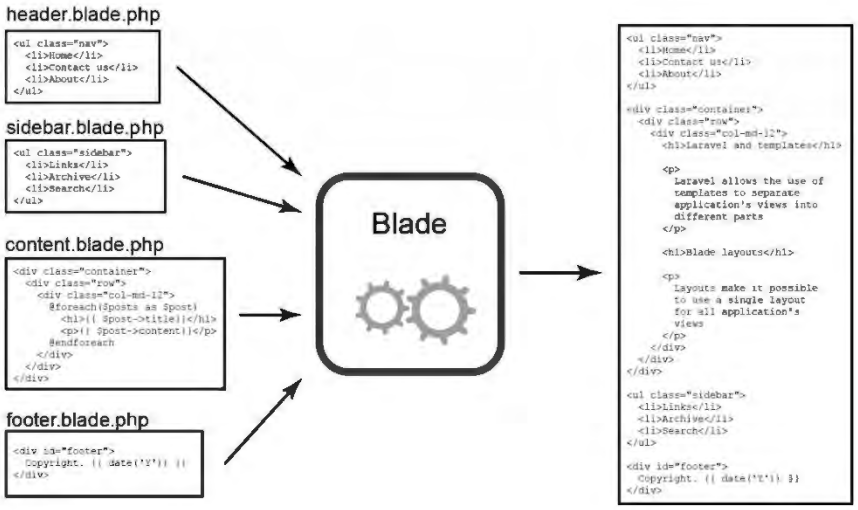
Esquema modelo-vista-controlador

- También es importante la inclusión de la consola Artisan, una interfaz que permite ejecutar un gran número de funcionalidades desde la línea de comandos. Esto le concede a los desarrolladores una gran facilidad para crear controladores, bases de datos o ver las rutas de la aplicación.

```
make:command      Create a new command class
make:console      Create a new Artisan command
make:controller   Create a new resource controller class
make:event        Create a new event class
make:job          Create a new job class
make:listener     Create a new event listener class
make:middleware   Create a new middleware class
make:migration    Create a new migration file
make:model        Create a new Eloquent model class
make:policy       Create a new policy class
make:provider     Create a new service provider class
make:request      Create a new form request class
make:seeder       Create a new seeder class
make:test         Create a new test class
```

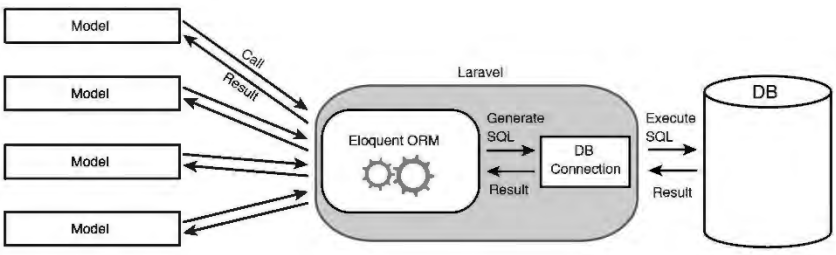
Principales comandos de Artisan

- Otro punto relevante es el motor de plantillas de Laravel, conocido como Blade. Este motor compila las plantillas en código PHP y le otorga al programador la capacidad de crear páginas web con una gran potencia. Esta es la causa por la que Laravel puede utilizarse como framework para desarrollo full stack, aunque esto no es muy relevante en este proyecto, ya que el Frontend es una aplicación móvil que hecha con React Native.



Composición de las plantillas Blade

- Laravel incluye además Eloquent ORM (mapeador relacional de objetos) que le asigna un modelo a cada base de datos con el fin de hacer muy sencilla la interacción con las bases de datos. Estos modelos convierten a Laravel en un framework intuitivo para realizarle las habituales consultas a las bases de datos (recuperar, eliminar, insertar o actualizar). En este punto es relevante mencionar que Laravel permite actualizar y migrar las bases de datos sin tener que crearlas nuevamente tras haberse producido variaciones en el desarrollo.

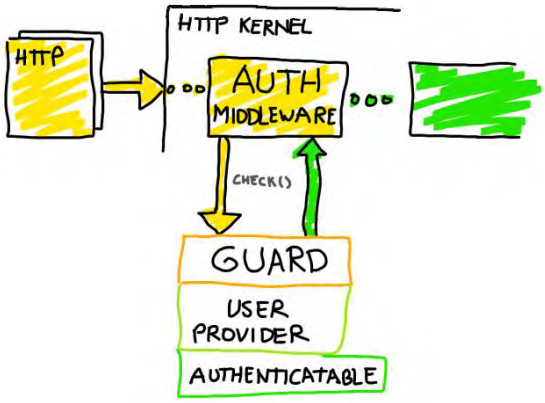


Modelo Eloquent

- Igualmente es importante resaltar los middlewares de Laravel, que se encargan de crear puentes entre aplicaciones independientes entre sí. Esto resulta muy útil para la gestión de las peticiones HTTP, aunque no es

estrictamente necesario utilizarlos para conseguir gestionar de manera correcta dichas peticiones.

- Como última característica principal de Laravel hay que hablar de la autenticación y la autorización que proporciona a la hora de desarrollar. Éste es uno de los factores más relevantes para el uso de Laravel como framework de desarrollo de APIs, y por lo tanto es un punto muy importante para este proyecto. Su sistema de autenticación consta de “guards” y “providers”, donde los primeros se encargan de autenticar a los usuarios cada vez que realizan una solicitud y los segundos tienen la responsabilidad de recuperar los usuarios que se hallan almacenados. Este sistema permite la creación de un token de seguridad que se le devolverá al Frontend cuando un usuario se autentifique. Después se podrán configurar rutas que requieran de ese token de seguridad porque serán exclusivas para cada usuario. En cuanto al sistema de autorización, Laravel permite filtrar a los usuarios de modo que aunque uno esté autentificado no pueda realizar ciertas acciones, esto es particularmente útil para esta aplicación ya que permite con mucha facilidad que los usuarios con el rol de profesor tengan ciertos privilegios con respecto a los alumnos.



Funcionamiento de autenticación

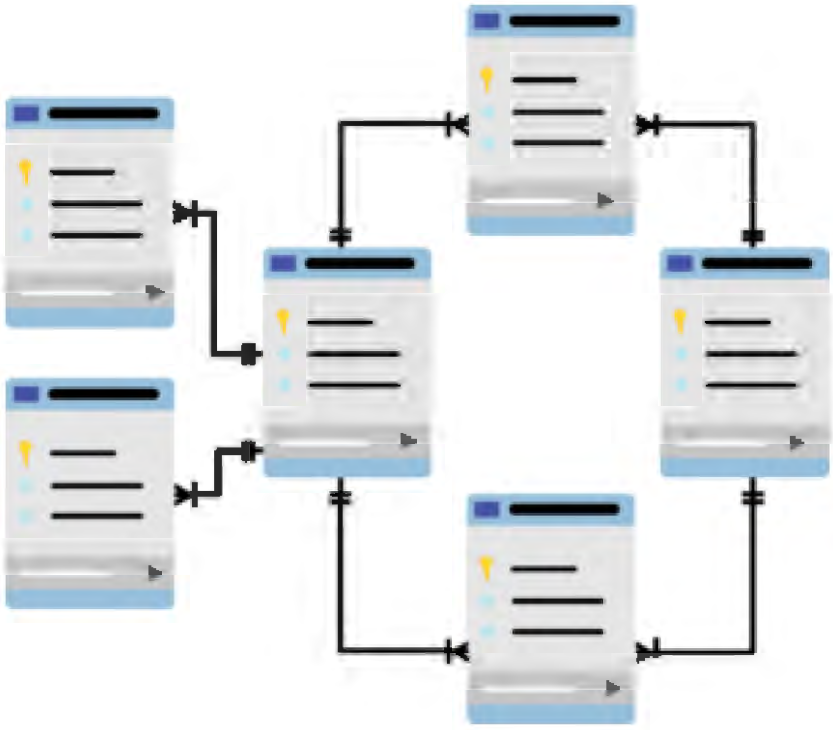
4.2.4. ¿QUÉ ES MYSQL?

MySQL es un sistema de código abierto de gestión de bases de datos relacionales con un modelo cliente-servidor. Está desarrollado y soportado por Oracle, y su desarrollo está basado en ANSI, C y C++. [24] Para una correcta compresión del funcionamiento de este sistema resulta imprescindible entender qué es una base de datos relacional y qué es el modelo cliente-servidor.

Antes de explicar lo que es una base de datos relacional es importante conocer la definición de base datos. Una base de datos es una recopilación organizada de información o datos estructurados, que generalmente se almacena en un sistema informático, y que además suelen estar controladas por un sistema de gestión de bases de datos (DBMS). Los datos recopilados se organizan en filas

y columnas formando así las tablas de las bases de datos. De esta forma se pueden organizar y consultar más cómodamente los datos. Según cómo se organicen sus datos las bases de datos pueden ser relacionales o no relacionales. [25]

Las bases de datos relacionales serán por lo tanto aquellas que almacenan y proporcionan acceso a puntos de datos relacionados entre sí. [26] Cada fila de una tabla representa un registro que tendrá un identificador (ID) único que se conoce como clave primaria de la tabla. Por su parte, las columnas contienen los atributos de los datos, y así cada registro de la tabla contendrá un valor. De esta forma se pueden establecer las relaciones entre tablas de una forma muy sencilla. Las relaciones se establecen haciendo coincidir datos en columnas de tablas diferentes que generalmente tienen el mismo nombre, para definir esta relación se crea una conexión entre la clave primaria de una tabla y un campo de la otra tabla, que pasará a denominarse clave foránea en esa tabla.

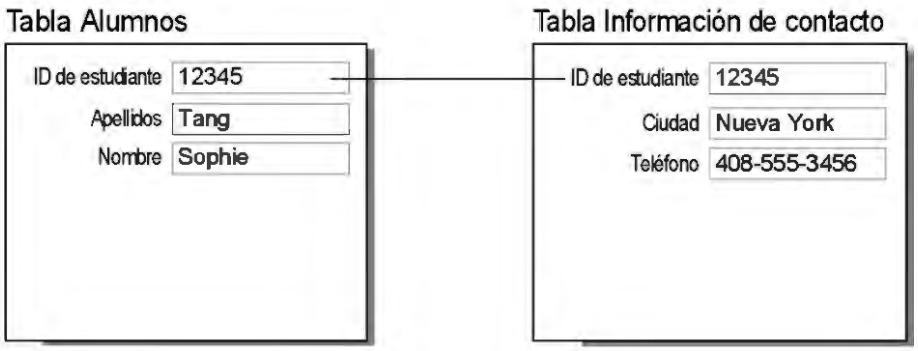


Esquema de una base de datos relacional

Las relaciones que se establecen en una base de datos relacional pueden ser de tres tipos diferentes:

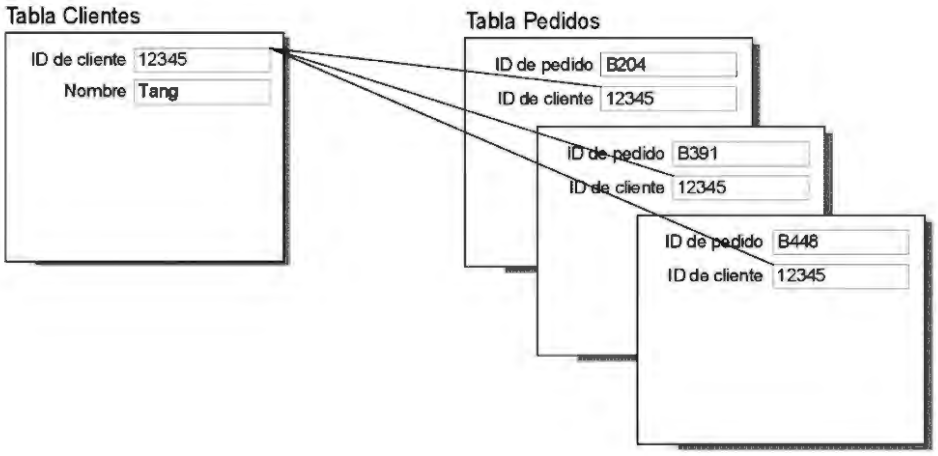
- Relación unívoca (uno a uno). Este tipo de relación se da cuando una fila de una tabla solo puede tener coincidir con una de otra tabla, y la fila de la segunda tabla solo puede igualmente coincidir con una fila de la primera. Es muy poco habitual porque podría agruparse toda la información en una misma tabla. Sin embargo, se puede dar en casos excepcionales como en un

almacenaje de datos de corta duración para así poder deshacerse de ellos con mayor facilidad, división de las columnas de una tabla, o simplemente por motivos de seguridad.



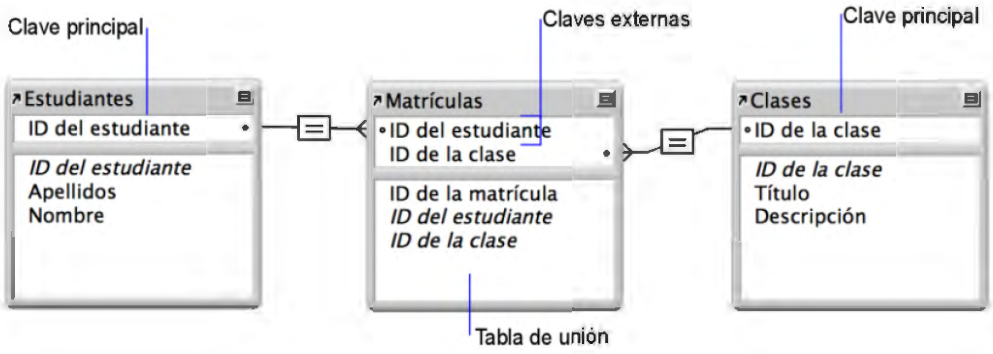
Ejemplo relación uno a uno

- Relación uno a muchos. Se establece cuando una fila de una tabla puede tener muchas filas coincidentes en otra tabla, pero a su vez una fila de la segunda tabla solo puede coincidir con una fila de la primera. Es la relación más habitual en las bases de datos relacionales.



Ejemplo relación uno a muchos

- Relación muchos a muchos. En este caso una fila de una tabla puede tener muchas filas coincidentes en otra tabla y la fila de la segunda tabla también puede tener muchas filas coincidentes en la primera. Se necesita una tabla pivote intermedia para poder crear establecer este tipo de relación. [27]



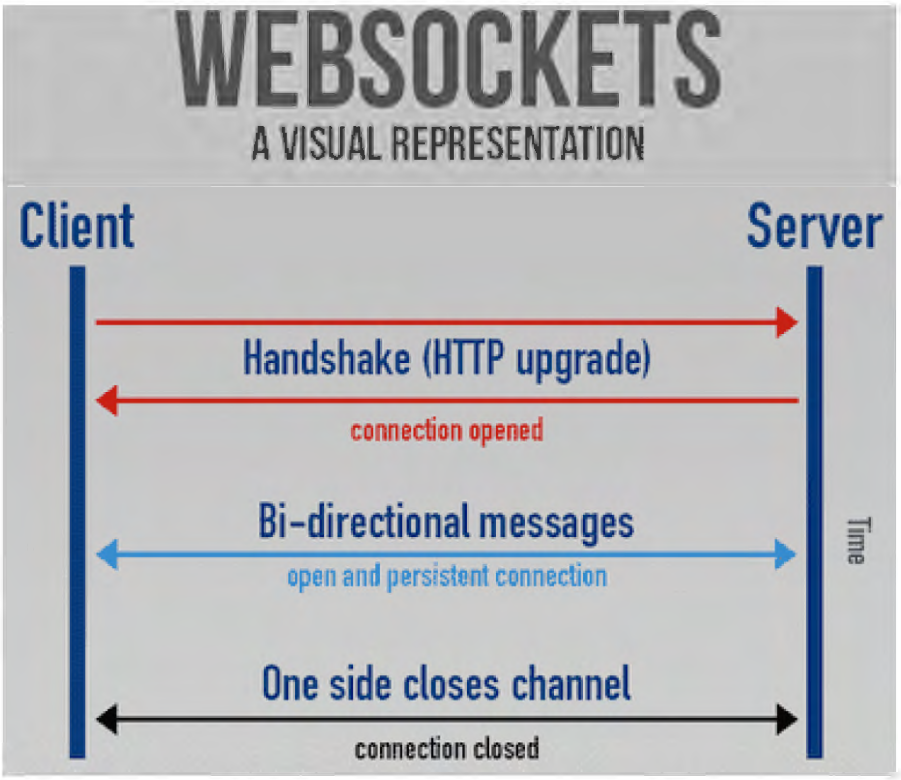
Ejemplo relación muchos a muchos

En relación al modelo cliente-servidor que sigue MySQL se puede decir que consiste en un cliente que le envía peticiones SQL al servidor, donde evidentemente se encuentran los datos almacenados. En función de cómo sean esas peticiones el servidor realizará unas u otras operaciones. Estas operaciones pueden ser una consulta de datos para únicamente solicitar información, una manipulación de datos para realizarle modificaciones a la base de datos, un cambio de identidad de datos para ajustar el tipo de los datos, o un control de acceso a los datos como medida de seguridad de protección restringiendo así el acceso a la base de datos.

Además de todo lo mencionado también ha de ser resaltada su facilidad de uso, así como su seguridad, ya cuenta con un sistema de privilegios de acceso y de administración de cuentas de usuario. Todo esto sumado a su excelente rendimiento y su compatibilidad con Laravel explica por qué ha sido utilizado en este proyecto.

4.2.5. WEBSOCKETS

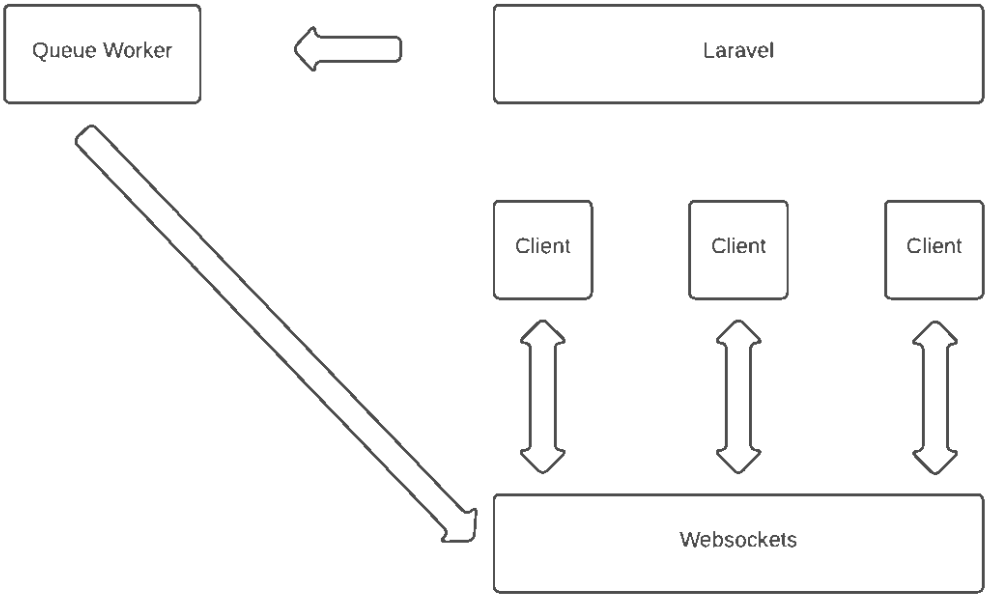
Implementando todo lo explicado en los puntos anteriores podría construirse un Backend para recibir peticiones por parte del Frontend, pero no sería posible establecer una comunicación en tiempo real entre dos o más usuarios. Y es ahí donde entra Websockets, un protocolo capaz de crear una red de intercomunicación para dotar al usuario de una comunicación directa e interactiva con el servidor. [28] De esta forma es posible que un usuario reciba un evento del servidor únicamente habiendo abierto un canal de comunicación, es decir no necesita realizar ninguna petición al servidor para que este le envíe la información correspondiente. Esto resulta imprescindible para la creación de un chat, ya que al abrir un usuario una sala de chat estaría abriendo un canal de comunicación directa con el servidor y podría recibir los mensajes que los otros usuarios del chat hayan enviado al servidor sin la necesidad de realizarle ninguna petición al servidor.



