UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

APLICACIÓN WEB CON REACT Y LARAVEL PARA PLATAFORMA ACADÉMICA EN TIEMPO REAL

TRABAJO FIN DE GRADO

Junio - 2022

AUTOR: Sergio Calderón Ferrández

DIRECTOR/ES: Miguel Onofre Martínez Rach

RESUMEN

En estos últimos años se está llevando a cabo una gran revolución tecnológica que está cambiando notablemente muchos ámbitos de nuestra vida, incluido el sector educativo y laboral. Esto ha creado la necesidad de actualizar y desarrollar herramientas que nos permitan aumentar nuestra productividad a la vez que facilitarnos las tareas de dichas actividades.

Por esa razón, como Trabajo de Fin de Grado (TFG) se ha realizado una aplicación web que preste un servicio de comunicación en tiempo real a los estudiantes y profesores de un mismo centro educativo mediante un sistema de chat interno, en el cual los estudiantes puedan compartir sus dudas y estas ser resueltas por sus compañeros o profesores. Además, las respuestas a estas preguntas generarán hilos automáticos dentro del chat, que permitirán crear conversaciones organizadas en torno a las preguntas planteadas, como si de un foro se tratase. Asimismo, estos hilos podrán ser elegidos por los profesores responsables de las asignaturas para que sean mostrados a estudiantes de otros cursos en la web de la asignatura.

El presente documento contiene la memoria del Trabajo Fin de Grado donde se encuentra toda la información necesaria para comprender el proceso de desarrollo del proyecto, las tecnologías utilizadas, la arquitectura, el diseño y los resultados finales obtenidos, así como una serie de conclusiones sobre el proyecto y mejoras futuras que se pueden implementar en la aplicación.

Palabras clave

- Aplicación web.
- Chat.
- React.
- Laravel.
- API Rest.
- Websocket.
- PHP.
- HTML.
- JavaScript.
- Backend.
- Frontend.

ABSTRACT

Recently, a great technological revolution is taking place, which is notably changing many areas of our lives, including the educational and working sectors. This has created the urge to update and develop new tools which allow us to increase our productivity and making these activities easier.

Because of this issue, as my Final Degree Project I have decided to make a web application for a real-time communication service between students and teachers within the same educational center. Therefore I created an internal chat system, where students can ask their doubts to their own classmates or teachers. In addition, the answers will generate automatic threads within the chat, which will generate organized issues around the exposed questions, like a forum. As a result, these threads can be chosen by the teachers so they can be shown to the students from other courses on the subject's website.

This document contains the memory of the Final Degree Project where all the necessary information can be found to understand the project development process, the technologies used, the architecture, the design and the final results obtained, as well as a series of conclusions about the project and future improvements that can be implemented in the application.

Keywords

- Web Application.
- Chat.
- React.
- Laravel.
- API Rest.
- Websocket.
- PHP.
- HTML.
- JavaScript.
- Backend.
- Frontend.

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

1. INTRODUCCIÓN

En este primer capítulo se expone el contexto y los objetivos por los que se desarrolla el proyecto, así como una mención al desarrollo en la aplicación y se explica la estructura que seguirá este documento.

1.1. CONTEXTO

En la actualidad nos encontramos inmersos en una revolución tecnológica, en la cual el avance tecnológico y su aplicación en los diferentes ámbitos de la sociedad están causando una transformación en el mundo tal y como lo conocemos, este proceso de digitalización afecta a todos los sectores, incluido el sector educativo, en el cual las nuevas tecnologías han ido ganando gradualmente protagonismo. Además, la llegada de la pandemia del COVID-19, que causó el cierre de los centros educativos y trasladó la docencia a los hogares de manera online, ha provocado que estas nuevas herramientas tecnológicas sean indispensables para la educación. Sin embargo, la pandemia no hizo más que acelerar un proceso de transformación digital que ya se estaba produciendo.

Estas nuevas tecnologías de la información y comunicación no son solo un mecanismo más de aprendizaje o una herramienta, sino que además son una forma de desarrollar aptitudes tecnológicas, nuevas formas de pensar y actuar para el desarrollo integral de los estudiantes.

Las TIC han cambiado radicalmente la manera en la que nos comunicamos los seres humanos. Cada vez menos interacciones sociales ocurren en vivo y en directo, y más allá del potencial peligro que ello supone, la era de la inmediatez obliga a que se practique una interrelación más veloz e instantánea.