Representación Websockets

Para este proyecto no es necesaria una comunicación bidireccional entre los usuarios, es suficiente con que un usuario envíe un mensaje al servidor mediante una petición HTTP y sea el servidor el que tenga una comunicación directa con el cliente o clientes en cuestión. Así un cliente que abra una sala de chat y envíe un mensaje lo hará con una petición POST al servidor para enviarle los datos del mensaje en cuestión, el servidor los guardará y a su vez se lo hará llegar a todos los clientes que se encuentren suscritos al canal correspondiente. Es decir, un cliente envía un mensaje y el servidor lo difunde a los usuarios que sea necesario.

Para implementar este proceso encaja perfectamente Laravel ofreciendo la posibilidad de hacer broadcasting a través de la librería Laravel Echo y de Pusher, o lo que es lo mismo, permitiendo una difusión de información a los clientes suscritos a los canales por los que se difunde dicha información. Los canales de comunicación pueden ser públicos, donde la información se difundirían a través de todos los usuarios. Pero en este caso resulta evidente que los canales han de ser privados, ya que los mensajes deben llegar única y exclusivamente a los usuarios deseados. [29]



Ejemplo de broadcasting con Laravel

Por un lado Pusher es un servicio de Websockets que ejerce con intermediario entre los clientes y el servidor. Cuando Pusher recibe un mensaje lo difunde a los clientes que se encuentren suscritos esperando recibir el mensaje.

Y por otro lado, Laravel Echo es una librería de JavaScript responsable de permitir a los usuarios indicar a qué canal suscribirse y de este modo enviarles la información a través de eventos.

4.3. ARQUITECTURA DEL FRONTEND

Este apartado será análogo al anterior, en él se explicará en qué se entiende por Frontend y cómo queda estructurado en el proyecto.

4.3.1. ¿QUÉ ES EL FRONTEND?

El concepto de Frontend es radicalmente opuesto al de Backend, es la interfaz de una aplicación ya sea web o móvil y es la que interactúa con el usuario, es por lo que se dice que está del lado del cliente. Es el diseño de la aplicación, y abarca desde la estructura hasta los colores o animaciones.

Pero que sea la parte del diseño no quiere decir que no se utilice código, sino todo lo contrario, ya que dentro del desarrollo de un mismo Frontend se utilizan lenguajes de marcado y estilo y lenguajes de programación. Los principales lenguajes de estilo y marcado son HTML y CSS, y el más utilizado como lenguaje de programación es JavaScript. Per de ellos derivan un notable número de frameworks y librerías como Bootstrap, React, o SASS. [30]

Concepto de Frontend

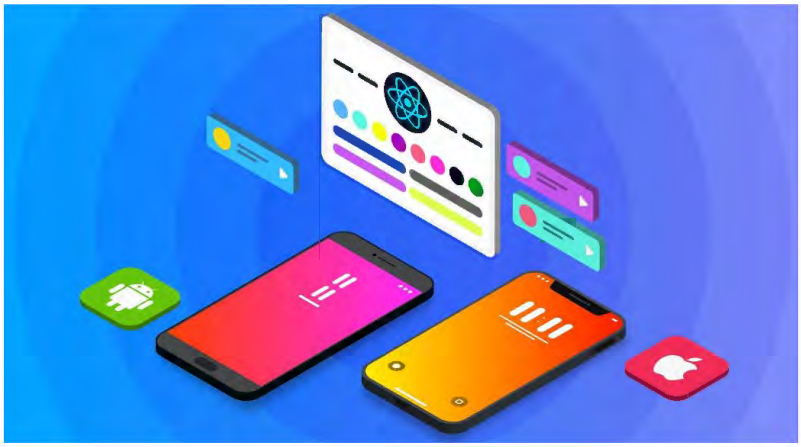
Evidentemente en esta parte de la aplicación es donde se le pedirán los datos al Backend y serán recibidos y representados por pantalla. Por lo tanto, el Frontend debe estar preparado para la recepción y representación de esos datos. Por ejemplo, si la imagen de perfil del usuario se encuentra en el servidor, el Frontend ha de ser capaz de realizarle un petición al servidor para que le devuelva la imagen, ya sea implícitamente o mediante la acción del usuario, y después recibirá los datos de la imagen y la representará pasándole los datos al componente encargado de representar la imagen.

Ya que el Frontend desarrollado en este proyecto se trata de una aplicación móvil, es importante también mencionar el sistema de navegación de pantallas, que le permitirá al usuario desplazarse por las diferentes pantallas de la aplicación y en los casos necesarios pasar datos entre pantallas.

4.3.2. REACT NATIVE

React Native es un framework basado en React utilizado con el fin de poder desarrollar aplicaciones nativas multiplataforma. Permite diseñar componentes

web pero con una preparación para que puedan funcionar directamente en los diferentes sistemas operativos de los dispositivos móviles de tal manera que sería prácticamente imposible diferenciar una aplicación desarrollada en Android nativo con una aplicación para Android diseñada con React Native; y lo mismo ocurriría con el sistema operativo iOS.



React Native como multiplataforma

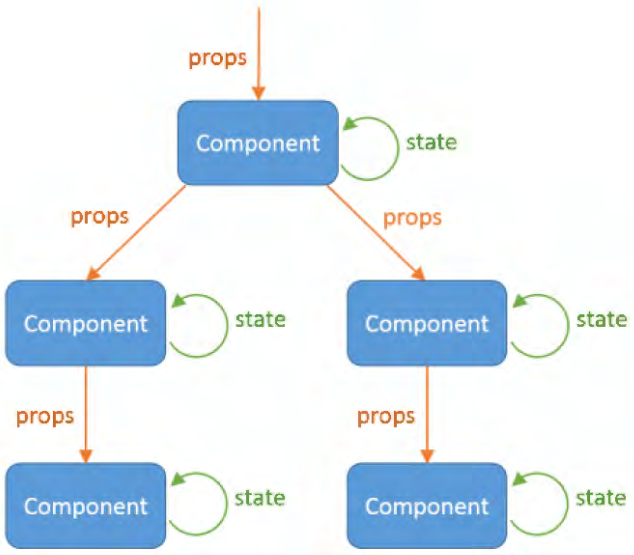
Por su parte, React es una librería de JavaScript para construir interfaces de usuario interactivas de forma clara y sencilla. Permite el diseño de vistas simples y es responsable de actualizar y renderizar de manera eficiente los componentes correctos cuando los datos cambien. Se basa en la programación declarativa, esto quiere decir que funciona a un alto nivel de abstracción, permitiendo una buena optimización y una gran facilidad a la hora de depurar el código. [31]

Volviendo a React Native es relevante decir que usa el mismo paradigma fundamental de construcción de bloques de UI (componentes visuales con los que interacciona el usuario) que las aplicaciones nativas reales de Android e iOS, pero gestiona la interacción entre los mismos utilizando las capacidades de JavaScript y React. [32]

De modo que podemos afirmar que el fundamento de programación de este framework se basa en la creación de componentes. Incluye por defecto algunos componentes como vistas, textos, botones, sliders o imágenes. Para utilizar estos componentes solamente deben ser importados en el archivo en el que se deseen usar. Pero lo más importante es que el desarrollador puede crear sus propios componentes y reutilizarlos, incluso con componentes creados se puede crear otros componentes mayores. Por ejemplo, si creásemos un componente “mesa” formado por un componente “tablero” y cuatro componentes “pata”, podríamos crear después un componente “salón” formado por varios componentes “mesa”.

Componentes de React Native

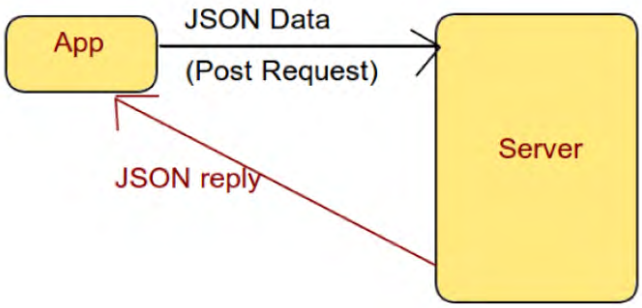
Además estos componentes contienen ciertas propiedades denominadas “props”. Los componentes tienen algunas props por defecto, pero al igual que pasaba con los componentes también se le pueden crear props personalizadas. Esto resulta especialmente útil para pasar variables o funciones entre componentes, así como parámetros que definan su estilo. De esta forma, se puede crear un componente y en función de las props que reciba poder crear tantos componentes diferentes a partir del primero como se deseen. Siguiendo el anterior ejemplo del componente mesa, si se le asignaran props de forma, dimensión y color podrían crearse una gran variedad de mesas a partir del componente padre.



Flujo de información a través de los componentes

En lo relativo a los estilos es interesante hablar de un componente en concreto, denominado StyleSheet. Este componente es una abstracción similar a las hojas de estilo CSS. Para utilizarlo se crea una constante, normalmente llamada styles, del tipo StyleSheet y se crearán dentro las variables correspondientes a los estilos de cada componente que se encuentre en el archivo. Después sólo habrá que pasarle como prop la variable de estilo al componente correspondiente, para ello los componentes incluyen por defecto la propiedad style. Pero esto no es todo, ya que es posible añadir estilos a un componente que haya sido creado con un determinado estilo, para así mejorar la capacidad de reutilización de los componentes.

También es importante destacar que React Native incluye una conectividad que permite hacer peticiones HTTP a una API REST, esto es perfecto para este proyecto ya que prácticamente toda la información de los usuarios debe ser cargada del servidor. Se pueden realizar las peticiones de diversas formas, como por ejemplo con el método Fetch incluido JavaScript. Para ejecutar esta sentencia es necesario pasarle la URL a la cual se hará la petición, así como el método, y cuando sea necesario, la cabecera y el contenido. Tras la realización de una petición permite también manejar la respuesta a dicha petición. Evidentemente todas estas peticiones serán asíncronas.



Petición HTTP al servidor

Una de las grandes ventajas es que permite a los desarrolladores de aplicaciones móviles crear aplicaciones de alto rendimiento en ciclos de desarrollo más cortos y tiempos de implementación más rápidos. Otro gran detalle es que proporciona animaciones extremadamente suaves ya que el código se convierte en vistas nativas antes de ser renderizado. Además, React Native es altamente compatible con complementos de terceros, como Google Maps. Si a todo esto le sumamos la sencilla curva de aprendizaje que tiene y la gran comunidad que lo respalda hace que se haya convertido en una de las tecnologías preferidas entre los desarrolladores de muchas pequeñas y

grandes empresas, y por supuesto también de los desarrolladores independientes.

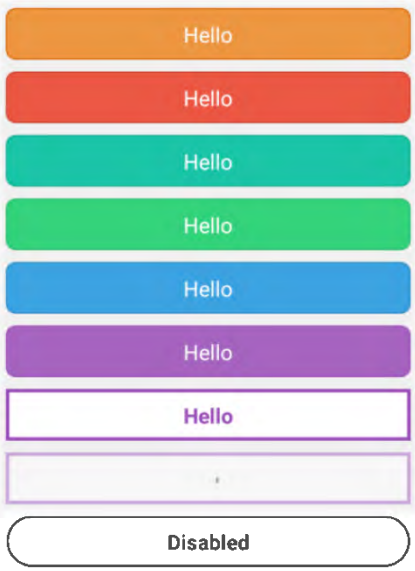
4.3.3. CLASES DE COMPONENTES EN REACT NATIVE

Los componentes de React Native se pueden dividir principalmente en los siguientes tipos:

- Componentes **básicos**. En este apartado entrarían componentes como View que es el equivalente a un div en HTML, Text para imprimir textos por pantalla, Image para sacar imágenes por pantalla, TextInput para crear formularios o ScrollView que permite hacer scroll en la pantalla en la que se incluya además de contener varios componentes.

Ejemplo del componente TextInput

- Componentes **de interfaz de usuario**. En este grupo pertenecen algunos como Button que genera un botón capaz de hacer la función que tenga asignada cuando el usuario la presione, Picker para seleccionar un elemento entre varios de una lista, Slider para ajustar el valor de un parámetro en un rango de valores o Flatlist que crea una lista con un scroll propio para representar una cierta cantidad de objetos como si representasen mediante un bucle.



Ejemplos del component Button

- Componentes **exclusivos** de un sistema operativo. Aquí entran elementos que solo pueden ser utilizados en un sistema operativo, y si se desea desarrollar una aplicación para más de un sistema operativo no funcionará en el otro.

- **Otros**. Aquí pertenecen los componentes que se encargan de tareas específicas como Webview, Alert, etc.

También es importante destacar que aparte de los componentes que propiamente proporciona React Native en su documentación, hay otros muy interesantes creados por la comunidad que podemos instalar y utilizar en cualquier proyecto.

4.3.4. DEPENDENCIAS EXTERNAS UTILIZADAS

Además de las librerías y paquetes que vienen instalados por defecto en React Native se pueden instalar otros. Algunos son obligatorios para el correcto funcionamiento y compilación de la aplicación, como puede ser NPM o en su defecto Yarn. Otros acaban siendo imprescindibles como el que se encarga de la navegación entre las diferentes pantallas de la aplicación. Y por último, existen otros que simplemente le facilitan la vida al desarrollador a la hora de escribir código o de crear componentes. Las más importantes que se han utilizado para este proyecto se detallan a continuación.

npm

Node Package Manager (npm) es un gestor de paquetes viene incluido con los proyectos cuyo desarrollo está relacionado con Node. Por su parte, Node es un entorno en tiempo de ejecución multiplataforma muy utilizado en los proyectos relacionados con JavaScript. [33]

Este tipo de gestores resultan imprescindibles para este tipo de proyectos porque abarca funcionalidad desde la inicialización de un proyecto hasta la ejecución y compilación del mismo, pasando por la instalación de módulos y dependencias. Y además permite todo esto desde la ventana de comandos.

Existen alternativas a npm como yarn, que tiene prácticamente las mismas funcionalidades, y normalmente en la documentación de los paquetes a instalar se proporciona un método de instalación con npm y otro con yarn.

Cabe destacar que los paquetes que se explicarán a continuación han sido instalados en el proyecto gracias a npm.

Representación de npm como gestor de paquetes

Lodash

Lodash es una librería de JavaScript que tiene como objetivo facilitarle la vida al desarrollador simplificando el manejo de objetos o listas. Esto permite la creación de un código mucho más claro, por lo que será más fácil de escribir y mucho más sencillo de depurar para el desarrollador. [34]

Contiene infinidad de funciones, así vamos a nombrar sólo algunas de las más importantes destacan algunas como filter para filtrar los elementos de una lista según la condición deseada, find para encontrar el primer elemento de una lista que cumpla la condición dada, uniq para borrar los elementos duplicados de una lista y quedarse así con elementos únicos, difference para calcular la diferencia entre dos vectores, compact para eliminar elementos nulos o indefinidos en una lista, size para obtener el tamaño de una lista, o map para recorrer todos los elementos de una lista.

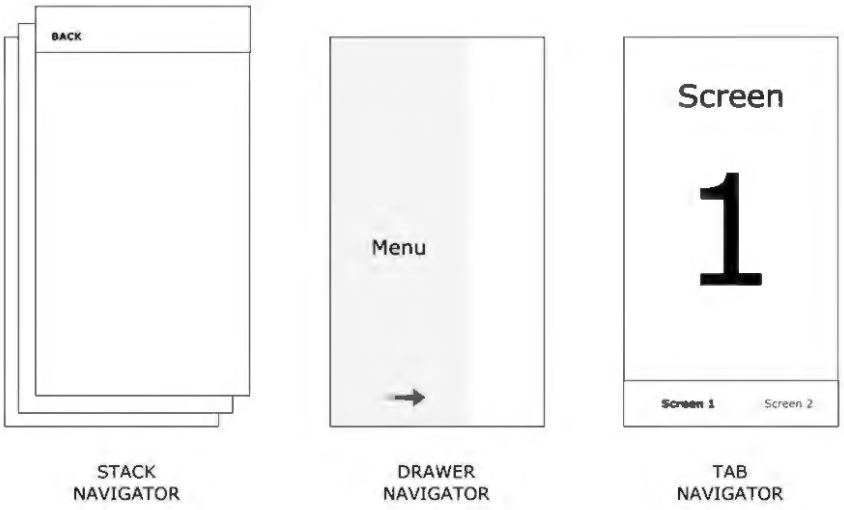
```
function square(n) {
  return n * n;
}

_.map([4, 8], square);
// => [16, 64]

_.map({ 'a': 4, 'b': 8 }, square);
// => [16, 64] (iteration order is not guaranteed)

var users = [
  { 'user': 'barney' },
  { 'user': 'fred' }
];

// The `_.property` iteratee shorthand.
_.map(users, 'user');
// => ['barney', 'fred']
```

Ejemplo de la función map

React Navigation

Esta dependencia es la que permitirá que se pueda navegar entre las diferentes pantallas de la aplicación. Se pueden crear con ella componentes de navegación muy interesantes con poco esfuerzo. Incluye por defecto ciertos componentes de navegación como pueden ser un menú lateral o un menú de tabs en el que habrá que crear las pantallas en el menú. Pero al final, debe ser el desarrollador el que monte el sistema de navegación que desee para su aplicación, pudiendo utilizar por supuesto los componentes que ofrece React Native. [35]

También es posible pasar parámetros entre las diferentes pantallas de navegación y editar la barra superior de la aplicación en cada pantalla pudiendo incluir los componentes necesarios.

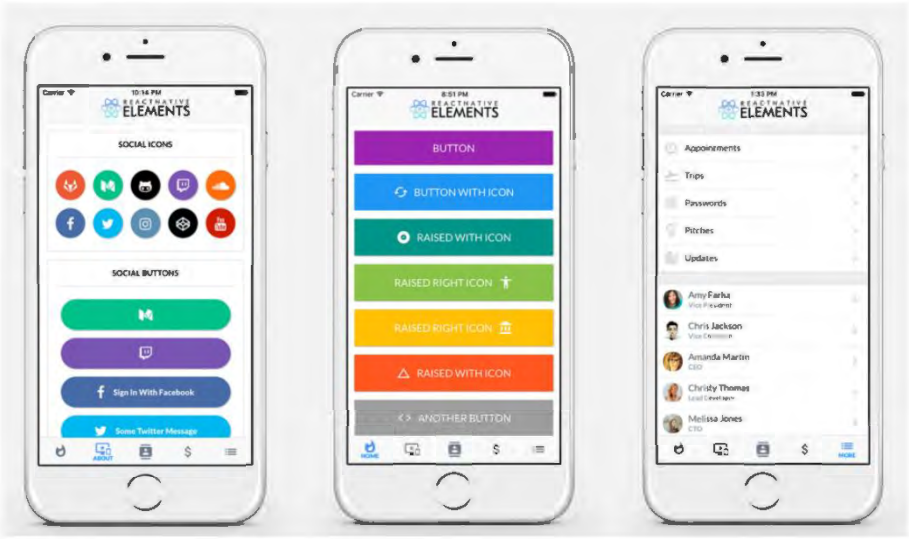


Ejemplos de navegación

React native elements

React Native Elements es una librería de interfaz de usuario multiplataforma desarrollada con Javascript. Según sus propios autores esta librería se basa más en la estructura de los componentes que ofrece que en el diseño de estos. Por lo tanto, esta parte del diseño quedará en manos del desarrollador que utilice la librería en su proyecto. Y esto permite que cada desarrollador pueda tener componentes funcionales pero con una personalización única con muy poco esfuerzo. [36]

Está desarrollada para la comunidad, siendo obviamente de código abierto. Todo esto la convierte en una de las librerías más utilizadas en React Native, siendo uno de los proyectos de este tipo que cuenta con más estrellas en github.



Ejemplos de componentes ofrecidos por React Native Elements

4.4. HERRAMIENTAS

Para poder realizar el proyecto han sido necesarios ciertos programas como editores de código que permitan escribir el código y ejecutarlo, herramientas de diseño o herramientas que brinden la posibilidad de realizar pruebas de envío de peticiones al Backend. A continuación se hará una breve explicación de cada una de las herramientas utilizadas.

4.4.1. MYSQL WORKBENCH

MySQL Workbench es una herramienta destinada a la arquitectura, desarrollo y administración de bases de datos. La herramienta proporciona modelado de datos, desarrollo de SQL y herramientas de administración integrales para la configuración del servidor, la administración de usuarios o las copias de seguridad. [37]

Logotipo de MySQL Workbench

Para el caso que nos ocupa únicamente se ha utilizado para el realizar el diseño de la base de datos. Para realizar el diseño el software permite crear las tablas con sus respectivos campos y también asignar el tipo de dato a cada uno de los campos. Además es posible asignar una relación entre tablas. Crear el diseño con este programa facilitará la tarea de crear la base de datos en Laravel.

4.4.2. VISUAL STUDIO CODE

Visual Studio Code es un editor de código fuente desarrollado por Microsoft para Windows, macOS y Linux. Es un editor ligero pero muy potente. Incorpora soporte para leguajes como JavaScript, TypeScript y NodeJS, además permite la instalación de extensiones para otros lenguajes como pueden ser Python, C++, Java o PHP. Contiene también múltiples opciones interesantes tales como, el control de versiones con Git o la posibilidad de depurar de código. [38]

Logotipo de Visual Studio Code

En este proyecto se ha utilizado como editor de código principal, ya que permite desarrollar proyectos de React Native si se instalan las herramientas y los paquetes necesarios, y asimismo permite desarrollar el código de Laravel que se utilizará como Backend.

Cuenta además con una línea de comandos propia que facilita la instalación de herramientas utilizadas en el proyecto, y además permite la instalación de extensiones que se han utilizado para facilitar la escritura del código del proyecto.

4.4.3. EXPO GO

Expo Go es creado con el objetivo de poder cubrir informáticamente todas las fases de un evento en una única plataforma, en la cual los organizadores de eventos y clientes puedan acceder a la información desde cualquier lugar y en cualquier momento.

Expo es un conjunto de herramientas, librerías y servicios que permiten desarrollar apps nativas en iOS y Android escritas en JavaScript. Expo utiliza

Expo SDK, que es una librería de JavaScript que se encarga de proporcionar acceso a las funcionalidades del dispositivo sin la necesidad de modificar el código. Es relevante decir que es capaz de visualizarse en cualquier dispositivo que tenga la app de Expo Go instalada. [39]

Logotipo de Expo Go

En este proyecto se ha utilizado esta herramienta para poder utilizar la aplicación en un dispositivo iOS sin la necesidad de generar un archivo ipa (equivalente a los apk en Android). Para utilizarlo es necesario tener instalado expo en el proyecto, ejecutar un comando en la ventana de comandos e instalar la aplicación en el dispositivo iOS para poder visualizar los resultados.

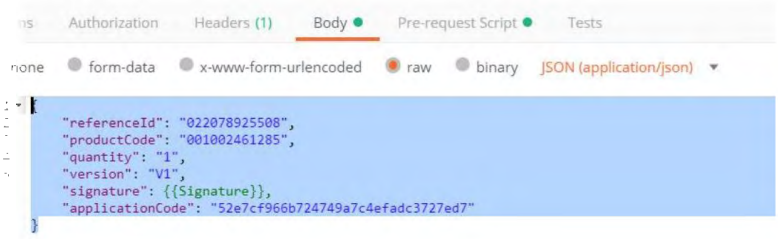
4.4.4. POSTMAN

Postman es una plataforma de código abierto que permite la gestión de APIs de forma sencilla y eficiente gracias a la posibilidad de realizar peticiones HTTP que ofrece.

Tiene múltiples finalidades entre las que destacan la realización de test de APIs creadas o de terceros o la gestión del ciclo de vida de las APIs. Permite realizar peticiones con los métodos GET, POST, PUT o DELETE.

Logotipo de Postman

Se ha utilizado en este proyecto para probar que la API construida funciona correctamente y también para ver la respuesta en formato JSON que nos brinda el servidor y así poder realizar una correcta organización de dicha respuesta en el Frontend, que será el lugar en el que realmente se realizará la petición.



Ejemplo de respuesta en JSON en Postman

5. IMPLEMENTACIÓN

Llegados a este punto ya se ha explicado en qué consiste el proyecto, cómo funciona y de qué elementos consta. Queda, por lo tanto, detallar cómo se ha desarrollado el proyecto para su correcto funcionamiento.

Y eso es de lo que tratara este apartado. De nuevo estará separado en Backend y Frontend, y para cada uno se explicará cómo se ha creado y estructurado el proyecto, y se describirán los elementos que forman parte de la estructura de ambos, así como la relación presente entre ambas partes. También se mostrarán los resultados de cada parte.

5.1. IMPLEMENTACIÓN DEL BACKEND

A modo de resumen, el Backend de esta aplicación es una API REST diseñada con Laravel con la intención de que se puedan conectar a él Frontends de diversos tipos y provenientes de diferentes dispositivos. La API implementará una base de datos relacional hecha con MySQL.

5.1.1. DISEÑO DE LA BASE DE DATOS

El diseño de la base de datos es una parte muy importante, pues será el lugar en el cual se decidirá cómo está compuesta internamente la aplicación. Habrá que decidir cómo se relacionan los usuarios con las diferentes salas de chats que tienen, con las asignaturas a las que pertenece, los mensajes que tiene, etc.

En la siguiente figura se puede apreciar que el diseño es bastante complejo, es por ello que se explicará en qué consiste cada tabla y cómo están relacionadas entre sí dichas tablas.

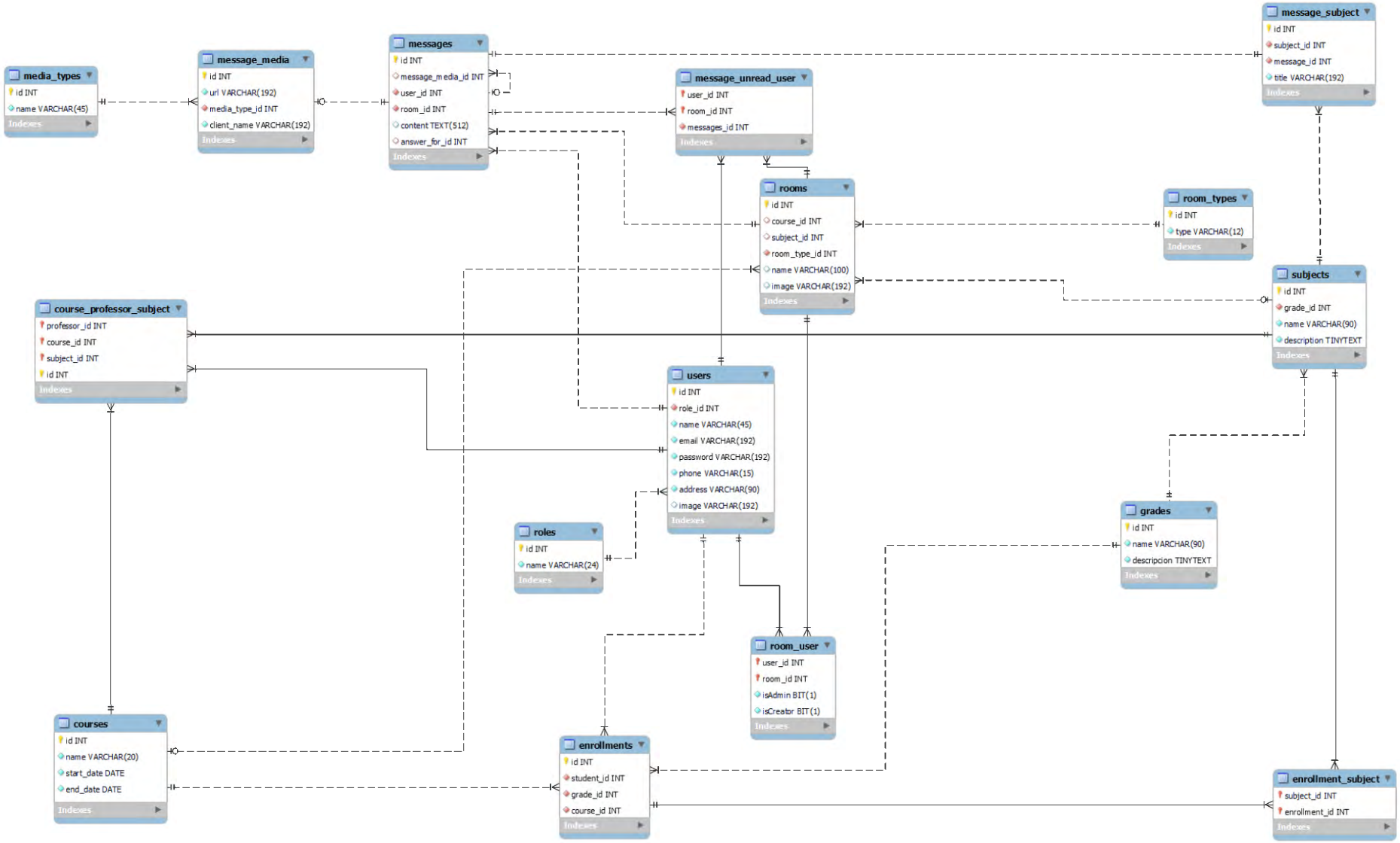
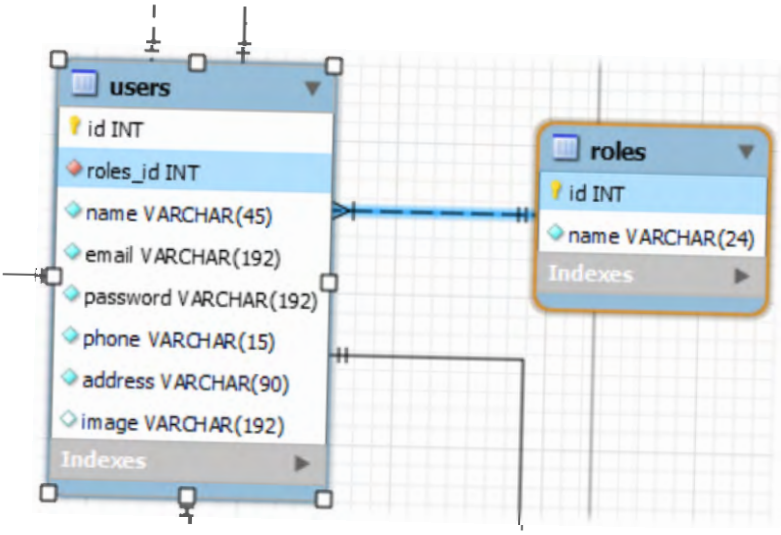

media_types
id INT
name VARCHAR(45)
Indexes
message_media
id INT
url VARCHAR(192)
media_type_id INT
client_name VARCHAR(192)
Indexes
messages
id INT
message_media_id INT
user_id INT
room_id INT
content TEXT(512)
answer_for_id INT
Indexes
message_unread_user
user_id INT
room_id INT
messages_id INT
Indexes
message_subject
id INT
subject_id INT
message_id INT
title VARCHAR(192)
Indexes
rooms
id INT
course_id INT
subject_id INT
room_type_id INT
name VARCHAR(100)
image VARCHAR(192)
Indexes
room_types
id INT
type VARCHAR(12)
Indexes
subjects
id INT
grade_id INT
name VARCHAR(90)
description TINYTEXT
Indexes
course_professor_subject
professor_id INT
course_id INT
subject_id INT
id INT
Indexes
users
id INT
role_id INT
name VARCHAR(45)
email VARCHAR(192)
password VARCHAR(192)
phone VARCHAR(15)
address VARCHAR(90)
image VARCHAR(192)
Indexes
roles
id INT
name VARCHAR(24)
Indexes
grades
id INT
name VARCHAR(90)
descripcion TINYTEXT
Indexes
room_user
user_id INT
room_id INT
isAdmin BIT(1)
isCreator BIT(1)
Indexes
courses
id INT
name VARCHAR(20)
start_date DATE
end_date DATE
Indexes
enrollments
id INT
student_id INT
grade_id INT
course_id INT
Indexes
enrollment_subject
subject_id INT
enrollment_id INT
Indexes

Tabla users

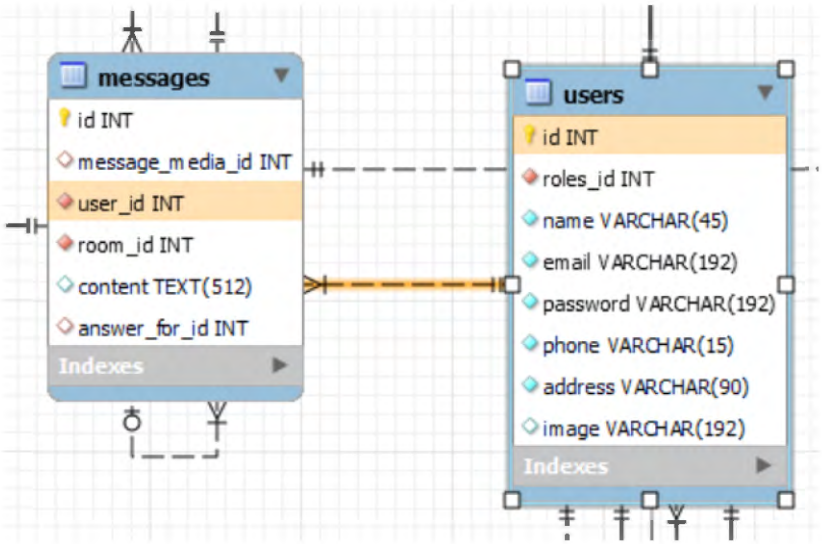
Empezaremos por la tabla de usuarios, ya que todo girará en torno a ella. En esta tabla pueden observarse campos de información del usuario como el nombre, el email, la contraseña, el número de teléfono, la dirección o la imagen de perfil; y además, hay otro campo llamado roles que indicará el rol que tiene el usuario en la aplicación. Por supuesto, también cuenta con un campo id que será único y generado automáticamente.

Para definir el rol que tiene un usuario es necesario construir una tabla que recoja todos los posibles roles y relacionarla con la tabla de usuarios. La relación se establece de tal forma que un usuario puede tener un único rol pero un rol puede pertenecer a muchos usuarios, es decir, existirán varios usuarios diferentes que posean un mismo rol. Los roles pueden ser de alumno, de profesor y de administrador; y cada uno tendrá privilegios diferentes.



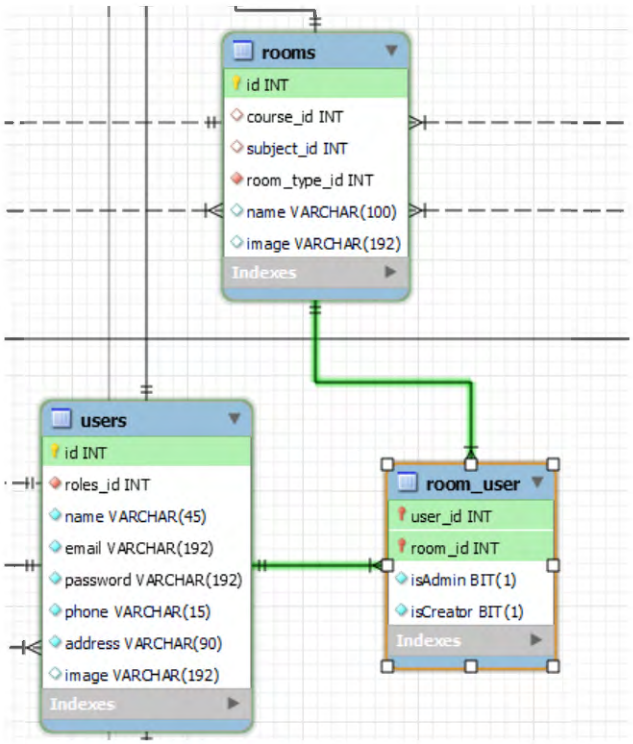
Relación de las tablas users y roles

La tabla de usuarios se relaciona con la de mensajes de forma totalmente opuesta a la de roles, es decir, un mensaje solamente podrá ser enviado por un usuario pero un usuario podrá enviar varios mensajes. De esta forma user_id pasará a ser una clave foránea en la tabla messages.



Relación users mensajes

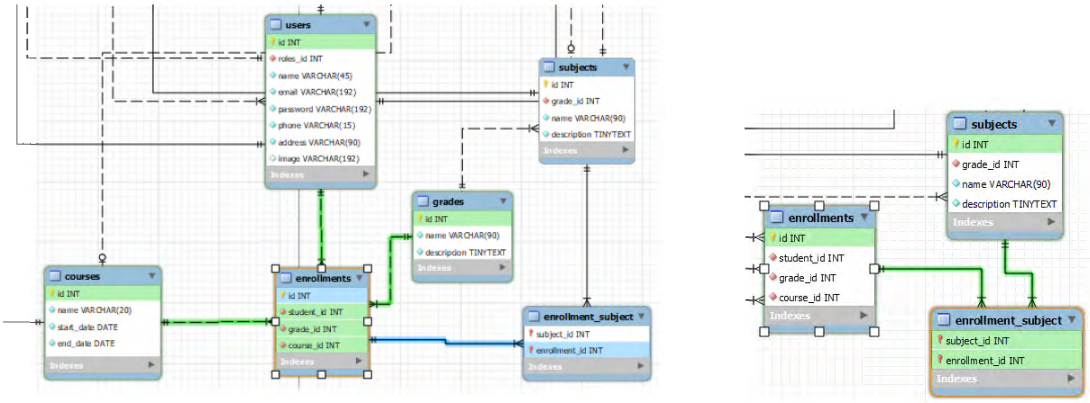
Otra relación muy relevante de la tabla usuarios es la que se establece con la tabla de salas de chat. Cada usuario puede pertenecer a varias salas de chats, y a su vez una misma sala de chats puede pertenecer a varios usuarios. Es por eso que se ha de establecer una relación de muchos a muchos entre ambas tablas. Para llevar a cabo esta relación es necesaria una tabla pivote, como se muestra en la imagen inferior:



Relación users rooms

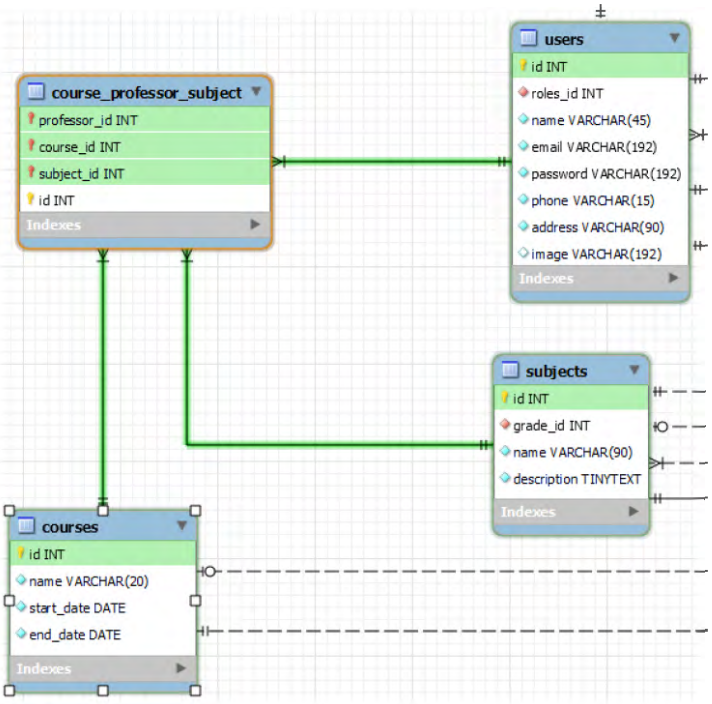
A su vez, la tabla usuarios debe relacionarse con las asignaturas en las que cada uno de los usuarios está presente. Esta relación dependerá de si el usuario es profesor o alumno.