Por ello, este TFG surge de la necesidad detectada a través de la experiencia obtenida a lo largo de mi paso por el sistema educativo, en el que las nuevas tecnologías de la información y comunicación están cada vez más presentes, pero no aún del todo implementadas.

Para cubrir esta necesidad, el presente proyecto tiene la pretensión de desarrollar un sistema que permita la comunicación en tiempo real entre los integrantes de un centro docente.

1.2. OBJETIVOS

El objetivo principal de este Trabajo de Fin de Grado es la creación y desarrollo de un prototipo de aplicación web que preste un servicio de comunicación en tiempo real, rápido, fácil y seguro a los estudiantes y profesores de un mismo centro educativo mediante un sistema de chat interno, en el cual los estudiantes puedan plantear sus dudas en los grupos de chat creados para cada una de las asignaturas y estas puedan ser resueltas por sus compañeros o profesores. Además, el chat será capaz de crear hilos para las preguntas que se propongan en los grupos de chat, lo que ayudará a crear conversaciones organizadas en torno a las preguntas presentadas. Asimismo, estos hilos podrán ser elegidos por los profesores responsables de las asignaturas para que sean mostrados a estudiantes de otros cursos en una sección en la web de la asignatura de "Preguntas destacadas".

Implícitamente, este Trabajo de Fin de Grado también tiene el objetivo de aplicar los conocimientos y aptitudes adquiridos a lo largo del Grado en Ingeniería en Tecnologías de Telecomunicación y aprender nuevas tecnologías que son ampliamente usadas por las empresas de todo el mundo para desarrollar aplicaciones web.

Si bien, este desarrollo se podría considerar como una nueva plataforma académica, la idea de este trabajo de fin de grado es crear una aplicación web que haga similitud a las plataformas de los centros académicos con la novedad de la incorporación de la herramienta de chat interno desarrollada.

1.2.1. OBJETIVOS DE LA APLICACIÓN

En este apartado detallaremos los objetivos que pretende cubrir la aplicación web que se va a desarrollar en esta memoria:

- Desarrollar una aplicación web similar a las usadas en las instituciones académicas.
- Desarrollar un chat interno que mejore la comunicación entre los estudiantes y profesores en los centros académicos.
- Implementar la creación de hilos para los mensajes del chat que ayude a crear conversaciones ordenadas.
- Poder tener una conversación a tiempo real en el chat, lo que conseguiría una mayor interacción entre los estudiantes y profesores.

- Permitir a los responsables de las asignaturas guardar hilos del chat para que sean mostrados en la web de las asignaturas en una sección de preguntas destacadas.

- Crear un servicio API para que pueda ser utilizado en un dispositivo móvil en forma de aplicación o mediante una aplicación web en el navegador.

- Desarrollar una aplicación web responsive para cualquier tipo de pantalla, ya sea móvil, tablet u ordenador.

1.2.2. OBJETIVOS PERSONALES

Como objetivos personales para este proyecto se proponen los siguientes:

- Aplicar los conocimientos adquiridos a lo largo del grado de Ingeniería en Tecnologías de Telecomunicación.

- Aprender y conocer el uso y funcionamiento del desarrollo de aplicaciones web y sus ventajas e inconvenientes frente a otros tipos de desarrollos.

- Adquirir experiencia en el manejo de la plataforma cloud de Amazon AWS.

- Adquirir experiencia en el uso de software de control de versiones (Git).

- Trabajar y administrar con comandos básicos y avanzados desde la terminal un servidor Linux.

- Aprender y utilizar la infraestructura LAMP (Linux, Apache Web Server, MySQL/ MariaDB y PHP) para desarrollar aplicaciones web.

- Adquirir experiencia en los lenguajes de programación PHP y JavaScript.

- Aprender a manejar el framework de Laravel para la creación de aplicaciones y servicios web con PHP.

- Adquirir experiencia en el diseño y creación de bases de datos relacionales.

- Comprender y utilizar la librería de React para la creación de interfaces de usuario sencillas, interactivas y con componentes reutilizables.

- Aprender a utilizar el framework de CSS de TailwindCSS para el diseño de páginas web.

- Comprender los conceptos básicos de websockets y su funcionamiento.

1.3. ESTRUCTURA DEL DOCUMENTO

En este punto se explicará brevemente cada uno de los capítulos de este documento:

1. **Introducción**. Se trata el capítulo introductorio y se pretende presentar el contexto y objetivos por los que se ha decidido a desarrollar este proyecto, además hay una pequeña mención al desarrollo y se expone la estructura que se llevara a cabo en este documento.