En el caso de ser alumno se relacionará a través de la tabla llamada enrollments, que representa las asignaturas de las que un alumno se ha matriculado. En esta tabla se guarda el curso de las asignaturas, el grado al que pertenecen y el alumno en cuestión. Además, consta de una clave primaria que será necesaria para crear una relación con la tabla de asignaturas. Se hace de esta manera porque una matrícula puede contener asignaturas de diferentes cursos. También cabe la posibilidad de que una asignatura tenga el mismo nombre en más de un grado.



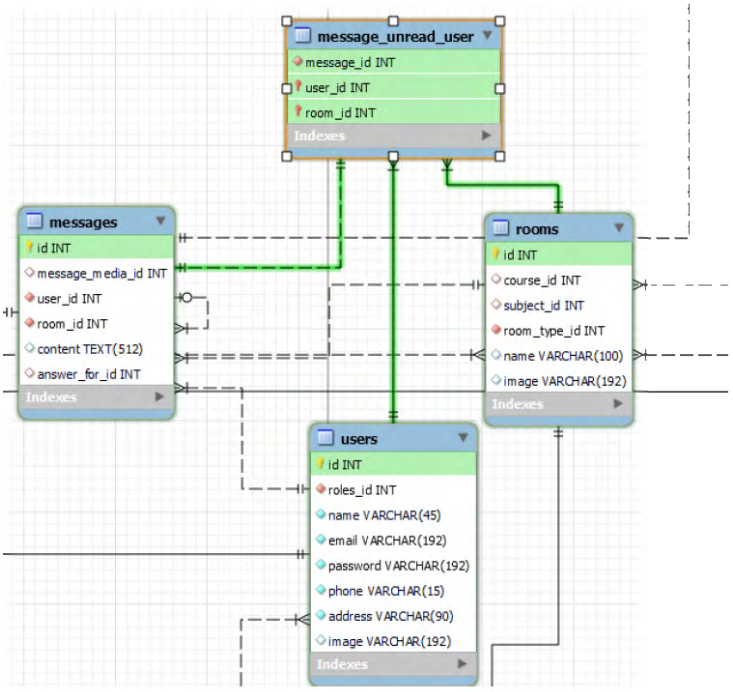
Relación de usuarios y asignaturas

En el caso contrario, cuando el usuario es un profesor, la tabla usuarios se relacionaría con la tabla asignaturas a través de una tabla intermedia llamada course_profesor_subject, que simplemente se encarga de unir a los usuarios con las asignaturas y con los diferentes cursos a los que puede pertenecer la asignatura.



Relación profesor asignaturas

Como último detalle de la tabla de usuarios, queremos saber qué mensajes aún no ha leído el usuario en cada una de sus salas de chats. Por lo tanto, será necesario relacionar los mensajes, usuarios y salas de chats a través de una tabla intermedia, la cual guardará que usuario no ha leído todavía un mensaje en una sala.



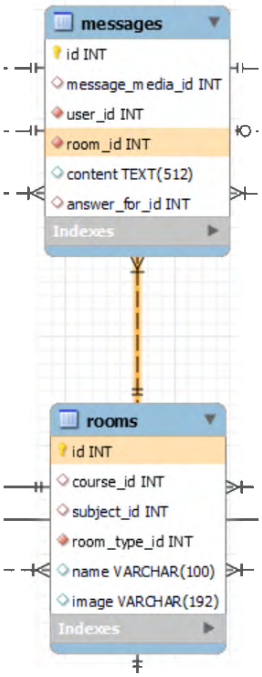
Base de datos mensajes no leídos

Tabla rooms

Otra de las tablas que tiene relaciones destacables con otras tablas es la que contiene la información de las salas de chat, conocida como rooms. Contiene un campo para guardar el nombre de la sala y otro para guardar la imagen, además de una clave primaria y tres claves foráneas.

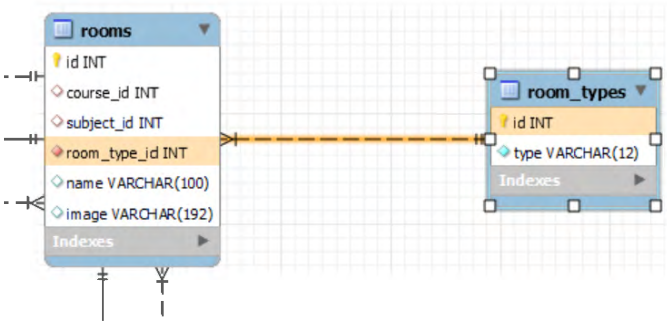
Una de sus relaciones ya se explicó anteriormente y es la que la conecta con la tabla users a través de una tabla pivote.

También debe conectarse con la tabla de mensajes a través de una relación de una a muchas, de la siguiente manera:



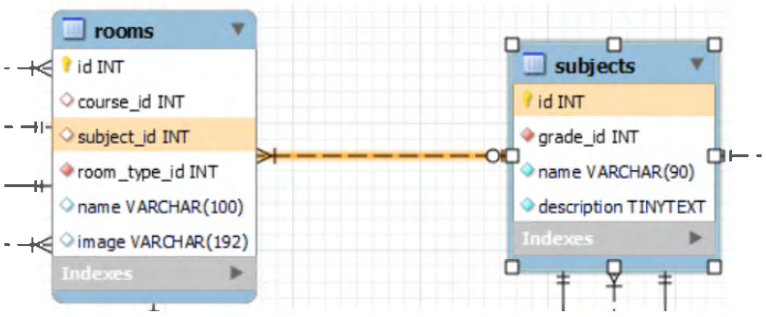
Relación mensajes-rooms

Una de las mencionadas claves foráneas es la conocida como room_type, y se conecta a otra tabla para saber si la sala de chat es de asignatura, grupal o privada.

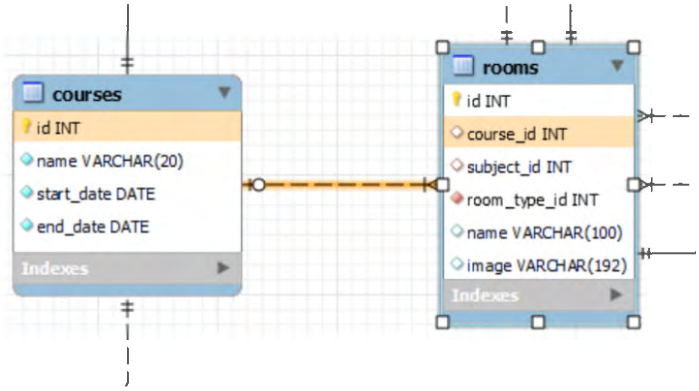


Relación tipos de sala

Las otras dos claves foráneas son las que relacionan a esta tabla con el curso y la asignatura a la que pertenece cada sala de chat. Estas relaciones son iguales que la de tipo de sala salvo por el detalle de que estos campos pueden ser nulos, ya que los grupos que no pertenezcan a ninguna asignatura no contendrán información en estos apartados.



Relación rooms subjects



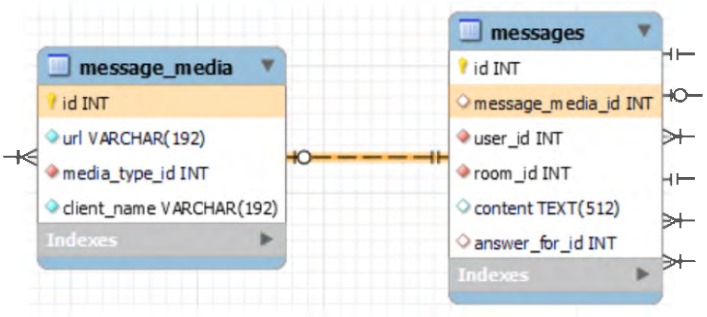
Relación curso room

Tabla messages

La última tabla de la que explicaremos sus relaciones será la tabla de mensajes. Esta tabla contiene un campo para la información del mensaje, además de una clave primaria y cuatro claves foráneas.

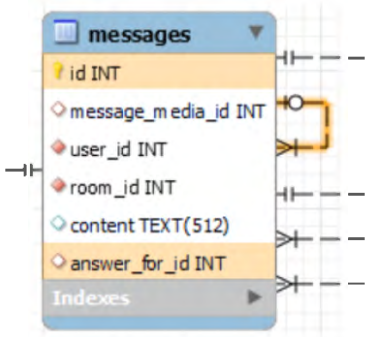
Dos de estas claves foráneas son las que relacionan a los mensajes con los usuarios y con las salas de chats, y ya han sido explicadas anteriormente.

Otra de las claves foráneas es la que indica si el mensaje es una imagen y establece la relación de la siguiente manera:



Relación tipos mensajes

La última de las claves foráneas es una relación de la tabla consigo misma, y define si el mensaje es una respuesta a otro mensaje de esa misma tabla. Obviamente puede ser nulo debido a que un mensaje no tiene por qué ser respuesta a otro.



Relación mensaje respuesta

5.1.2. ESTRUCTURA DEL PROYECTO LARAVEL

Una vez explicado cómo está diseñada la base de datos, podemos pasar a explicar cómo se ha creado y estructurado el proyecto Laravel.

A la hora de desarrollar un proyecto en Laravel es bastante útil conocer bien en qué consisten los archivos y carpetas que el propio Laravel nos facilita al crear un proyecto vacío.

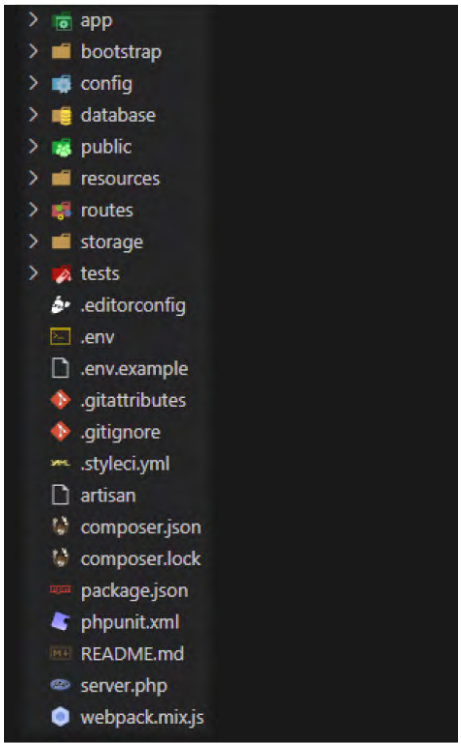
Para crear un proyecto Laravel hay que tener instalado tanto PHP como composer. PHP es un lenguaje de programación del que ya se ha hablado en anteriores puntos, y composer no es más que el gestor de dependencias de PHP. Una vez se haya comprobado que ambos elementos están correctamente instalados pasamos a crear el proyecto con un simple comando:

$composer create-project laravel/laravel mi-proyecto-laravel

Y para poner en marcha el proyecto y comprobar que se ha creado correctamente sólo es necesario otro sencillo comando:

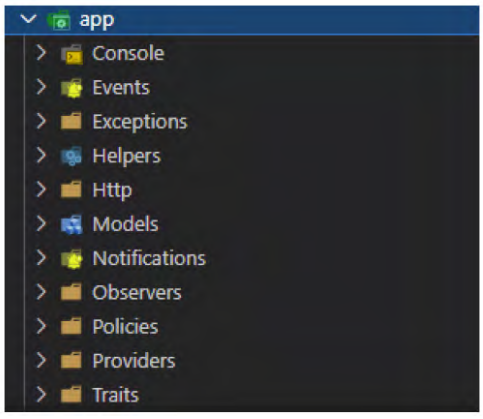
$php artisan serve

Ahora que ya tenemos la certeza de que el proyecto se ha creado correctamente podemos ver la siguiente estructura de carpetas y archivos: [40]



Estructura del proyecto Laravel

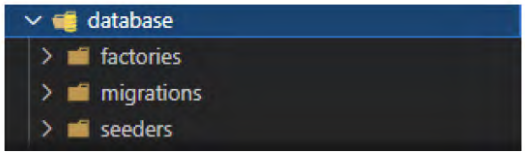
Uno de los directorios más importantes es el llamado app. Será el encargado de recoger todo el código que genere el desarrollador. Contiene archivos y también directorios muy importantes para un proyecto, como pueden ser controladores, eventos o modelos. Para esta carpeta se ha decidido utilizar una organización como esta:



Contenido de app

Ya que esta carpeta es la que modifica el desarrollador será explicada en profundidad en el siguiente apartado, especialmente la subcarpeta denominada Http, por el gran valor de su contenido.

Otra de las carpetas que incluye Laravel es la de database, donde se ubican los archivos que gestionan la base de datos. Su organización queda configurada así:



Contenido de database

Por su parte, el directorio config recoge la configuración del proyecto, que no es más que una serie de archivos llenos de matrices clave-valor y escritos en PHP . Hay dos archivos que destacan por su gran utilidad dentro de esta carpeta. Uno es el archivo app.php y permite escoger el lenguaje del proyecto, zona horaria, los providers o los aliases de las clases. Otro es el llamado database.php y es el que permite elegir el motor de la base de datos que se utilizará el proyecto.

```
'App' => Illuminate\Support\Facades\App::class,
'Arr' => Illuminate\Support\Arr::class,
'Artisan' => Illuminate\Support\Facades\Artisan::class,
'Auth' => Illuminate\Support\Facades\Auth::class,
'Blade' => Illuminate\Support\Facades\Blade::class,
'Broadcast' => Illuminate\Support\Facades\Broadcast::class,
'Bus' => Illuminate\Support\Facades\Bus::class,
'Cache' => Illuminate\Support\Facades\Cache::class,
'Config' => Illuminate\Support\Facades\Config::class,
'Cookie' => Illuminate\Support\Facades\Cookie::class,
'Crypt' => Illuminate\Support\Facades\Crypt::class,
'Date' => Illuminate\Support\Facades\Date::class,
'DB' => Illuminate\Support\Facades\DB::class,
'Eloquent' => Illuminate\Database\Eloquent\Model::class,
'Event' => Illuminate\Support\Facades\Event::class,
'File' => Illuminate\Support\Facades\File::class,
'Gate' => Illuminate\Support\Facades\Gate::class,
'Hash' => Illuminate\Support\Facades\Hash::class,
'Http' => Illuminate\Support\Facades\Http::class,
'Js' => Illuminate\Support\Js::class,
'Lang' => Illuminate\Support\Facades\Lang::class,
'Log' => Illuminate\Support\Facades\Log::class,
'Mail' => Illuminate\Support\Facades\Mail::class,
'Notification' => Illuminate\Support\Facades\Notification::class,
'Password' => Illuminate\Support\Facades\Password::class,
'Queue' => Illuminate\Support\Facades\Queue::class,
'RateLimiter' => Illuminate\Support\Facades\RateLimiter::class,
'Redirect' => Illuminate\Support\Facades\Redirect::class,
// 'Redis' => Illuminate\Support\Facades\Redis::class,
'Request' => Illuminate\Support\Facades\Request::class,
'Response' => Illuminate\Support\Facades\Response::class,
'Route' => Illuminate\Support\Facades\Route::class,
'Schema' => Illuminate\Support\Facades\Schema::class,
'Session' => Illuminate\Support\Facades\Session::class,
'Storage' => Illuminate\Support\Facades\Storage::class,
'Str' => Illuminate\Support\Str::class,
'URL' => Illuminate\Support\Facades\URL::class,
'Validator' => Illuminate\Support\Facades\Validator::class,
'View' => Illuminate\Support\Facades\View::class,
```

Aliases de las clases en el archivo app.php

Tenemos también el directorio routes, que es el que contiene las rutas a las que después se le realizarán las peticiones desde el Frontend.

El último directorio que destacaremos es el llamado storage que es donde Laravel graba la información. Dentro de este directorio es importante también la carpeta caché.

Entre los archivos que se crean al generar un proyecto en Laravel es muy relevante el llamado .env porque será el encargado de guardar la información de la base de datos.

5.1.3. ELEMENTOS DEL PROYECTO

Una vez conocida la estructura que se ha utilizado en el proyecto podemos pasar a explicar cómo se han desarrollado los elementos más importantes del proyecto. Bastará con detallar un ejemplo de cada uno de los elementos, ya que se desarrollan de la misma manera y sería muy repetitivo explicar todos los archivos de un mismo tipo

Base de datos

Como ya tenemos diseñada la base de datos, únicamente hay que implementarla en Laravel siguiendo dicho diseño. Para la creación de las tablas seguiremos los pasos que se describen a continuación. [41]

En primer lugar, hay que realizar las pertinentes configuraciones de la base datos. Para ello hay que editar los datos del archivo .env del proyecto Laravel.

```
APP_NAME=myclass
APP_ENV=local
APP_KEY=base64:Olldvl7GqPxZ1H9XKpfIKX9Ga7Q4oX8PUjg9hS/k2jI=
APP_DEBUG=true
APP_URL=http://15.237.84.107/

LOG_CHANNEL=stack
LOG_DEPRECATIONS_CHANNEL=null
LOG_LEVEL=debug

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=db_myclass
DB_USERNAME=tfgmyclass
DB_PASSWORD=
```

Archivo .env

Entre los datos editados destacan la dirección de la base de datos y el nombre de ésta.

Una vez establecidos estos parámetros podemos pasar a realizar migraciones mediante la interfaz de Laravel conocida como Artisan.

Dado que este proceso se realiza de la misma forma para todas las tablas de la base de datos, es suficiente con explicar un único caso, que será la tabla de usuarios (users).

Para crear la tabla hay que hacerlo mediante la ejecución del siguiente comando:

$php artisan make:migration create_users_table

Una vez hecho esto, podemos observar que se ha creado un archivo en la carpeta migrations que se encuentra dentro de la carpeta database de Laravel. Será en este archivo en el que podremos introducir los pertinentes campos de nuestra tabla, siempre siguiendo el diseño establecido con anterioridad.

```php
public function up()
{
    Schema::create('users', function (Blueprint $table) {
        $table->id();
        $table->foreignId('role_id')->constrained()->onUpdate('cascade')->onDelete('cascade');
        $table->string('name',45);
        $table->string('email')->unique();
        $table->string('password');
        $table->string('phone',15);
        $table->string('address',90);
        $table->string('image')->nullable();
        $table->datetime('last_login')->nullable();
        // $table->rememberToken();
        $table->timestamps();
    });
}
```

Campos de la tabla users

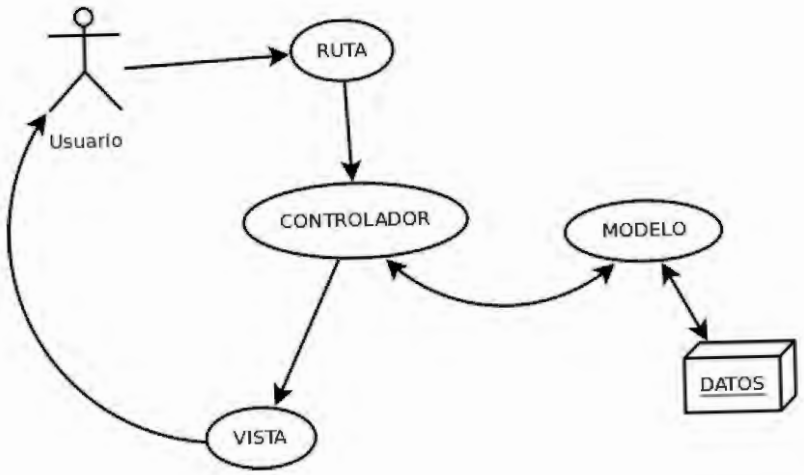
En este código destacamos la presencia de la clase Blueprint, la cual nos permite manipular las consultas como objetos. Entre los campos creados destacamos el campo role_id, que será una clave foránea y así se establecerá una relación con la tabla de roles. También se le pueden asignar diferentes características a los campos como el número de caracteres que acepta, si pueden ser nulos o si son elementos únicos como el email.

De esta manera vamos construyendo todas las tablas de nuestro diseño y una vez completados todos los valores necesarios podemos pasar a ejecutar las migraciones de forma que queden definidas las tablas y su estructura en la base de datos. Esto se puede hacer automáticamente con el siguiente comando:

$php artisan migrate

Controladores y rutas

Los controladores y las rutas en Laravel son componentes muy relacionados entre sí, ya que las rutas serán las encargadas de llamar a los controladores, y de esta forma se accederá a la ya conocida estructura modelo-vista-controlador.



Rutas en MVC

Los controladores serán los encargados de definir el comportamiento de la aplicación, permitiendo agrupar la lógica de peticiones HTTP y ejerciendo de intermediario entre las vistas y los modelos. [42]

En este caso nos centraremos en la creación del controlador asociado a las acciones del usuario, llamado UserController en nuestro proyecto. Para crear un controlador simplemente hay que ejecutar el siguiente comando de artisan en la consola: [43]

$php artisan make:controller UserController

Tras ejecutarlo observaremos que se ha creado un nuevo archivo en la carpeta Http, y éste será el archivo en el que definiremos las funciones que tiene que realizar nuestro controlador.

La primera función que destacaremos será la función index, que se encargará de mostrar una lista de información de los usuarios de la aplicación. El procedimiento de esta función es sencillo, únicamente comprobará si el usuario tiene autorización y en caso de tenerla le devolverá información sobre los usuarios tan relevante como su nombre, email o rol.

```php
public function index(Request $request)
{


    if (Auth::User()->cannot('viewAny', User::class) ) {
        return $this->handleError('No tienes autorización para realizar esta acción.', 403);
    }

    $name = $request->input('name');
    $email = $request->input('email');
    $role = $request->input('role');
    $nopaginate = $request->input('nopaginate');


    $query = User::where( function($query) use($name){
        $query->where('name', 'like', '%'.$name.'%');
    });

    $query = $query->where( function($query) use($email){
        $query->where('email', 'like', '%'.$email.'%');
    });


    $query = $query->whereHas('role', function($query) use($role){
        $query->where('name', 'like', '%'.$role.'%');
    });

    if( $nopaginate === 'true' ){
        return $this->handleResponse('Lista de asignaturas mostrada con exito', UserResource::Collection( $query->get() ) );
    }

    $user_collection = $query->paginate(25)->appends( $request->all() );
    return $this->handleResponse('Lista de usuarios mostrada con exito.' , new UserCollection($user_collection) );
}
```

Index en UserController

También debemos poder mostrar la información de un único usuario de la aplicación, filtrando también por los permisos que tenga el usuario en cuestión. Y es en ese aspecto donde entra la función show.

```php
public function show(User $user)
{
    if (Auth::User()->cannot('view', $user) ) {
        return $this->handleError('No tienes autorización para realizar esta acción.', 403);
    }

    return $this->handleResponse('Usuario mostrado con exito.', new UserResource($user) );
}
```

Show en UserController

La siguiente función es la denominada store, que será la responsable de almacenar los nuevos usuarios creados. Para ello debe comprobar si se dispone de la pertinente autorización y a continuación realizar una validación de los datos recibidos, en el caso de ser válidos se creará el usuario.

```php
public function store(Request $request)
{
    if (Auth::User()->cannot('create', User::class) ) {
        return $this->handleError('No tienes autorización para realizar esta acción.', 403);
    }

    $validated = $request->validate([
        'email' => 'bail|required|email|unique:users,email',
        'name' => 'bail|required|min:3|max:45|regex:/^[a-zA-ZñÑáéíóúÁÉÍÓÚ\s]+$/',
        'address' => 'bail|required|min:3|max:90',
        'phone' => 'bail|required|min:9|max:15',
        'role_id' => 'bail|required|in:1,2,3',
    ]);

    $PasswordWihoutHash = Str::random(8);
    $validated['password'] = \Hash::make($PasswordWihoutHash);

    $user = User::create( $validated );

    $user->notify( new WelcomeNotification($user,$PasswordWihoutHash) );

    return $this->handleResponse('Usuario creado con exito.', 201);
}
```

Store en UserController

De la misma manera que podemos almacenar nuevos recursos debemos poder actualizar los existentes, y para ello implementamos la función update.

```php
public function update(Request $request, User $user)
{
    if (Auth::User()->cannot('update', $user ) ) {
        return $this->handleError('No tienes autorización para realizar esta acción.', 403);
    }

    $validated = $request->validate([
        'email' => 'bail|email|unique:users,email,' . $user->id,
        'name' => 'bail|min:3|max:45|regex:/^([0-9a-zA-ZñÑáéíóúÁÉÍÓÚ_-])+((\s*)+([0-9a-zA-ZñÑáéíóúÁÉÍÓÚ_-]*)*)+$/',
        'password' => ['bail', 'confirmed','max:16', PasswordRules::min(8)->letters()->numbers()],
        'address' => 'bail|min:3|max:90',
        'phone' => 'bail|min:9|max:15',
        'image' => 'bail|image|max:1000',
        'role_id' => 'bail|in:1,2,3',
    ]);

    if( $request->has('name') && strtolower($request->name) !== strtolower($user->name) && Auth::user()->role->id === 3 ){
        $user->name = $validated['name'];
    }

    if( $request->has('email') && strtolower($request->email) !== strtolower($user->email) ){
        $user->email = $validated['email'];
    }

    if( $request->has('password') ){
        $user->password = Hash::make( $validated['password'] );
        $user->notify(new PasswordChangedNotification($user) );
    }

    if( $request->has('address') && strtolower($request->address) !== strtolower($user->address) ){
        $user->address = $validated['address'];
    }

    if( $request->has('phone') && $request->phone !== $user->phone ){
        $user->phone = $validated['phone'];
    }

    if( $request->hasFile('image') ){
        if($user->image !== null){
            Storage::disk('avatars')->delete($user->image);
        }
        $user->image = Storage::disk('avatars')->put('', $validated['image']);
        $path = Storage::disk('avatars')->getAdapter()->getPathPrefix();
        $image_src = $path.$user->image;
        $this->ImageThumbnail( $image_src, $image_src, 160 );
    }

    if( $request->has('role_id') && $request->role_id !== $user->role_id && Auth::user()->role->id === 3 ){
        $user->role_id = $validated['role_id'];
    }


    if( !$user->isDirty() ){
        return $this->handleError("Debe modificar uno de los campos.", 422);
    }

    $user->save();

    return $this->handleResponse('Usuario actualizado con exito',new UserResource($user), 200);
}
```

Update en UserController

Asimismo, debe ser posible eliminar recursos. Para llevar a cabo este proceso implementamos la función destroy, que comprobará si el usuario en cuestión tiene los permisos necesarios y de ser así procederá al borrado de los datos.

```php
public function destroy(User $user)
{
    if (Auth::User()->cannot('delete', $user ) ) {
        return $this->handleError('No tienes autorización para realizar esta acción.', 403);
    }

    if($user->image !== null){
        Storage::disk('avatars')->delete($user->image);
    }

    if(!$user->delete()){
        $this->handleError("Usuario eliminado con exito.", 422);
    }

    return $this->handleResponse("Usuario eliminado con exito.");
}
```

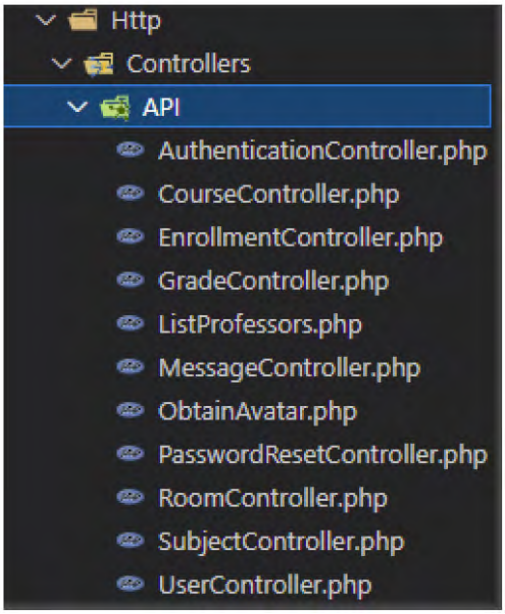
Destroy en UserController

Por último, queremos que sea posible obtener las asignaturas de un usuario. De este modo creamos la siguiente función, que distinguirá si el usuario es alumno o profesor y filtrará las asignaturas por curso, se llamará getUserSubjects.

```php
public function getUserSubjects( Request $request ){
    $validated = $request->validate([
        'course_id' => 'bail|numeric|exists:courses,id'
    ]);

    if( $request->has('course_id') ){
        $course = Course::findOrFail( $request->course_id );
    }else{
        $course = Course::latest('id')->first();
    }

    $subjects = [];

    if( Auth::user()->role->id === 1 ){
        $subjects = Auth::user()->enrollments()->whereRelation('course', 'course_id' , $course->id )->first()->subjects;
    }

    if( Auth::user()->role->id === 2 ){
        $subjects = Auth::user()->subjects()->where('course_id', $course->id)->get()->makeHidden(['pivot', 'created_at', 'updated_at']);
    }

    return $this->handleResponse( "Lista de asignaturas enviada con éxito.", $subjects , 200 );
}
```

Index en UserController

Siguiendo esta misma lógica crearemos los controladores restantes, obteniendo así la siguiente serie de archivos en la carpeta de controladores, donde cada uno contendrá sus respectivas funciones:



Lista de controladores

Por la otra parte, las rutas en Laravel son el componente más importante y poderoso del Framework, ya que es el componente responsable del manejo del flujo de peticiones HTTP, desde el cliente y hacia la URL concreta. Por supuesto, las peticiones pueden ser de los tipos GET, POST, PUT o DELETE. [44]

Las rutas de un proyecto Laravel pueden ser de tipo web o de tipo API, pero dado que este proyecto consiste en un API a la que un Frontend le realizará peticiones, únicamente construiremos rutas de tipo API.

Para la creación de una ruta únicamente se ha de definir el método de la ruta (GET, POST, PUT, DELETE) y pasarle dos parámetros, el primero será el nombre de la ruta y el segundo será el controlador al que llamará.

En nuestra aplicación tendremos una ruta sin protección que servirá de autenticación y una serie de rutas protegidas para las que habrá que haber iniciado sesión previamente para poder acceder a ellas. Las rutas protegidas utilizadas son las siguientes:

```php
Route::middleware(['auth:sanctum'])->group(function () {

    /* Rutas de sesión */
    Route::get('/user', [AuthenticationController::class, 'user']);
    Route::post('/signout', [AuthenticationController::class, 'signout']);

    /* Rutas para acceso a archivos privados de avatares */
    Route::get('users/get_avatar/{image}', ObtainAvatar::class);
    /* Ruta que lista los profesores de la aplicacion */
    Route::get('professors', ListProfessors::class);

    /* Rutas recurso usuario */
    Route::get('users/subjects', [UserController::class, 'getUserSubjects' ]);
    Route::apiResource('users', UserController::class);
    /* Rutas recurso grados */
    Route::apiResource('grades', GradeController::class);
    /* Rutas recurso asignaturas */
    Route::apiResource('subjects', SubjectController::class);
    Route::get('subjects/{subject}/users', [SubjectController::class, 'getUsersOnSubject']);
    /* Rutas cursos */
    Route::apiResource( 'courses', CourseController::class)->except(['update']);
    /* Rutas enrollments */
    Route::apiResource( 'enrollments', EnrollmentController::class );
    /* Rutas salas de chat */
    Route::apiResource( 'rooms', RoomController::class );
    Route::post('rooms/{room}/add', [RoomController::class, 'addPartipant']);
    Route::post('rooms/{room}/modify', [RoomController::class, 'togglePermision']);
    Route::post('rooms/{room}/remove', [RoomController::class, 'removeParticipant']);
    Route::get('rooms/{room}/exit', [RoomController::class, 'exitRoom']);
    Route::get('rooms/{room}/avatar', [RoomController::class, 'getRoomAvatar']);
    Route::get('rooms/{room}/read', [RoomController::class, 'markAsReadRoom']);
    /* Rutas salas de chat */
    Route::apiResource( 'messages', MessageController::class )->except(['update']);
    Route::get('messages/{message}/download', [MessageController::class, 'getMessageFile']);
    Route::get('messages/{message}/thumbnail', [MessageController::class, 'getMessageFileThumbnail']);
});
```

Rutas protegidas

Entre ellas podemos observar algunas para obtener las salas de chat o los mensajes de una sala, otras para obtener información sobre el usuario o incluso otra para cerrar la sesión.

Eventos

Los eventos en Laravel son los encargados de proporcionar observadores para determinadas funcionalidades, o dicho de otra manera, un evento es una función que se ejecuta siempre que sucede una acción determinada en la aplicación. [45]

En la presente aplicación los eventos se utilizarán de diversas formas, pero la más importante de todas es el envío de mensajes en una sala de chat, permitiendo la comunicación en tiempo real de los diferentes interlocutores.

Con el fin de facilitar esta comunicación en tiempo real en la sala de chat se debe utilizar una conexión Websockets.

Para realizar esta conexión, como primer paso, se debe instalar la librería Pusher mediante los siguientes comandos:

$composer require beyondcode/laravel-websockets

$composer require pusher/pusher-php-server

Una vez instalada correctamente, se debe configurar la librería en el archivo .env estableciendo las siguientes variables de entorno:

```
PUSHER_APP_ID=your-pusher-app-id
PUSHER_APP_KEY=your-pusher-key
PUSHER_APP_SECRET=your-pusher-secret
PUSHER_APP_CLUSTER=mt1
```

Archivo .env librería pusher

Una vez hecho esto, podemos pasar a instalar y configurar Laravel Echo, que recibirá los eventos de transmisión en el lado del cliente. Sin embargo, la parte del cliente se explicará en la implementación del Frontend, ya que es ahí donde se tratará dicha parte en su totalidad.

Siguiendo con la correspondiente parte del lado del servidor, pasaremos a crear el canal de difusión por el que se enviarán los mensajes a los usuarios que estén en escucha.

```
/* Canal para nuevos mensajes del chat */
Broadcast::channel('chat.{room}', function ($user, Room $room) {
    return $room->users()->where('user_id', $user->id)->exists();
});
```

Canal difusión mensajes

Por último solo queda crear el evento que se encargará de enviar los mensajes por el canal de difusión. Para ello primero creamos el evento con el siguiente comando de artisan:

$php artisan make:event ChatMessageEvent

Y una vez creado lo configuramos de la siguiente manera:

```php
public function __construct( Message $message )
{
    $this->message = $message;
}

public function broadcastOn()
{
    return new PrivateChannel('chat.'.$this->message->room_id);
}

public function broadcastWith()
{
    return (new MessageResource($this->message))->resolve();
}
```

Evento de mensaje

Donde únicamente hay que crear un constructor para instanciar el evento y añadir las funciones de difusión, especificando en este caso que el canal por el que se difundirá el evento es privado, ya que solamente nos interesa que el mensaje se envíe a los usuarios de la sala de chat en cuestión.

Puertas y Políticas

El marco de Laravel implementa la autorización en forma de puertas y políticas. Laravel nos provee de un sistema de autorización que funciona junto con el sistema de autenticación para identificar la sesión de usuario legítima. [46]

A continuación, crearemos una política que se encarga de autorizar al usuario a realizar ciertas acciones.

```php
public function before(User $user, $ability)
{
    if ($user->role->name === 'Administrador') {
        return true;
    }
}

public function viewAny(User $user)
{
    return true;
}

public function view(User $user, User $model)
{
    return true;
}

public function create(User $user)
{
    return false;
}

public function update(User $user, User $model)
{
    return $user->id === $model->id;
}

public function delete(User $user, User $model)
{
    return $user->id === $model->id;
}
```

Política de usuario

La política de este código tiene la posibilidad de dar autorización a diversas acciones. Algunas nos permiten acceder a un recurso de un modelo o a todos ellos a la vez; mientras que otras permiten crear, borrar o actualizar usuarios.

5.2. IMPLEMENTACIÓN DEL FRONTEND

El Frontend de este proyecto consta únicamente de una interfaz de aplicación móvil desarrollada con React Native que puede conectarse al servidor para realizar peticiones y representar la información.

5.2.1. ESTRUCTURA DEL PROYECTO REACT NATIVE

Tras la creación de un proyecto de React Native vemos un proyecto con la estructura generada por defecto, la cual sirve de base para que el desarrollador estructure el proyecto a su gusto.

Pero antes de nada vamos a ver cómo se crea el proyecto. El único requerimiento que necesitamos cumplir para generar un proyecto de React Native es tener instalados tanto Node como el editor Visual Studio Code, que incluye la posibilidad de utilizar la consola de comandos dentro del proyecto para poder escribir en ella los comandos para crear el proyecto y para ir instalando las dependencias que vayamos a utilizar a medida que sean necesarias. Una vez se tiene todo instalado sólo hay que crear la aplicación y darle nombre utilizando estos dos sencillos comandos:

$npm install -g react-native-cli

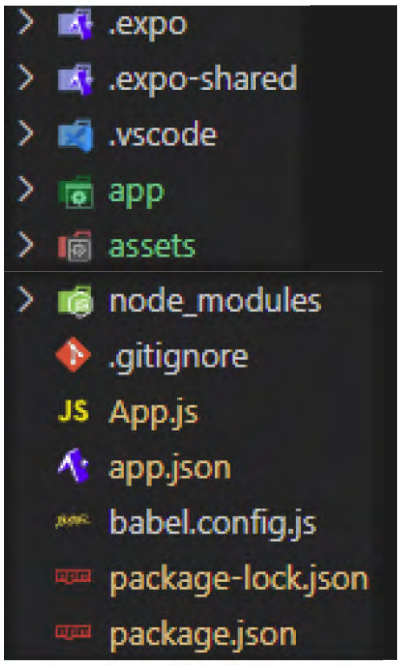
$react-native init nombreProyecto

Para comprobar que la creación del proyecto se ha producido correctamente podemos escribir el siguiente comando dentro de la ruta del proyecto:

$npm start

De este modo se crea un servidor local que se abrirá en tu navegador, contendrá un código QR que tendremos que escanear con nuestro dispositivo móvil para ejecutar allí la aplicación. Aunque es importante destacar que previamente hay que tener instalada la app de Expo Go en el dispositivo móvil.

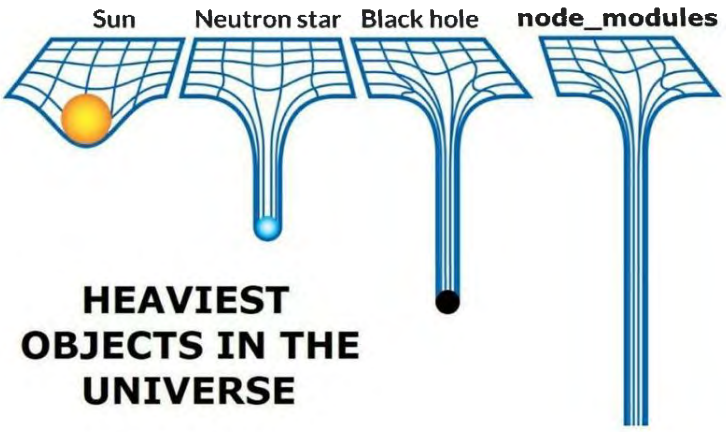
Una vez tenemos el proyecto funcionando correctamente podemos observar los archivos y carpetas que contiene por defecto el proyecto: [47]



Estructura del proyecto

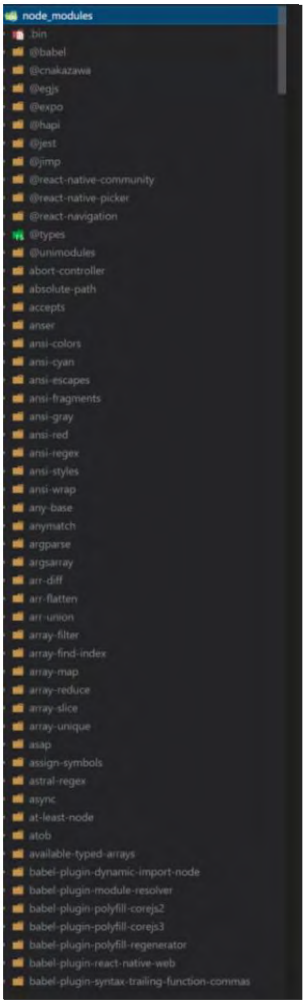
Esta imagen por sí sola no nos dice demasiado, por lo que vamos a dar un repaso por cada una de las carpetas y cada uno de los archivos que se muestras para explicar en qué consiste cada una de ellas.