2. **Estado del arte**. En este capítulo realizaremos una investigación sobre aplicaciones similares a la desarrollada en este proyecto, así como un análisis a las tecnologías utilizadas para la creación y desarrollo de aplicaciones web.

3. **Análisis de requisitos**. En este apartado definiremos el proyecto, los requisitos necesarios para su desarrollo e implementación y los casos de uso para los actores del sistema.

4. **Diseño de la aplicación**. En este capítulo se realiza una descripción detallada de la solución implementada y se especifican las tecnologías usadas para su desarrollo.

5. **Implementación**. En este apartado se expondrá cómo se ha desarrollado la aplicación tanto para el lado de servidor como para el lado del cliente con ejemplos de código para los componentes más importantes.

6. **Resultados**. En este capítulo expondremos el resultado final de la aplicación.

7. **Conclusiones**. En este apartado se presentan las conclusiones sobre el trabajo realizado, así como una serie de posibles mejoras para el proyecto.

8. **Bibliografía**. En este apartado se enumeran las diferentes fuentes utilizadas para la elaboración de esta memoria.

1.4. MENCIÓN AL DESARROLLO

Cabe mencionar que en este Trabajo de Fin de Grado el Backend se ha realizado conjuntamente con un compañero en el Grado en Ingeniería en Tecnologías de Telecomunicación, que ha desarrollado el TFG “App full stack para plataforma académica con Frontend móvil”. En su proyecto se utilizará el Backend para implementar una aplicación similar con el formato de aplicación móvil.

2. ESTADO DEL ARTE

En este capítulo haremos un repaso sobre las plataformas de los centros educativos en la actualidad y, posteriormente, una comparación de algunas aplicaciones similares que existen en el mercado que cumplan con el objetivo de este proyecto. Para finalizar en este capítulo se realizará una comparación general de las tecnologías más populares que existen en el mercado y que podrían utilizarse para el desarrollo de este proyecto.

2.1. ACTUALIDAD

Hoy en día, en la mayoría de las plataformas de los centros educativos no se integran herramientas propias para facilitar la comunicación entre estudiantes y profesores, provocando que sea necesario hacer uso de aplicaciones y plataformas externas, que en algunos casos tienen costo para el centro docente y crean una dependencia con estas. A continuación, se expondrán algunos ejemplos de aplicaciones usadas por los centros educativos para cumplir con el objetivo de mejorar la comunicación entre docentes y estudiantes.

2.2. APLICACIONES SIMILARES

Como se ha comentado en el capítulo de introducción, la transformación tecnológica y el aceleramiento a la búsqueda de soluciones para la educación a distancia en la pandemia han causado la creación de muchas aplicaciones y plataformas para la comunicación de los estudiantes y profesores de los centros educativos en estos últimos años. A continuación, se expondrán algunos ejemplos de las plataformas y aplicaciones más usadas para la comunicación en el sector de la educación.

Empezaremos por una de las plataformas más populares, **Microsoft Teams**, la plataforma educativa gratuita para instituciones educativas creada por el gigante Microsoft.

Ilustración 1. Aplicaciones similares: Microsoft Teams

Microsoft Teams es una plataforma unificada de comunicación y colaboración, que facilita la interacción entre colaboradores, conocido especialmente por ser un centro de trabajo basado en chats. Teams para el ámbito educativo reúne todo lo que se necesitas en clase y en la escuela. Videollamadas, chats, pantalla compartida, modo Juntos, Privacidad y Seguridad, uso compartido de archivos, aplicaciones y flujos de trabajo, y una infinidad de plugins desarrollados para integrarse con ella, lo que convierte a Microsoft Teams en una de las mejores opciones. [1] [2]

Otra de las plataformas más usadas en el ámbito educativo es **Google Classroom**. Esta herramienta fue creada por Google en 2014, y permite gestionar un aula de forma colaborativa a través de Internet.

Ilustración 2. Aplicaciones similares: Google Classroom

Google Classroom permite la asignación de tareas de forma selectiva, permite compartir documentos con todas las clases, y facilita la organización de la información al generar estructuras automáticas de carpetas para organizar los recursos.

Gracias a esta aplicación, profesores y alumnos pueden mantenerse en contacto fácilmente tanto dentro como fuera del centro. Classroom permite ahorrar tiempo y papel, así como crear clases, distribuir tareas, comunicarse con otros usuarios y mantener el trabajo organizado de manera sencilla. [3]

Otra aplicación de comunicación para el ámbito educativo es **Remind**, creada por una empresa con sede en San Francisco, se trata de una herramienta de comunicación fácil de usar, segura y gratuita con la que los profesores pueden contactar al instante con estudiantes.