Por su innegable importancia empezaremos hablando sobre la carpeta node_modules, la cual es un directorio generado con los pertinentes paquetes o dependencias instaladas mediante npm. Los paquetes que se instalarán en el futuro mediante a través de la ventana de comandos también se reflejarán en él, y de esta manera no será necesario manipular el proyecto para realizar una instalación de paquetes, razón por la cual podemos instalar las dependencias desde la línea de comandos. Dicho de otra forma, cualquier paquete instalado en nuestro proyecto se almacenará en la carpeta node_modules, y dentro de esta carpeta se guardará automáticamente en otra carpeta con el nombre del paquete en cuestión y con todos los ficheros de las dependencias respectivas. Por estos motivos, entre los desarrolladores se hace referencia a la carpeta node_modules como el martillo de Thor, la carpeta Tardis (infinita por dentro) o directamente como el objeto más pesado del universo: [48]



La carpeta node_modules como el objeto más pesado del universo

En la siguiente imagen se muestra sólo una mínima parte del contenido de node_modules para que los lectores puedan hacerse una idea de su colosal tamaño, por no mencionar que algunas de las carpetas que se muestran contienen una gran cantidad de carpetas en su interior.



Pequeña parte del contenido de la carpeta node_modules

Como último punto a destacar sobre la carpeta node_modules, debe mencionarse que jamás debe subirse a un control de versiones como git (en el caso de utilizarlo). Existen otros métodos infinitamente más eficientes para recuperar las dependencias a la hora de clonar un proyecto versionado con git.

Por otra parte, el contenido de las carpetas expo y de la carpeta assets es sencillo de explicar, las primeras contiene archivos de configuración de expo, mientras que assets contendrá las imágenes que se utilizarán en el proyecto de forma local.

Continuemos ahora con los archivos. Concretamente con el archivo App.js, siendo este el componente que será mostrado inicialmente al ejecutar aplicación. En el momento en el que se crea un proyecto únicamente cuenta con una plantilla básica para que veamos que el proyecto está corriendo de forma correcta. Pero cuando la aplicación esté finalizada será el componente sobre el que se renderizarán los componentes más importantes, como puede ser el menú de navegación o los contextos que englobarán información muy relevante de la aplicación.

Seguidamente de App.js nos encontramos con App.json que contiene información básica de la aplicación como el nombre, el icono o las plataformas que soporta; pero también algunos de los permisos que ha sido necesario editar durante el desarrollo, como el acceso a la galería de imágenes.

Contenido del archivo App.js

A continuación, vemos otro archivo de extensión .json llamado package.json, que es generado por Node al crear un proyecto y en él se guarda toda la información que se conoce sobre el proyecto. Es simplemente un fichero de texto en formato JSON con información de tipo muy variado. A través de sus campos se puede guardar y recuperar información vital sobre el proyecto. Fundamentalmente se utiliza para tener controladas las dependencias instaladas, de tal manera que las añadidas en un futuro serán añadidas aquí. Volviendo al caso de utilizar un control de versiones, cuando alguien clone el proyecto podrá instalar estas dependencias gracias a este archivo, y así no tendrá que clonar la pesada carpeta node_modules. [49]

Seguidamente se muestran algunos de los campos más habituales del archivo package.json:

Tipo	Campo	Descripción
STRING	name	Nombre del proyecto, librería o paquete. Se recomienda que coincida con el repositorio.
STRING	version	Versión del paquete. Generalmente se utiliza **semver** (lo veremos más adelante).
STRING	description	Descripción breve del paquete o proyecto.
STRING	main	**Punto de entrada** del proyecto. Suele ser `index.js` (node) o `index.html` (browser).
STRING	module	Idem al anterior, pero respecto a ES Modules en lugar de **CommonJS**.
OBJECT	scripts	Colección de scripts del proyecto (lo veremos más adelante).
ARRAY	keywords	ARRAY de STRING con palabras clave relacionadas con el proyecto. Util en búsquedas.
STRING	author	Nombre del autor del paquete o un ARRAY con `name`, `email` y/o `url`.
STRING	license	Tipo de licencia del paquete o proyecto. Por defecto, ISC W.
OBJECT	dependencies	Colección de paquetes para producción y la versión instalada.
OBJECT	devDependencies	Colección de paquetes para desarrollo y la versión instalada.
STRING	homepage	URL de la página principal del paquete.
OBJECT	repository	URL del repositorio. Se debe indicar `type` (git, svn...) y `url` (ruta).
OBJECT	bugs	Objeto con campo `url` con la URL de la página de issues del proyecto.

Campos habituales package.json

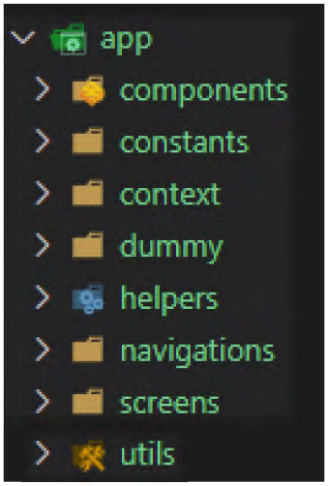
Con un nombre muy similar al anterior le sigue el archivo package-lock.json, que de la misma manera se genera automáticamente. Su principal función es la de mantener un historial de los paquetes instalados y optimizar la forma en que se generan las dependencias del proyecto y los contenidos de la carpeta node_modules. Este archivo debe conservarse bajo cualquier causa, incluso añadirlo al repositorio Git del proyecto, razón por la cual no ha de ser incluido en el fichero .gitignore. [50]

El archivo .gitignore lo hemos dejado para el final ya que para la realización de este proyecto no ha sido implementado ningún control de versiones y éste es un archivo de texto que le dice al control de versiones git qué archivos o carpetas ignorar cuando el desarrollador guarda o descarga una versión de git.

De esta forma podemos clonar un proyecto quitando archivos innecesarios y bastante pesados, como la ya famosa carpeta node_modules.

En cuanto a las carpetas todavía no ha sido explicado nada sobre la carpeta app. Esto se debe a que su contenido será la estructura de carpetas y archivos que vayamos añadiendo al proyecto. Dicho de otra manera, es la parte que el desarrollador tendrá que ajustar en función de la aplicación que vaya a desarrollar. Aunque es importante destacar que esta estructura es decisión personal de cada desarrollador. Por lo tanto, no es relevante en proyectos desarrollados por una única persona, pero sí lo es en proyectos de empresa que sean desarrollados entre varios empleados y deban trabajar todos en el mismo proyecto.

Sabiendo todo esto podemos proceder a explicar la estructura seguida para la elaboración de esta aplicación.



Contenido de la carpeta app

La navegación principal de la aplicación se basa en un sistema de tabs (pestañas). Cuenta únicamente con dos tabs, uno para el perfil y otro para los chats. Pero dentro de cada pestaña se puede cambiar de pantalla mientras sigues en la misma pestaña. El conjunto de pantallas que forman parte de la misma pestaña se conoce como stack de navegación. Una vez explicado esto es fácil comprender el contenido de la carpeta navegación, pues en ella se encuentra el sistema de navegación de tabs y los stacks de navegación de las pestañas de perfil y de chats.

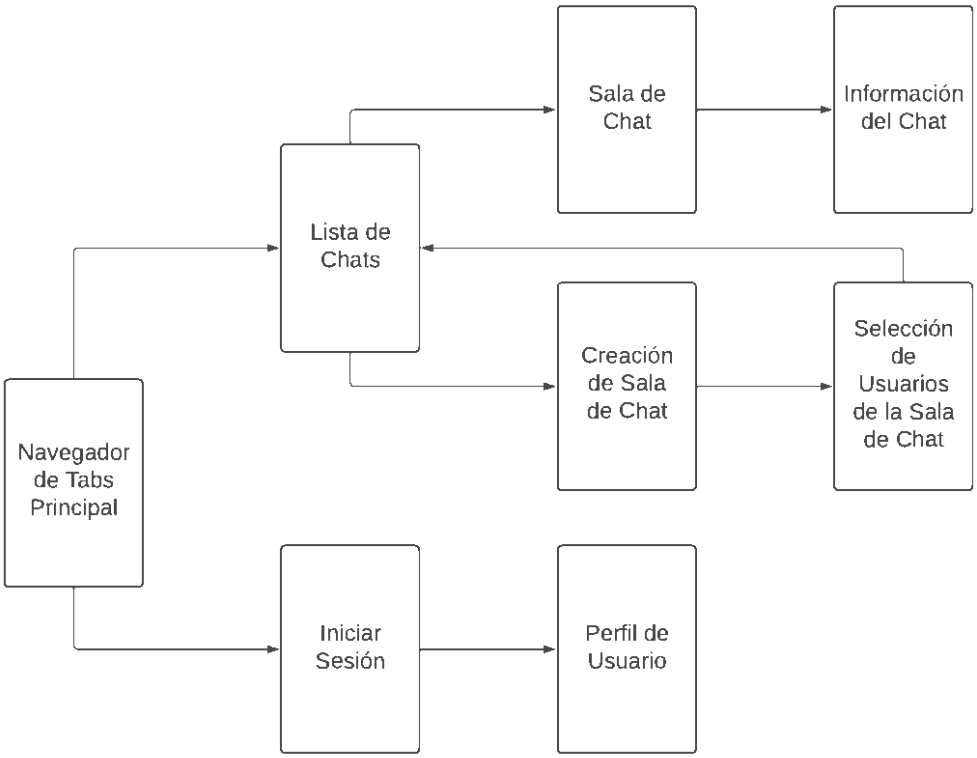


Diagrama de la navegación de la aplicación

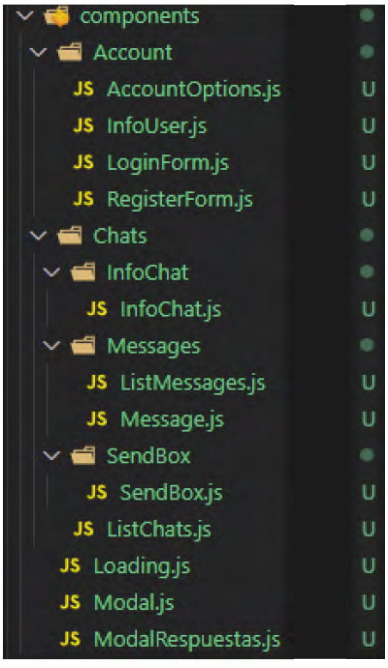
Pasemos ahora a explicar las pantallas contenidas en cada una de las pestañas, así como la relación que guardan entre todas ellas. Todas las pantallas que se mencionarán a continuación se encuentran dentro de la carpeta llamada screens.

En la de perfil encontraremos una pantalla de bienvenida que se cargue al momento de iniciar la aplicación, desde ahí podremos navegar a una segunda pantalla que podrá ser la de inicio de sesión o la de edición de perfil, en función de si hemos iniciado sesión o no. La aplicación es capaz de juzgar si se ha iniciado sesión almacenando en memoria el token que le devolverá el servidor una vez hay enviado los credenciales correctamente. Este token de seguridad tiene una duración de 2 horas, por lo que la sesión se mantendrá iniciada durante ese tiempo aunque se cierre la aplicación. Aunque si el usuario lo desea puede cerrar sesión manualmente desde la pantalla de perfil.

En la pestaña de navegación encontraremos la pantalla en la cual se renderizarán todas las salas de chat en una lista con su información correspondiente siempre y cuando el usuario haya iniciado sesión, así como un botón para crear una nueva sala de chat. En caso de no haber iniciado sesión se le hará saber al usuario y además contará con un botón de redirección a la pantalla de inicio de sesión. Pero vayamos al caso de que el usuario ha iniciado sesión, caso en el cual dispondrá de dos opciones de navegación. En primer

lugar, puede entrar a cualquiera de las salas de chats y allí ver todos los mensajes enviados y escribir nuevos si lo desea. Desde esta pantalla podrá navegar a la pantalla de información de dicha sala de chat haciendo click en la imagen de la sala de chat situada en la barra superior de la aplicación. En segundo lugar hacer click en el botón de crear nueva sala de chat y ser dirigido a una pantalla en la que elegirá el tipo de sala de chat que desea crear (chat privado, grupo o asignatura). Al seleccionar el tipo de chat a crear la aplicación nos llevará a una pantalla que permitirá elegir al usuario o usuarios con los que se quiere abrir una comunicación. Al crear la sala de chat seremos redirigidos a la pantalla en la que se muestran las salas de chat.

Una vez visto el sistema navegación entre pantallas de toda la aplicación podemos explicar cómo están constituidas las pantallas. Cada pantalla está formada por uno o varios componentes que se han creado en la carpeta llamada components que se mostraba en la imagen. Dentro de esta carpeta se han ordenado los componentes en función de la pantalla a la cual pertenecen, de esta forma los tenemos más accesibles en el caso de querer modificar algún parámetro o en el caso de haber encontrado algún error mientras se realiza una prueba de la aplicación. Además esta carpeta contiene los componentes modales, que son componentes que se superponen a la pantalla de la aplicación. Estos son por ejemplo las imágenes del chat que se muestran en un modal en tamaño completo al hacer click en la miniatura de la imagen, o también las respuestas a un mensaje que se visualizan al hacer click en el mensaje en cuestión. Para un mayor entendimiento de esta parte, los componentes serán explicados en profundidad en el siguiente subapartado.



Estructura de los componentes que forman las pantallas de la aplicación

Con todo lo explicado en los últimos párrafos ya sería posible empezar a programar la aplicación, pero habría ciertas partes del código que resultaría bastante incómodo de escribir. La función de los archivos contenidos en el resto de carpetas es, por lo tanto, facilitar la escritura del código y permitir que quede más ordenado y legible.

Empezaremos por la carpeta context, que contiene componentes que le proveen a la aplicación una manera de pasar datos a través del árbol de componentes sin tener que pasar props manualmente en cada nivel [51]. Normalmente los datos se pasan por props de componentes mayores a componentes menores, pero en el caso de tener que pasar el mismo dato por más de dos componentes es conveniente utilizar un context porque ya resulta mucho más cómodo. Esto resulta especialmente útil si queremos tener datos globales en nuestra aplicación, ya que añadiendo el context una vez ya podemos utilizarlo en cualquier componente de la aplicación.

Por otro lado, la carpeta helpers suele utilizarse para generar ciertas partes de código propio. En este caso se ha utilizado para implementar el cliente de Pusher para poder suscribirse a canales de difusión y poder recibir así la información a través de Websockets.

Continuando con el contenido de la estructura, la carpeta constants nos permitirá tener a mano valores fijos que vamos a utilizar en muchas partes de nuestra aplicación, en esta aplicación únicamente se ha utilizado para guardar la dirección base de alojamiento del servidor, de manera que si algún día se cambiase de alojamiento solamente habría que cambiar la dirección en este archivo y no en todas las partes en las que se hace un petición HTTP.

Otra carpeta interesante es la denominada utils, que suele utilizarse para guardar servicios que puedan servir de utilidad en ciertos momentos del código. Por ejemplo, a la hora de validar el contenido de un campo de texto, guardamos en dicha carpeta una función que valide si el campo introducido se ajusta a las necesidades deseadas y desde el campo de texto invocar a la función.

Por último mencionaremos la carpeta dummy, donde se guardarán los dummy-data utilizados durante el proceso de desarrollo. Son datos ficticios de información benigna que no contienen datos útiles, pero sí que sirven para reservar espacio donde los datos reales están nominalmente presentes [52]. Una vez se realizamos las pruebas y pasamos a colocar los datos reales provenientes del servidor, los dummy-data pueden eliminarse. Así mismo puede eliminarse la carpeta, pero es oportuno dejarla por si en un futuro se continúa desarrollando la aplicación y queremos realizar nuevas pruebas con nuevos dummy-data.

5.2.2 COMPONENTES DEL PROYECTO

El código implementado en esta parte es denso, e incluso en muchas partes resulta muy repetitivo. Es por eso que el código de aquellos componentes que se repitan sólo será explicado una única vez.

Para que la explicación sea más ordenada esta sección se explicará por partes, las cuales serán definidas por las pantallas a las que pertenecen los componentes y de esta forma también se comprenderá mejor el contexto que rodea a dichos componentes.

Modelo general de un componente

Cada componente que se ha creado en la aplicación sigue un patrón de creación similar.

```
import React, { useEffect, useState, useContext } from "react";

const InfoUser = (props) => {
};

const styles = StyleSheet.create({
});

export default InfoUser;
```

Patrón de creación de un componente

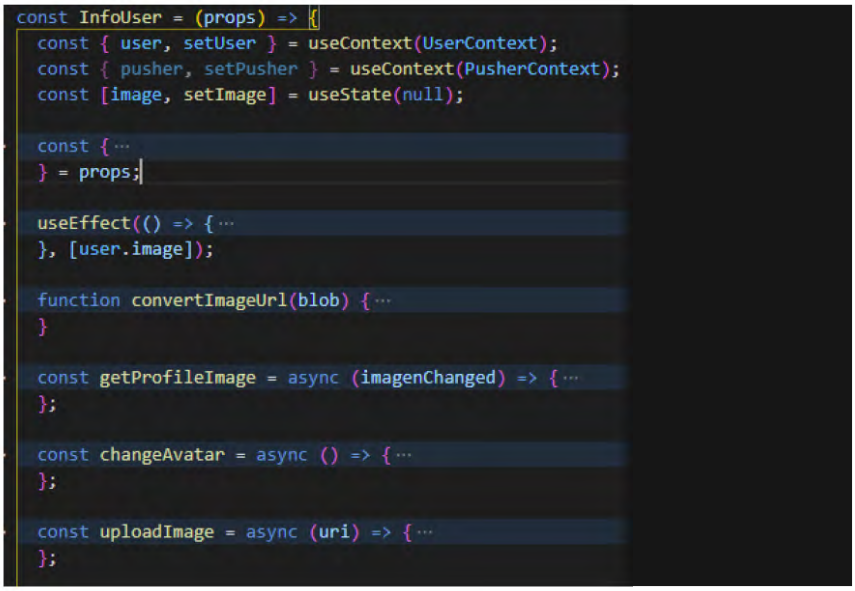
En primer lugar, se importan los componentes que serán necesarios en éste componente. Los importados pueden ser de librerías de React, de librerías externas o incluso componentes creados en el propio proyecto.

```
import React, { useEffect, useState, useContext } from "react";
import {
  StyleSheet,
  Text,
  View,
  Image
} from "react-native";
import { Avatar } from "react-native-elements";
import * as Permissions from "expo-permissions";
import * as ImagePicker from "expo-image-picker";
import BaseUrl from "../../constants/enviroments";
import UserContext from "../../context/UserContext";
import PusherContext from "../../context/PusherContext";
```

Importaciones en un componente

Seguidamente vemos la función principal del componente, que contará con la lógica del componente y devolverá el método render, que permitirá mostrar por

pantalla el componente. En esa parte se puede ver claramente que el componente creado está formado por los componentes importados anteriormente.



```
const InfoUser = (props) => {
  const { user, setUser } = useContext(UserContext);
  const { pusher, setPusher } = useContext(PusherContext);
  const [image, setImage] = useState(null);

  const {···
  } = props;

  useEffect(() => {···
  }, [user.image]);

  function convertImageUrl(blob) {···
  }

  const getProfileImage = async (imagenChanged) => {···
  };

  const changeAvatar = async () => {···
  };

  const uploadImage = async (uri) => {···
  };
```

Lógica del componente

```
return (
  <View style={styles.viewUserInfo}>
    <Avatar
      rounded
      size="large"
      showEditButton
      onEditPress={changeAvatar}
      containerStyle={styles.userInfoAvatar}
      source={
        data.image
          ? {
              uri: image,
            }
          : require("../../../assets/img/avatar-default.jpg")
      }
    />
    <View>
      <Text style={styles.displayName}>
        {!data.name ? "Anónimo" : data.name}
      </Text>
      <Text>{data.email}</Text>
    </View>
  </View>
);
};
```

Método render del componente

A continuación, podemos ver la creación de los estilos utilizados en los componentes.

```
const styles = StyleSheet.create({
  viewUserInfo: {
    alignItems: "center",
    justifyContent: "center",
    flexDirection: "row",
    backgroundColor: "#f2f2f2",
    paddingBottom: 30,
    paddingTop: 30,
  },
  userInfoAvatar: {
    marginRight: 20,
  },
  displayName: {
    fontWeight: "bold",
    paddingBottom: 5,
  },
});
```

Creación de los estilos

Por último, únicamente hay que exportar el componente para que pueda ser utilizado en otros componentes de la aplicación.

Componente principal de la aplicación

El archivo App.js contiene los componentes que proveen los contexts a la aplicación y el sistema principal de navegación.

```
const Stack = createStackNavigator();

const App = (props) => {
  return (
    <UserContextProvider>
      <PusherContextProvider>
        <NavigationContainer>
          <Stack.Navigator>
            <Stack.Screen
              name="navigation"
              component={Navigation}
              options={{ headerShown: false }}
            />
            <Stack.Screen
              name="screen-chat"···
              })}
            />
            <Stack.Screen
              name="info-chat"···
              options={{ title: "Info" }}
            />
          </Stack.Navigator>
        </NavigationContainer>
      </PusherContextProvider>
    </UserContextProvider>
  );
};
```

Componentes de App.js

El primer context que se aprecia es el de usuario, el cual es un componente en el que se podrán guardar todos los datos que devuelva el servidor en referencia al usuario si la sesión ha sido iniciada, o estará vacío en caso contrario. Al englobar toda la aplicación puede usarse en cualquier parte de ella. En cuanto a su uso es importante resaltar que es de lectura y escritura, ya que permite no solamente acceder a los datos que tiene guardados, sino también cambiar su valor en todo momento.

```
const Context = createContext({});

export function UserContextProvider({ children }) {
  const [user, setUser] = useState(null);
  return (
    <Context.Provider value={{ user, setUser }}>{children}</Context.Provider>
  );
}

export default Context;
```

Código de UserContext

En este código vemos un hook de React. Los hooks son elementos que permiten usar el estado y otras características de React sin escribir una clase. [53] El hook que vemos es useState, que es uno de los más utilizados y básicamente sirve para almacenar el estado de una variable y cambiar dicho estado en cualquier momento mediante la función set. La variable user y la función setUser se pasan como prop del Context. Provider y de esta manera se podrán utilizar en cualquier parte de la app. Por último se exporta el componente para que pueda ser importado en cualquier otro componente. Dentro de las etiquetas del Context.Provider vemos la prop children, que sirve para conseguir que se le pueda pasar un componente completo cualquiera.

El segundo context es exactamente igual al primero, pero almacenará la información relativa a la configuración de conexión de pusher.

Siguiendo con la imagen del código de App.js vemos que dentro de los contexts hay un componente de navegación que alberga tres posibles destinos. El componente de navegación ha sido creado previamente y será el navegador principal de la aplicación. Uno de los destinos es el componente de navegación de tabs (Tab navigator), y los dos restantes son las pantallas en las que no queremos ver el menú de navegación, en este caso las pantallas que albergan la sala de chat y de información de la sala de chat. Se ha decidido realizar así porque en la pantalla de chat resulta molesto ver el menú de tabs. Y por otra parte, a la pantalla de información se accede desde la de chat, de modo que podría resultar extraño para el usuario acceder desde el menú a una pantalla en la cual no se ve el menú y desde esa otra acceder a una que se vuelve a

visualizar el menú. A cada uno de los destinos le pasaremos por props el nombre del destino, la pantalla que le corresponden, y algunas opciones adicionales como las correspondientes a la barra superior de la aplicación.

Tab navigator

Este menú agrupará las pestañas de home y de chats. La de home es la pestaña en la cual se inicia la aplicación cada vez que se abre, y en ella el usuario encontrará información sobre su perfil como su nombre, email o foto, siempre y cuando haya iniciado sesión; por otra parte, en la de chats encontrará la lista de chats a los que pertenece, pero en el mismo caso que la otra pestaña, debe haberse autentificado. El componente se ha escrito de la siguiente manera:

```
const Tab = createBottomTabNavigator();

const Navigation = (props) => {
  return (
    <Tab.Navigator
      initialRouteName="account"
      tabBarOptions={{ ···
      }}
      screenOptions={({ route }) => ({
        tabBarIcon: ({ color }) => screenOptions(route, color),
      })}

      <Tab.Screen
        name="chats"
        component={ChatsStack}
        options={{ title: "Chats" }}
      />
      <Tab.Screen
        name="account"
        component={AccountStack}
        options={{ title: "Cuenta" }}
      />
    </Tab.Navigator>
  );
};
```

Código de Tab Navigator

El código es muy parecido al navegador principal de la aplicación, la única curiosidad es como se han añadido los iconos a la barra en la que se muestras las pestañas, para ello se puede apreciar que se utiliza una función llamada screenOptions, cuyo contenido se muestra en la siguiente imagen:

```
function screenOptions(route, color) {
  let iconName;

  switch (route.name) {
    case "chats":
      iconName = "chat-outline";
      break;
    case "account":
      iconName = "home-outline";
      break;
    default:
      break;
  }
  return (
    <Icon type="material-community" name={iconName} size={22} color={color} />
  );
}
```

Código de screenOptions

En la función se puede observar que devuelve un icono de la librería material-community, cuyo nombre se define mediante un switch (control de selección) en función de la ruta en la que nos encontremos.

AccountStacks

Este es el componente que contiene todas las pantallas que se pueden mostrar en la pestaña de home. En cierto modo es un navegador dentro de tab navigator, por lo que su sintaxis es muy parecida.

Las pantallas que contiene son la de login (inicio de sesión) y la de información de usuario (denominada como Account).

Componente Account

En un primer momento, sin estar logueado, nos mostrará una pantalla de bienvenida, y desde ella se podrá navegar a la pantalla de login, mientras que si ya estamos logueados nos llevará a la pantalla de información de usuario.

Su código se muestra a continuación y contiene elementos bastante interesantes de los que hablar.

```
const Account = (props) => {
  const { user, setUser } = useContext(UserContext);

  useFocusEffect(
    useCallback(() => {
      const _retrieveData = async () => {
        const value = await AsyncStorage.getItem("token");
        if (value) {
          try {
            let response = await fetch(BaseUrl + "/user", {
              method: "GET",
              headers: {
                Authorization: "Bearer " + value,
                Accept: "application/json",
                "Content-Type": "application/json",
              },
            });
            let json = await response.json();
            setUser(json.data);
          } catch (error) {···
          }
        }
      };
      _retrieveData();
    }, [])
  );

  return user ? <UserLogged /> : <UserGuest />;
};
```

Código del componente Account

En un primer vistazo podemos ver que devuelve un componente u otro en función del parámetro user, esto se conoce como ternaria y es una de las técnicas más utilizadas en React Native por su potencia y sencillez.

Se puede observar también que capta el context del usuario mediante un nuevo hook conocido como useContext, que como se puede deducir fácilmente sirve para almacenar el contenido del context que se le pasa como parámetro.

También vemos que realiza una petición HTTP con el método GET para obtener los datos del usuario. Esta petición se realiza con la función fetch y es de tipo asíncrona. A la función fetch hay que pasarle como argumentos la URL a la que se le realiza la petición, el método de la petición y las cabeceras. En algunos casos también se le puede pasar el contenido, pero al ser tipo GET no lo necesita. La parte más importante de la cabecera es la autorización, ya que será la forma de pasarle el token al servidor para que éste compruebe si es válido. El token se toma de la memoria de la aplicación con la sentencia AsyncStorage. Después, se guardará la respuesta en el context de usuario (variable llamada user en este componente). De este modo, si ya se había iniciado sesión habrá un token en la memoria, el servidor podrá devolver un objeto en formato JSON con los datos del usuario, se almacenará en el context de usuario y por último se renderizará el componente de usuario logueado en

esta pantalla; en caso contrario, el valor de la variable user será nulo y se renderizará el componente de usuario invitado.

Por último, vemos que toda la petición HTTP está dentro de una función y esa misma función es llamada dentro de otro hook llamado useCallback. Los callbacks le asignan una función como parámetro a otra función. En React Native se utiliza este hook para devolver una versión memorizada del callback que sólo cambia si una de las dependencias ha cambiado [54]. Este hook está, a su vez, dentro de otro hook conocido como useFocusEffect. Teniendo este último un uso muy particular, ya que se utiliza para ejecutar todo lo que hay dentro de él cuando se carga por primera vez una pantalla. Normalmente se utilizaría el hook useEffect para hacer esto, pero en el caso de ser una pantalla de un menú de tabs useEffect no funciona, de ahí que su uso sea bastante particular.

El componente userLogged también contiene alguna que otra parte interesante en su código, que como siempre, se muestra a continuación:

```
const navigation = useNavigation();

const logout = async () => {
  try {
    await AsyncStorage.removeItem("token");
    setUser(null);
    navigation.navigate("account");
  } catch (error) {···
  }
};

useEffect(() => {
  obtenerDatos();
}, []);

const obtenerDatos = async () => {
  const value = await AsyncStorage.getItem("token");
  // const aux_pusher = new PusherClient(value);
  setPusher(new PusherClient(value));

  console.log(pusher);
  try {
    let response = await fetch(BaseUrl + "/user", {
      method: "GET",
      headers: {
        Authorization: "Bearer " + value,
        Accept: "application/json",
        "Content-Type": "application/json",
      },
    });
    let json = await response.json();
    setUserInfo(json);
  } catch (error) {
    console.error(error);
  }
};
```

Código de useLogged

Vemos una función llamada logout que se utiliza para cerrar sesión de forma manual. Aunque recordemos que si no se hace de forma manual se cerrará automáticamente a las 2 horas, que es el tiempo transcurrido desde la creación del token hasta su caducidad. Esta función únicamente borra el token de la

memoria y los datos del usuario almacenados en la aplicación, y nos redirige a la pantalla de bienvenida, desde la cual podremos acceder a iniciar sesión.

Seguidamente en el código tenemos una función dentro del hook useEffect. Este hook ejecuta lo que haya en su interior la primera vez que se carga el componente, y también se vuelve a ejecutar cada vez que cambie el parámetro que le pasemos dentro de los corchetes. Como en este caso no se le pasa ningún parámetro, solo se ejecutará la primera vez que se cargue el componente.

La función que hay dentro del useEffect es una petición HTTP del tipo GET como la explicada anteriormente. Pero posee una particularidad, que es la inicialización del cliente Pusher. Este cliente está declarado en una clase, que se ha creado de la siguiente manera:

```
let APP_KEY = "54TmwEk3KVTFT83jYsycdtNrYTzKvxHzKEY";
let AUTH_URL = "http://15.237.84.107/api/";
let WEBSOCKET_URL = "15.237.84.107";
let WEBSOCKET_PORT = "6001";

export default (token) => {
  let options = {
    encrypted: false,
    key: APP_KEY,
    wsHost: WEBSOCKET_URL,
    wsPort: WEBSOCKET_PORT,
    disableStats: true,
    authEndpoint: AUTH_URL + "broadcasting/auth",
    forceTLS: false,
    auth: {
      headers: {
        Authorization: "Bearer " + token,
      },
    },
    // enabledTransports: ["ws"],
  };
  return new Pusher(options.key, options);
};
```

Código de pusherClient

Únicamente vemos que se le han asignado los datos del servidor necesarios para su correcto funcionamiento. Entre estos datos destacan la URL de escucha, el puerto de escucha, la clave de acceso o la autentificación mediante token. De esta forma los datos del cliente quedan guardados en un context, y así el usuario puede suscribirse al canal de difusión cuando lo desee.

Volviendo ahora al código de userLogged, se ha explicado la parte lógica del componente, pero no del método render (return del componente), que se encargará de mostrar por pantalla la información del usuario. En esta renderización podemos resaltar el componente que contiene la imagen de perfil

y permite editar seleccionando una imagen de la galería del dispositivo móvil. Para gestionar el acceso a la galería se ha implementado la siguiente función:

```
const changeAvatar = async () => {
  const resultPermissions = await Permissions.askAsync(
    Permissions.CAMERA_ROLL
  );
  const resultPermissionsCamera =
    resultPermissions.permissions.mediaLibrary.status;

  if (resultPermissionsCamera === "denied") {
    console.log("permiso denegado");
  } else {
    const result = await ImagePicker.launchImageLibraryAsync({
      allowsEditing: true,
      aspect: [4, 3],
    });

    if (result.cancelled) {
      console.log("Has cerrado la selección");
    } else {
      uploadImage(result.uri);
    }
  }
};
```

Código de acceso a la galería de imágenes del móvil

En esta función se puede ver cómo se gestionan los permisos de acceso a la galería del teléfono, para que en el caso de que el usuario deniegue dichos permisos la aplicación respete su decisión y no acceda a la galería. Pero en el caso de haber aceptado los permisos se ejecutará un proceso asíncrono que acabará subiendo la imagen al servidor mediante la función uploadImage.

Dicha función es una petición HTTP como las ya explicadas, pero será del método PUT, ya que queremos actualizar la imagen existente en el servidor.

Otro punto importante de la imagen de perfil se da a la hora de representarla por pantalla en nuestra aplicación. Tras haber recibido la respuesta del servidor a una petición GET que contiene los datos de los usuarios, vemos que la imagen no se devuelve en un formato que React Native pueda representar. Para su correcto uso tiene que ser convertida a base64, que se realiza de la siguiente manera:

```
function convertImageUrl(blob) {
  var reader = new FileReader();
  reader.readAsDataURL(blob);
  reader.onloadend = function () {
    var base64data = reader.result;
    setImage(base64data);
  };
}
```

Conversión a base64

Esto es todo lo relevante en relación al componente userLogged, pero anteriormente dijimos que en el caso de no haber iniciado sesión la aplicación mostraría un componente de bienvenida que nos dirigiría a la pantalla login que permite al usuario autentificarse con sus credenciales.

Inicio de sesión

Una vez dentro de la pantalla de login encontraremos un campo de texto para el email, otro para la contraseña y un botón para enviar los datos de los campos al servidor mediante, nuevamente una petición HTTP de tipo POST. Por lo tanto, lo único interesante de esta parte son los campos de texto y cómo se gestionan los cambios que se producen en ellos.

```
<Input
  placeholder="contraseña"
  containerStyle={styles.inputForm}
  onChange={(e) => onChange(e, "password")}
  password={true}
  secureTextEntry={showPassword ? false : true}
  rightIcon={
    <Icon
      type="material-community"
      name={showPassword ? "eye-off-outline" : "eye-outline"}
      iconStyle={styles.iconRight}
      onPress={() => setShowPassword(!showPassword)}
    />
  }
/>
```

Código de un campo de texto

En la imagen superior se muestra el campo correspondiente a la contraseña. De las props recibidas podemos destacar dos que están muy relacionadas entre sí, que son el rightIcon y secureTextEntry. La primera prop es el icono que se colocará en la parte derecha del campo y que, mediante la gestión del estado de la variable showPassword, cada vez que se haga click sobre él se cambiará el propio icono y la otra prop mencionada. Su objetivo es mostrar u ocultar la contraseña, por eso hacemos que el icono cambie y así indicarle al usuario si la contraseña es visible o no también mediante el icono.

Aparte de esas props, también es muy relevante la llamada onChange, que gestionará instantáneamente los cambios que se produzcan en el campo mediante el evento que genera. Es decir, para cada carácter que se escriba en el campo se actualizará el estado de este, para conseguir eso hay que

mantener los caracteres existentes en una variable (llamada formData) y añadirle el nuevo carácter que se haya escrito, de la siguiente manera:

```
const onChange = (e, type) => {
  // setFormData({ [type]: e.nativeEvent.text }); -> no mantiene los datos
  setFormData({ ...formData, [type]: e.nativeEvent.text }); // para mantener los datos y añadir los nuevos
};
```

Gestión del evento de cambio

El formulario del email es muy similar a este, solamente cambia en que el icono de la derecha es fijo, y que debe validarse que el campo introducido es un email.

Lista de chats

Hasta aquí toda la parte del apartado home, ahora pasaremos a comentar el apartado de los chats, donde en un primer momento y siempre que nos hayamos autenticado veremos una lista con las salas de chats disponibles. En el caso de estar autenticados también veremos el botón de añadir nuevo chat

Para poder representar la mencionada lista lo primero que necesitamos es pedirle la información al servidor mediante, cómo no, una petición HTTP. La respuesta del servidor se guardará en un array de objetos, que se llamará chats en el código. Para renderizar este array utilizaremos un componente que incluye React Native llamado Flatlist, al que se le pasará como parámetro el array y él mismo renderizará de uno en uno todos los objetos del array. Abajo se ejemplifica este componente para una mejor comprensión de lo explicado.

```
<FlatList
  data={chats}
  renderItem={(chat) => (
    <Chat chat={chat} token={token} navigation={navigation} />
  )}
  keyExtractor={(item, index) => index.toString()}
  onEndReachedThreshold={0.5}
/>
```

Lista de chats

Vemos que la prop renderItem recibe un componente chat, en el que se ordenará la información de cada uno de los objetos para que el usuario pueda visualizarla de forma cómoda. Por su parte, el componente chat estará formado componentes ya explicados, básicamente texto y una imagen. Lo único destacable es que está envuelto por un componente propio de React Native llamado TouchableOpacity, que consigue al hacer click sobre el ítem de la lista

en cuestión el usuario navegue a la pantalla del chat correspondiente a dicho ítem.

Pantalla de chat

Esta pantalla constará de dos componentes principales. Uno encargado de representar todos los mensajes enviados a esa sala de chat y otro que permitirá el envío de nuevos mensajes. Su código alberga algunos puntos muy interesantes de comentar.

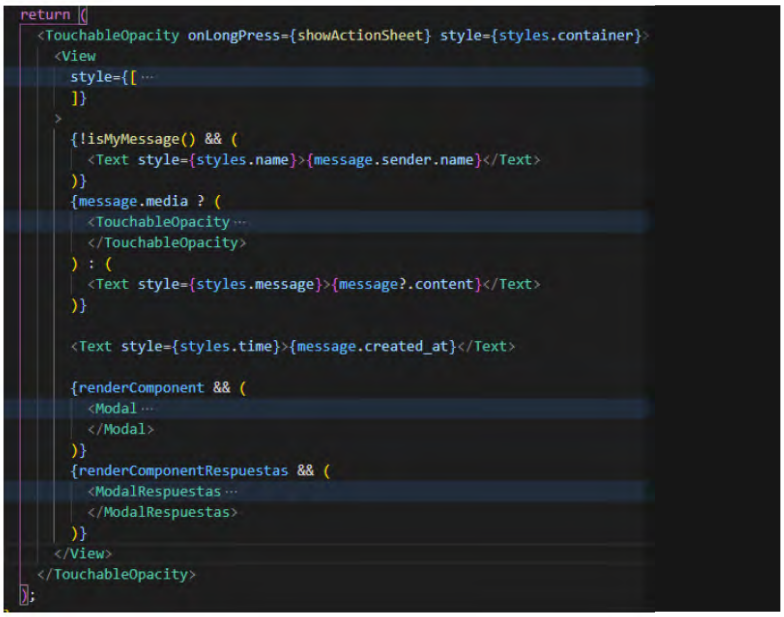
En primer lugar, la lista de mensajes se muestra de forma muy similar a la que lo hacían la lista de chats, pero tienen la particularidad de que solo carga un número limitado de 25 mensajes en cada petición que se realiza al servidor. Así logramos una mejor optimización, ya que no sabemos cuántos mensajes puede llegar a tener un chat, y si cargamos todos de una vez la aplicación puede volverse excesivamente lenta. La idea es cargar los mensajes a medida que se hace scroll de los mensajes existentes, de esta manera cada vez que el usuario llegue al último mensaje cargado hasta la fecha, se enviará una petición al servidor para que devuelva otros 25 mensajes. Esto es posible gracias a una prop de Flatlist, conocida como onEndReached.

```
<FlatList
  data={messages}
  renderItem={({ item }) => (
    <Message answerId={answerId} setAnswerId={setAnswerId} message={item} />
  )}
  keyExtractor={(item) => item?.id?.toString()}
  onEndReachedThreshold={0.5}
  inverted
  onEndReached={getNextMessages}
  ListFooterComponent={next && <Text>Cargando...</Text>}
/>
```

Lista de mensajes dentro de un chat

En el código de la imagen anterior apreciamos que la prop onEndReached recibe una función que es, como es obvio, la petición al servidor para que devuelva los 25 siguientes mensajes. También es bastante adecuado indicarle al usuario que el servidor está enviando nuevos mensajes, ahí es donde entra la prop ListFooterComponent, el cual se renderiza cada vez que el usuario llega último mensaje en la aplicación pero no al último del servidor, es decir, se ejecuta justamente mientras el servidor está enviando la respuesta.