Ilustración 3. Aplicaciones similares: Remind

En concreto, la aplicación permite crear grupos por clase o materia y así enviar mensajes a los alumnos: recordatorios, tareas, propuestas de lecturas o de debates, etc. También admite el envío de archivos, documentos o fotografías, muy útil en el caso de excursiones, actividades extraescolares o para fotografiar la pizarra. [4]

Otra herramienta que está en auge para la comunicación en el ámbito educativo es **Slack**, se trata de una herramienta de mensajería lanzada en el 2014, que se ha convertido en una plataforma favoritas con las que trabajan las empresas y que está empezando a utilizarse en el ámbito educativo.

Ilustración 4. Aplicaciones similares: Slack

Slack fue diseñado en un inicio para mejorar la efectividad de la comunicación entre compañeros y el trabajo en equipo en empresas, sus múltiples aplicaciones en la elaboración de proyectos y ejecución de tareas la ha situado como una herramienta muy potente de comunicación que facilita la colaboración para todas aquellas personas que sean miembros de un equipo de trabajo.

En el ámbito educacional, utilizar Slack puede ayudarte como docente a crear una comunidad lo que permitirá que todos los estudiantes hagan sus consultas o aportaciones de forma pública, evitando repetir información y a consolidar lineamientos en tareas o proyectos. [5]

Como última aplicación tenemos **Moodle**, se trata de una plataforma de aprendizaje diseñada para proporcionarle a educadores, administradores y estudiantes un sistema integrado único, robusto y seguro para crear ambientes de aprendizaje personalizados.

Ilustración 5. Aplicaciones similares: Moodle

Moodle proporciona un conjunto de herramientas flexible para soportar la educación semipresencial o totalmente remota. Moodle es una herramienta altamente configurable donde se pueden habilitar o deshabilitar características, integrando herramientas colaborativas externas tales como foros, wikis, chats y blogs. [6]

2.3. TECNOLOGÍAS ACTUALES

En esta sección se explicarán las tecnologías utilizadas por una aplicación web y se hará una comparativa general de las tecnologías que existen actualmente en el mercado y que podrían utilizarse para la realización del proyecto.

2.3.1. SERVIDOR DE APLICACIONES WEB

Un servidor web es un software que se utiliza para servir archivos a sitios web en Internet. El servidor web es responsable de garantizar que la comunicación entre el servidor y el cliente sea segura y sin fallos. Cuando un usuario realiza

una petición, el servidor web coge los archivos del servidor físico y los entrega al usuario. De este modo, los servidores web tienen que servir diferentes páginas a diferentes usuarios al mismo tiempo. [7]

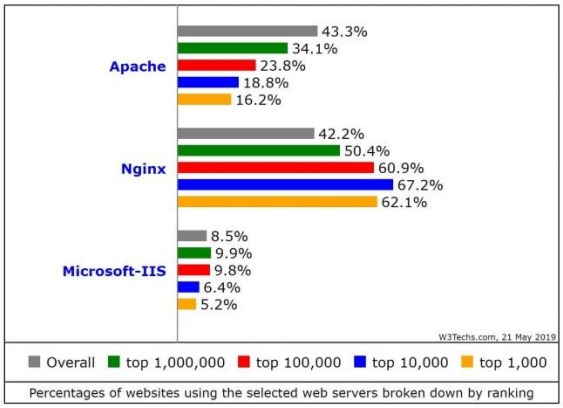


Ilustración 6. Ranking servidores web

Como se puede observar, la figura anterior muestra el porcentaje de uso de los servidores web más importantes en internet, concluyendo que los preferidos por la comunidad de desarrolladores son Apache y Nginx con un 85.5% de la cuota de mercado entre ambos, por ello nos centraremos solo en estos dos servidores web.

2.3.1.1. NGINX

Nginx es un servidor web de código abierto y de alto rendimiento creado en 2004, que se presenta como la solución ideal para gestionar sitios web de alto tráfico. Muchos sitios de alto tráfico de usuarios como Netflix o Pinterest utilizan el servidor web Nginx. [7]

Sus características más importantes son:

- Gratuito y de código abierto.
- Arquitectura asíncrona basada en eventos.
- Alto rendimiento.
- Bajo consumo de recursos.
- Compatibilidad IPv6.