En la lista de mensajes el ítem que se renderiza es un componente llamado mensaje, del cual hay algunos puntos interesantes a destacar en su código.



Código de message

Una parte importante es la forma en la que renderiza el mensaje. Y esto es debido a que el mensaje se colocará a la derecha de la pantalla en caso de enviarlo el usuario de la aplicación, y a la izquierda en el caso de que lo envíe otro usuario distinto al que está viendo la pantalla. Además en el caso de que sea otro usuario también aparecerá el nombre de éste. La función isMyMessage es la que le dice al componente si el mensaje es del usuario actual o no.

También importa el tipo de mensaje, ya que éste puede ser un texto o una imagen. Esta información ya viene dada desde el servidor. En el caso de que sea una imagen debe convertirse a formato base64 y obtener una previsualización de la imagen dentro de un mensaje. Para esta previsualización el servidor devuelve un thumbnail de la imagen, de esta forma la imagen pesa menos y así tendremos mejor optimizada la aplicación.

Al hacer click sobre una de estas imágenes se abrirá en un modal a mayor tamaño. Para cerrar el modal únicamente hay que hacer click fuera de él.

Otro punto de interés reside en que el mensaje puede mantenerse pulsado (aproximadamente medio segundo) para desplegar un menú conocido como actionSheet, que permite la posibilidad de realizar dos acciones diferentes.

Código del menú actionSheet

Una de las acciones es ver las respuestas al mensaje, que se visualizarán en un modal al igual que las imágenes.

La otra acción que permite es, como no puede ser de otra manera, crear las respuestas de las que se habla en el anterior párrafo. Para conseguir esto únicamente hay que indicárselo al servidor a la hora de hacer la petición HTTP.

Pero para enviar una respuesta antes debe ser posible enviar un mensaje cualquiera, y esto se hace en un componente que consta de un campo de texto y de un botón de envío. El código llevado a cabo es exactamente igual que el que se utilizaba en los campos de iniciar sesión. También permite acceder a la galería y enviar imágenes, también de la misma manera que ya conocemos.

Con lo explicado hasta ahora sobre esta pantalla sería posible enviar y recibir mensajes, pero no en tiempo real. Aquí es donde entra en juego la parte más importante de esta pantalla (y probablemente de toda la aplicación), la suscripción a los canales de difusión por los que Websockets se comunica en tiempo real con el cliente.

Suscripción a Websockets

Este código es más sencillo de lo que en un primer momento pueda parecer. En primer lugar, el cliente pusher guardado en un context de la aplicación es suscrito a un canal privado de la sala de chat en la que se encuentra. Y después, se pone en escucha mediante bind, para poder recibir los nuevos mensajes en tiempo real. Por último, se cancela la suscripción cuando el usuario deja la pantalla del chat.

InfoChat

A esta pantalla se navega haciendo click en la imagen de barra superior de una sala de chat. No incluye nada que no se haya explicado hasta la fecha. Únicamente contendrá la imagen del chat y una lista con los usuarios.

Crear chat

En la pantalla de la lista de chats había un botón que servía para añadir un nuevo chat. Al pulsarlo te lleva a una nueva pantalla que te permite elegir el tipo de chat mediante una serie de selectores.

```
<Picker
  selectedValue={roomType}
  style={{ height: 50, width: "50%" }}
  onValueChange={(itemValue, itemIndex) => setRoomType(itemValue)}
>
  <Picker.Item label="privado" value="privado" />
  <Picker.Item label="grupo" value="grupo" />
</Picker>
```

Código de un selector

El código del selector es muy sencillo, únicamente hay que crearlo y añadir dentro tantas opciones como se deseen.

Habrá un total de 3 selectores. El primero indicará si el chat es privado o de grupo. El segundo sólo se activará en el caso de que el chat sea de grupo y permitirá elegir entre un grupo con cualquier usuario o uno que directamente filtre a los usuarios por asignatura. El tercero permitirá escoger la asignatura que se desea, pero obviamente sólo en el caso de que el grupo sea de tipo asignatura.

De cualquier manera se puede avanzar a la siguiente pantalla y elegir el usuario, o usuarios que deseamos que formen parte de la sala de chat. Esta pantalla cuenta con un campo de texto para introducir el nombre de la sala de

chat y una lista con los usuarios que pueden formar parte de dicha sala. Además tiene un botón para crear el chat en el caso que todo sea correcto.

Parece igual que algunos de los componentes ya creados, pero la lista tiene una particularidad, y es que permite seleccionar a los usuarios y guardarlos en un estado. Cada elemento de la lista será en realidad un checkbox acompañado de una descripción (el nombre del usuario). El código del checkbox se muestra a continuación:

```
<BouncyCheckbox
  size={25}
  fillColor="red"
  unfillColor="#FFFFFF"
  text={itemid}
  iconStyle={{ borderColor: "red" }}
  onPress={() => {
    if (selectedUsers.some((item) => item == itemid)) {
      setSelectedUsers((prevData) =>
        prevData.filter((item) => item != itemid)
      );
    } else {
      setSelectedUsers((prevData) => [...prevData, itemid]);
    }
  }}
/>
```

Código de la creación de un chat

Vemos que al pulsar el checkbox el programa comprueba si el usuario ya está en la lista de seleccionados. En caso de no estar se añade a la lista, y en caso contrario se borra de ésta.

6. RESULTADOS

En este apartado se expondrán algunas de las pantallas de la aplicación para mostrar el resultado de éstas.

Al iniciar la aplicación, seremos dirigidos a la pantalla de inicio de sesión, que nos pedirá nuestros credenciales.

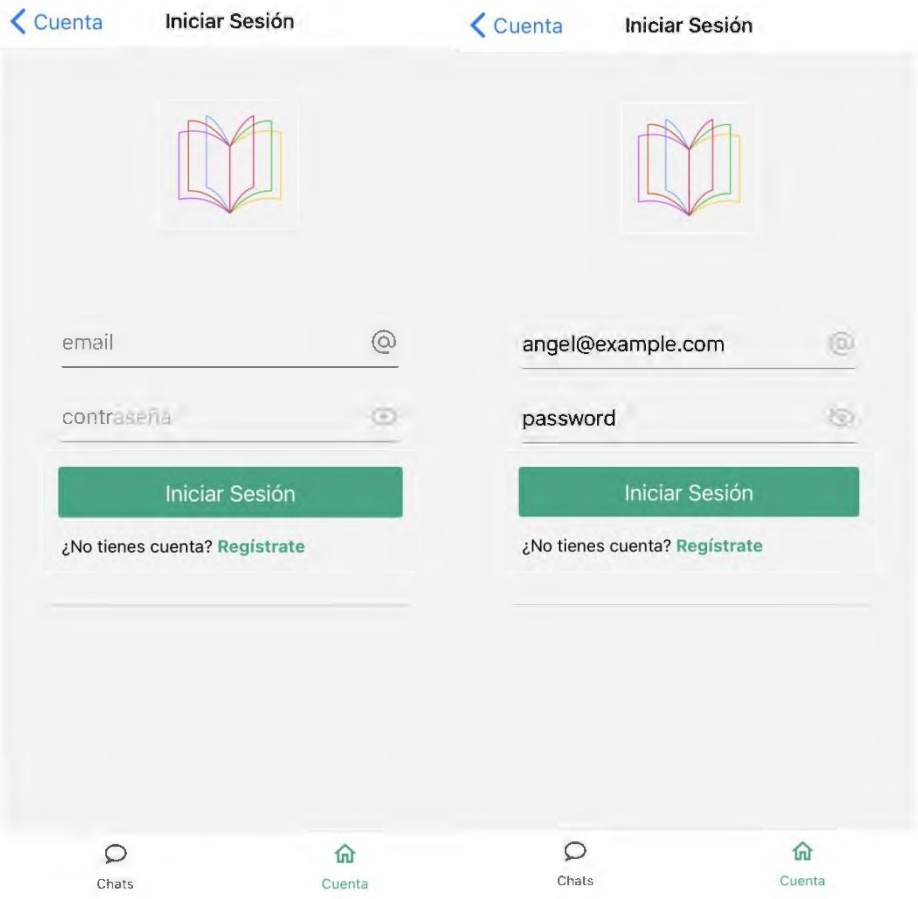
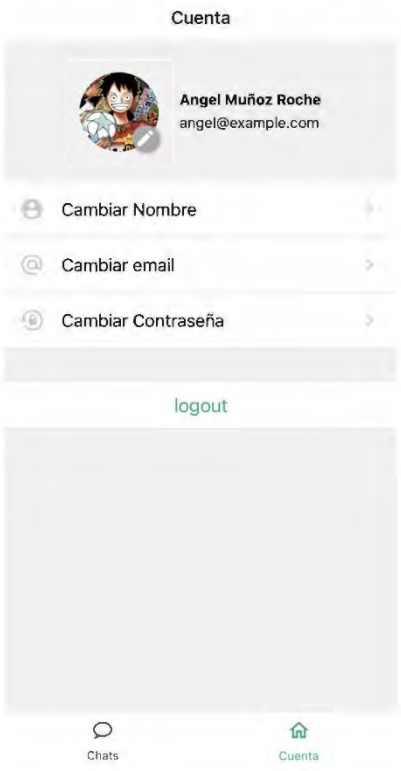


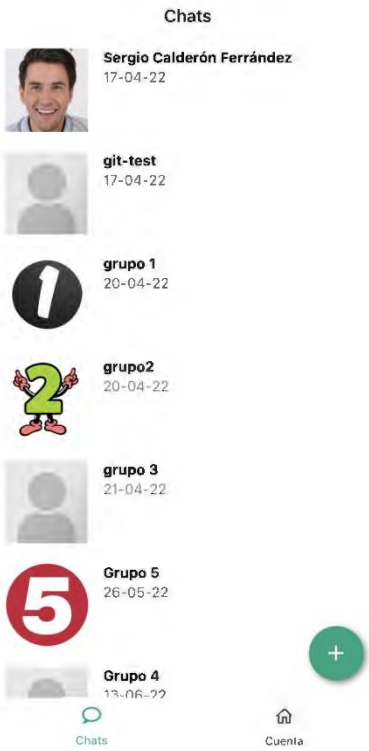
Imagen de la pantalla de inicio de sesión

Al enviar los datos, si son correctos, seremos dirigidos a la pantalla de perfil de usuario. Que cuenta con la imagen de perfil y una serie de datos como el correo y el nombre.



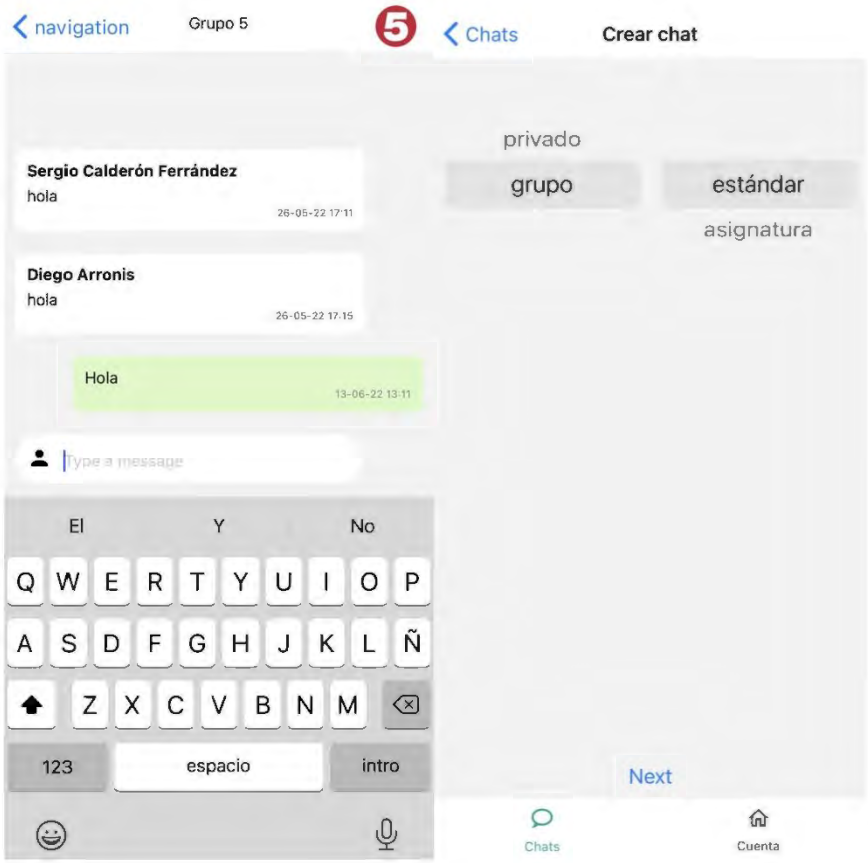
Pantalla de perfil de usuario

Todo lo mostrado hasta ahora se encontraba en la pestaña de cuenta, pero en el menú inferior podemos apreciar también la pestaña de chats. En esta pestaña, tras iniciar sesión, aparecerá una lista de chats del usuario.



Pantalla de lista de chats

Si pulsamos sobre un chat, seremos dirigidos a la sala de chat en cuestión, donde podremos leer los mensajes presentes en dicha sala, y enviar nuevos mensajes. Asimismo, desde la pantalla de lista de chats podemos acceder a la pantalla de creación de chats pulsando el botón con el icono de “más”. Esta pantalla permitirá elegir el tipo de sala que se desee crear.



Pantallas de chat y de creación de sala de chat

En la pantalla de chat podemos observar una lista de mensajes cuyo diseño dependerá de si el mensaje ha sido enviado por el usuario actual o por otros usuarios. También vemos un campo de texto que permitirá enviar un nuevo mensaje.

7. CONCLUSIONES Y LÍNEAS FUTURAS

En el presente proyecto se ha llevado a cabo todo el desarrollo de una aplicación móvil. Se ha desarrollado tanto la parte del servidor como la parte del cliente. La conclusión a la que ha llegado es que se ha visto cumplido el principal objetivo de la aplicación, que era crear un chat funcional en tiempo real con una interfaz sencilla e intuitiva. A su vez, se cumplen los requisitos y casos de uso expuestos en el tercer apartado de esta memoria.

La aplicación desarrollada, a pesar de ser completamente funcional, es también una buena base de cara a futuras implementaciones que se pueden realizar sobre ella.

La principal mejora que se podría implementar sobre la aplicación sería el uso de inteligencia artificial en las salas de chat de las asignaturas de tal forma que cuando un alumno pregunte algo que ya ha sido respondido la aplicación le referencie los mensajes relacionados con dicha pregunta. Lo interesante en este caso sería guardar la información de una asignatura para utilizarla en los años posteriores, y en de esta forma en los siguientes años la aplicación sería capaz de responder a prácticamente cualquier duda que se haya planteado. También se les daría a los profesores la opción de destacar y relacionar mensajes, y así ir entrenando a la red neuronal que se haya implementado.

También es importante mencionar que a la hora de implementar la aplicación en un centro académico real habría que utilizar los datos reales de la base de datos de ese centro, por lo que otra mejora importante sería un proceso que automatizara lo máximo posible esa transferencia de datos y así facilitarle la tarea a los administradores de la aplicación designados por el centro.

8. BIBLIOGRAFÍA

[1] https://nodejs.org/es/

[2] https://expressjs.com/

[3] https://www.startechup.com/es/blog/express-js-what-it-is-used-for-and-when-where-to-use-it-for-your-enterprise-app-development/

[4] https://openwebinars.net/blog/que-es-flask/

[5] https://desarrolloweb.com/articulos/que-es-mvc.html

[6] https://j2logo.com/flask/tutorial-como-crear-api-rest-python-con-flask/#:~:text=Flask%20Restful%3A%20Es%20una%20extensi%C3%B3n,a%20trav%C3%A9s%20de%20su%20ORM.

[7] https://flask-restful.readthedocs.io/en/latest/

[8] https://www.excella.com/insights/creating-a-restful-api-django-rest-framework-vs-flask

[9] https://openwebinars.net/blog/que-es-laravel-caracteristicas-y-ventajas/

[10] https://kotlinlang.org/docs/android-overview.html

[11] https://kotlinlang.org/

[12] https://www.apple.com/es/swift/

[13] https://aurestic.es/que-es-flutter/

[14] https://inlab.fib.upc.edu/es/blog/que-es-el-lenguaje-de-programacion-dart

[15] https://reactnative.dev/

[16] https://www2.deloitte.com/es/es/pages/technology/articles/que-es-react-native.html

[17] https://rafarjonilla.com/que-es/backend/

[18] https://www.arimetrics.com/glosario-digital/backend

[19] https://geekytheory.com/que-es-una-api-rest-y-para-que-se-utiliza/

[20] https://www.arsys.es/blog/programacion/proyectar-un-backend-hoy

[21] https://www.php.net/manual/es/intro-whatis.php

[22] https://openwebinars.net/blog/que-es-laravel-caracteristicas-y-ventajas/

[23] https://www.qualitydevs.com/2021/06/23/que-es-laravel/

[24] https://www.hostinger.es/tutoriales/que-es-mysql

[25] https://www.oracle.com/es/database/what-is-database/

26] https://www.oracle.com/es/database/what-is-a-relational-database/

[27] https://docs.microsoft.com/es-es/office/troubleshoot/access/define-table-relationships

[28] https://developer.mozilla.org/es/docs/Web/API/WebSockets_API

[29] https://virtumedia.wordpress.com/2020/06/05/laravel-broadcasting-con-pusher-y-laravel-echo/

[30] https://www.servnet.mx/blog/backend-y-frontend-partes-fundamentales-de-la-programaci%C3%B3n-de-una-aplicaci%C3%B3n-web

[31] https://es.reactjs.org/

[32] https://www2.deloitte.com/es/es/pages/technology/articles/que-es-react-native.html

[33] https://www.hostinger.es/tutoriales/que-es-npm

[34] https://lodash.com/

[35] https://reactnative.dev/docs/navigation

[36] https://reactnativeelements.com/

[37] https://www.mysql.com/products/workbench/

[38] https://visualstudio.microsoft.com/es/

[39] https://medium.com/leopark-lab/por-qu%C3%A9-desarrollar-apps-de-react-native-en-expo-2e1b83e4d00a

[40] https://richos.gitbooks.io/laravel-5/content/capitulos/chapter4.html

[41] https://norvicsoftware.com/base-de-datos-y-migraciones-en-laravel-8/

[42] https://es.wikieducator.org/Usuario:ManuelRomero/Laravel/componentes_controladores

[43] https://styde.net/creacion-y-uso-de-controladores-en-laravel-5-1-2/

[44] https://norvicsoftware.com/vistas-y-rutas-en-laravel-8/#:~:text=Las%20rutas%20en%20Laravel%20son,HTTP%20hacia%20una%20URL%20concreta.

[45] https://gabrielchavez.me/eventos-laravel/#:~:text=cuando%20publique%20material.-,%C2%BFQu%C3%A9%20son%20los%20eventos%20en%20Laravel%3F,podemos%20hacer%20con%20los%20eventos

[46] https://code.tutsplus.com/es/tutorials/gates-and-policies-in-laravel--cms-29780

[47] https://aulasoftwarelibre.github.io/taller-de-react-native-docs/project-structure/

[48] https://lenguajejs.com/npm/administracion/carpeta-node_modules/

[49] https://docs.npmjs.com/cli/v7/configuring-npm/package-json

[50] https://docs.npmjs.com/cli/v8/configuring-npm/package-lock-json

[51] https://es.reactjs.org/docs/context.html

[52] https://en.wikipedia.org/wiki/Dummy_data

[53] https://es.reactjs.org/docs/hooks-intro.html

[54] https://es.reactjs.org/docs/hooks-reference.html#usecallback