Ilustración 7. Logo NGINX

2.3.1.2. APACHE

El servidor web Apache, también llamado Apache HTTP Server, se trata de un servidor web gratuito y de código abierto que ofrece muchos módulos de seguridad, autenticación, caching, reescritura de URLs, etc. [7]

Entre sus características destacan:

- Gratuito y de código abierto.
- Arquitectura basada en módulos.
- Fácil configuración y personalización.
- Actualizaciones regulares.
- Gran comunidad de desarrollo.
- Compatibilidad con IPv6.

Ilustración 8. Logo de Apache

2.3.2. BASES DE DATOS

La mayoría de las aplicaciones web de la actualidad requieren del uso de una base de datos que almacene grandes cantidades de información de una forma organizada y accesible para su futuro uso.

Las bases de datos se pueden clasificar en dos grandes grupos según la forma en la que administran los datos: relacionales (SQL) y no relacionales (NoSQL).

2.3.2.1. BASE DE DATOS RELACIONALES

Una base de datos relacional es un tipo de base de datos que almacena y proporciona acceso a puntos de datos relacionados entre sí. Las bases de datos relacionales se basan en el modelo relacional, una forma intuitiva y directa de representar datos en tablas. En una base de datos relacional, cada fila de la tabla es un registro con un ID único llamado clave. Las columnas de la tabla contienen atributos de los datos, y cada registro generalmente tiene un valor para cada atributo, lo que facilita el establecimiento de las relaciones entre los puntos de datos. [8]

En los siguientes apartados se exponen los gestores de bases de datos (SGBD) más usados para las bases de datos relacionales:

MySQL

Es un sistema gestor de base de datos multihilo y multiusuario utilizado en la gran parte de las páginas web actuales. Además es el más usado en aplicaciones creadas como software libre.
Sus principales características son su fácil instalación, configuración y uso acompañados de un gran rendimiento, soporta SSL y multiplataforma. Sin embargo, no trabaja de forma eficiente con bases de datos que superan un determinado tamaño. [9]

Ilustración 9. Logo MySQL

MariaDB

Es un sistema gestor de base de datos derivado de MySQL que cuenta con todas las características de MySQL más algunas extensiones.
Sus principales características son su gran escalabilidad, seguridad, rapidez en transiciones y la posibilidad de añadir extensiones de la comunidad. [9]

Ilustración 10. Logo MariaDB

SQLite

SQLite se define como una biblioteca escrita en “C” que implementa un sistema gestor de base de datos que permite transacciones sin necesidad de un servidor ni configuraciones.
Las principales características de este SGBD son su reducido tamaño, gran estabilidad y portabilidad y rendimiento. Por el contrario, no soporta bases de

datos muy grandes lo que provoca una clara desventaja frente a las demás en la escalabilidad. [9]

Ilustración 11. Logo SQLite

PostgreSQL

Este sistema gestor de base de datos relacional está orientado a objetos y es libre, publicado bajo la licencia BSD. Como principales características destacan el control de concurrencia, flexibilidad a la hora de elegir lenguajes de programación, disponer de una herramienta fácil e intuitiva para la administración de base de datos, así como otorgar robustez, eficiencia y estabilidad. Multiplataforma. La principal desventaja de PostgreSQL es que es muy lenta para administrar bases de datos pequeñas ya que está optimizada para gestionar grandes volúmenes de datos. [9]

Ilustración 12. PostgreSQL

Oracle

Oracle se define como sistema de gestión de base de datos por excelencia en el mundo empresarial, considerado siempre por ser el más completo y robusto. Las principales características de Oracle son su gran estabilidad, escalabilidad y que es un gestor multiplataforma. La principal desventaja de este SGBD es el coste el software. [9]

Ilustración 13. Logo Oracle

Microsoft SQL Server

Es un sistema de gestión de base de datos relacional propiedad de Microsoft basado en el lenguaje “Transact-SQL” capaz de suministrar grandes cantidades de datos de manera simultánea. Las principales características de este SGBD son Escalabilidad, estabilidad y seguridad además de contar con soporte exclusivo de Microsoft, también tiene la posibilidad de cancelar consultas y poseer un potente entorno gráfico de administración. Su principal desventaja al igual que Oracle es el precio del software. [9]

Ilustración 14. Logo Microsoft SQL Server

2.3.2.2. BASE DE DATOS NO RELACIONALES

Las bases de datos NoSQL están diseñadas específicamente para modelos de datos concretos y tienen esquemas flexibles para crear aplicaciones modernas. Las bases de datos NoSQL son ampliamente reconocidas porque son fáciles de desarrollar, por su funcionalidad y el rendimiento a escala. [10]

En los siguientes apartados se exponen los gestores de bases de datos más usados para las bases de datos no relacionales:

MongoDB

MongoDB es el sistema gestor de base de datos no relacionales más popular y utilizado por los desarrolladores, está orientado a ficheros que almacenan la información en estructuras BSON (Binary JSON).
Sus principales características son su fácil escalabilidad horizontal y balanceo de carga, además de contar con un sistema de indexación que acelera las búsquedas y permite el almacenamiento de ficheros. Por el contrario, como principal desventaja es que no es adecuado para la realización de transacciones complejas. [9]

Ilustración 15. Logo MongoDB

Apache Cassandra

Es un gestor de base de datos NoSQL distribuido y masivamente escalable, que está basado en el almacenamiento clave-valor, usado por empresas como Facebook, Twitter o Netflix.
Sus principales características es que es un gestor de base de datos distribuido, multiplataforma, con un lenguaje propio denominado CQL (Cassandra Query Languaje), con escalabilidad lineal y horizontal. [9]

Ilustración 16. Logo Apache Cassandra

Redis

Es un gestor de base de datos NoSQL para el almacenamiento en memoria caché y la administración de sesiones, al igual que Apache Cassandra está basado en el almacenamiento clave-valor.
Sus principales características son su atomicidad y persistencia, gran velocidad, simplicidad y es Multiplataforma. [9]

Ilustración 17. Logo Redis

2.3.3. TECNOLOGÍAS DEL LADO DEL SERVIDOR

Dado que existen numerosas tecnologías del lado del servidor analizaremos los principales marcos de trabajo en el backend y expondremos sus características más destacadas.

2.3.3.1. LARAVEL

Laravel es un framework de PHP de código abierto basado en la arquitectura Modelo-Vista-Controlador (MVC) con una sintaxis sencilla y elegante, que proporciona gran facilidad a la hora de desarrollar proyectos de una forma rápida y efectiva. [11]

Alguna de las características más destacadas de laravel son:

- Motor de enrutamiento.
- Motor de plantillas ligeras e integradas (Blade).
- Inyección de dependencia.
- Múltiples motores de sesión y caché.
- ORM de base de datos (Eloquent).
- Migración de esquema de base de datos.
- Gran comunidad de desarrolladores.

Ilustración 18. Logo Laravel

2.3.3.2. DJANGO

Django es un marco web de Python de alto nivel que permite el desarrollo de sitios web de una manera rápida, segura y altamente mantenibles respetando el patrón de diseño conocido como Modelo-Vista-Controlador (MVC). [11]

Sus características más significativas son:

- Muy rápido para el desarrollo.
- Seguridad.
- Escalabilidad.

Ilustración 19. Logo Django

2.3.3.3. RUBY ON RAILS

Ruby on Rails es un framework web caracterizado por la simplicidad, usa el lenguaje de programación Ruby y se utiliza para construir aplicaciones web del lado del servidor (backend), además trabaja con el modelo MVC (modelo vista controlador), un patrón de arquitectura bastante famoso. [11]

Las características más importantes son:

- Sintaxis limpia y código sencillo.
- ORM de base de datos (Active Record).
- Gran comunidad de desarrolladores.
- Escalabilidad.

Ilustración 20. Logo Rails

2.3.3.4. EXPRESS

Express es un framework rápido, minimalista y flexible escrito en JavaScript de Node.js que permite crear APIs y aplicaciones web fácilmente. [11]

Las características más importantes de Express son:

- Alta escalabilidad.
- Rendimiento y velocidad.
- Permite múltiples motores de plantillas.
- Fácil configuración.
- Gran cantidad de extensiones y bibliotecas.
- Gran comunidad de desarrolladores.

Ilustración 21. Logo Express

2.3.4. TECNOLOGÍAS DEL LADO DEL CLIENTE

En esta sección se expondrán las principales tecnologías usadas en el lado del cliente y algunos de los frameworks más usados por estas.

2.3.4.1. HTML

HTML (HyperText Markup Language) es el lenguaje de marcado estándar que se usa para crear páginas y aplicaciones web. Sus elementos forman los bloques de creación de las páginas y representan texto con formato, imágenes, entradas de formulario y otras estructuras. Cuando un explorador realiza una solicitud a una dirección URL, con independencia de que se obtenga una página o una aplicación, lo primero que se devuelve es un documento HTML. Este documento HTML puede hacer referencia o incluir información adicional sobre su apariencia y diseño en forma de CSS, o el comportamiento en forma de JavaScript. [12]

Ilustración 22. Logo HTML

2.3.4.2. CSS

CSS (Hoja de estilos en cascada) se usa para controlar la apariencia y el diseño de los elementos HTML. Los estilos CSS se pueden aplicar directamente a un elemento HTML, o bien definirse por separado en la misma página o en un archivo independiente al que la página haga referencia. [12]

Ilustración 23. Logo CSS

Actualmente se han desarrollada frameworks de trabajo para CSS, para ayudar a los desarrolladores a estilizar sus aplicaciones de una manera más rápida, fluida y sencilla.

TailwindCSS es un framework de CSS de código abierto para el diseño de páginas web. La principal característica de esta biblioteca es que, a diferencia de otras como Bootstrap, no genera una serie de clases predefinidas para elementos como botones o tablas. En su lugar, crea una lista de clases CSS "de utilidad" que se pueden usar para dar estilos individuales a cada elemento. [13]

Ilustración 24. Logo Tailwind CSS

Bootstrap es un framework CSS de código abierto utilizado para el diseño rápido de aplicaciones web de una manera responsive que contiene plantillas de diseño basadas en HTML y CSS, es decir nos ofrece componentes HTML estilizados para cualquier tamaño de pantalla. [14]

Ilustración 25. Logo Bootstrap

Foundation es un framework CSS de código abierto responsivo que cuenta con una gran variedad de componentes HTML y CSS para el diseño de interfaces de usuario, diferentes fragmentos de código y plantillas, así como extensiones opcionales de JavaScript. [15]

Ilustración 26. Logo Foundation

2.3.4.3. JAVASCRIPT

JavaScript es un lenguaje de programación interpretado y dinámico que se ha estandarizado en la especificación del lenguaje ECMAScript. Es el lenguaje de programación de la web que nos sirve para otorgar dinamismo a las páginas.[12]

Ilustración 27. Logo JavaScript

En la actualidad, debido a la necesidad de hacer páginas web más rápidas, responsivas y dinámicas, se han ido creando numerosos frameworks y librerías para facilitar el desarrollo con JavaScript, donde las más utilizadas por la comunidad de desarrolladores son:

AngularJS

Angular es un framework de JavaScript de código abierto desarrollado por Google para facilitar el desarrollo de aplicaciones web de una sola página (SPA) a través del lenguaje de programación Typescript. [16]

Entre las principales ventajas que ofrece Angular destacan:

- Arquitectura modelo-vista-controlador (MVC).
- Modular.
- Alta escalabilidad.
- Inyección de dependencias.
- Reutilización de código.
- Permite la creación de nuevas etiquetas HTML.

Ilustración 28. Logo Angular

ReactJS

ReactJS es una librería de JavaScript declarativa, eficiente y flexible de código abierto que permite el desarrollo de interfaces de usuario complejas mediante pequeñas y aisladas piezas de código llamadas “componentes”. Es mantenida por Facebook y la comunidad de software libre. [17]

Las principales ventajas que ofrece ReactJS son las siguientes:

- Alto rendimiento y escalabilidad.
- Compatible con SEO.
- Multiplataforma (React Native).
- Fácil mantenimiento.
- Diseño modular.
- Gran comunidad de desarrolladores.
- Virtual DOM.

Ilustración 29. Logo React

VueJS

VueJS es un framework progresivo para construir interfaces de usuario. A diferencia de otros frameworks monolíticos, VueJS está diseñado desde cero para ser utilizado incrementalmente. [18]

Las características principales de VueJS son:

- Curva de aprendizaje sencilla.
- Framework progresivo.
- Modular.
- Virtual DOM.
- Comunidad de desarrolladores en crecimiento.

Ilustración 30. Logo Vue

3. ANÁLISIS DE LOS REQUISITOS

En este capítulo definiremos el proyecto y los requisitos necesarios para su implementación, así como los casos de uso.

3.1. DEFINICIÓN DEL PROYECTO

El objetivo principal de este Trabajo de Fin de Grado es la creación y desarrollo de un prototipo de aplicación web que preste un servicio de comunicación en tiempo real, rápido, fácil y seguro a los estudiantes y profesores de un mismo centro educativo mediante un sistema de chat interno, en el que los estudiantes puedan plantear sus dudas en los grupos de chat creados para cada una de las asignaturas y éstas puedan ser resueltas por sus compañeros o profesores. Además, el chat será capaz de crear hilos de respuestas para las preguntas que se propongan en los grupos de chat, lo que ayudará a crear conversaciones organizadas en torno a las preguntas presentadas. Asimismo, estos hilos podrán ser elegidos por los profesores responsables de las asignaturas para que sean mostrados a estudiantes de otros cursos en una sección en la web de la asignatura de “Preguntas destacadas”.

3.2. REQUISITOS FUNCIONALES

Los requisitos funcionales para cumplir con los objetivos de nuestro proyecto son los siguientes:

RF -1	**Acceso restringido**
Descripción	La aplicación sólo permitirá el acceso a usuarios registrados en la aplicación mediante un formulario de inicio de sesión.

Tabla 1. Descripción Requisito funcional 1: Acceso restringido

RF -2	**Restablecer contraseña**
Descripción	La aplicación permitirá el restablecimiento de contraseña a los usuarios sin autentificar.

Tabla 2. Descripción Requisito funcional 2: Restablecer contraseña

RF -3	**Cerrar sesión**
Descripción	El sistema permitirá cerrar sesión a los usuarios que estén autentificados.

Tabla 3. Descripción Requisito funcional 3: Cerrar sesión

RF -4	Roles de usuario
Descripción	La aplicación dispondrá de tres tipos de usuarios o roles: administradores, estudiantes y profesores.

Tabla 4. Descripción Requisito funcional 4: Roles de usuario

RF -5	Distintos niveles de autorización
Descripción	El sistema debe restringir las acciones en función del nivel de autorización del usuario o rol.

Tabla 5. Descripción Requisito funcional 5: Distintos niveles de autorización

RF -6	Gestión de usuarios
Descripción	El sistema dispone de un módulo de gestión de usuarios el cual solamente será accesible para los usuarios administradores.

Tabla 6. Descripción Requisito funcional 6: Gestión de usuarios

RF -7	Gestión de grados
Descripción	El sistema dispone de un módulo de gestión de grados el cual solamente será accesible para los usuarios administradores.

Tabla 7. Descripción Requisito funcional 7: Gestión de grados

RF -8	Gestión de asignaturas
Descripción	El sistema dispone de un módulo de gestión de asignaturas el cual solamente será accesible para los usuarios administradores.

Tabla 8. Descripción Requisito funcional 8: Gestión de asignaturas

RF - 9	Gestión de matrículas
Descripción	El sistema dispone de un módulo de gestión de matrículas el cual solamente será accesible para los usuarios administradores.

Tabla 9. Descripción Requisito funcional 9: Gestión de matrículas

RF – 10	Editar datos de usuario
Descripción	La aplicación permitirá a los usuarios autentificados cambiar sus datos y subir imágenes de perfil.

Tabla 10. Descripción Requisito funcional 10: Editar datos de usuario

RF – 11	Sistema de comunicación interno
Descripción	La aplicación tendrá un sistema de comunicación interno mediante un chat.

Tabla 11. Descripción Requisito funcional 11: Sistema de comunicación interno

RF – 12	Crear nuevos chats
Descripción	La aplicación permitirá crear chats privados, grupales a todos los usuarios y chats de las asignaturas a los usuarios profesores.

Tabla 12. Descripción Requisito funcional 12: Crear nuevos chats

RF - 13	Sistema de chat a tiempo real
Descripción	El sistema debe ser capaz de enviar y recibir mensajes en tiempo real y mostrarlos en el chat.

Tabla 13. Descripción Requisito funcional 13: Sistema de chat a tiempo real

RF – 14	Subir archivos multimedia
Descripción	El sistema de chat permitirá subir cualquier tipo de archivo ya sea audio, video, imágenes o ficheros de otros formatos.

Tabla 14. Descripción Requisito funcional 14: Subir archivos multimedia

RF - 15	Responder mensajes del chat
Descripción	El sistema de chat permitirá responder mensajes seleccionados en los chats.

Tabla 15. Descripción Requisito funcional 15: Responder mensajes del chat

RF - 16	Creación de hilos de mensajes
Descripción	El sistema de chat creará hilos con las respuestas a un determinado mensaje.

Tabla 16. Descripción Requisito funcional 16: Creación de hilos de mensajes

RF - 17	Guardar hilos de mensajes
Descripción	Los usuarios profesores podrán seleccionar hilos de mensajes de los chats para poder añadirlos a la web de la asignatura en un apartado llamado “preguntas destacadas” para que éstos sean visibles para estudiantes de otros cursos.

Tabla 17. Descripción Requisito funcional 17: Guardar hilos de mensajes

RF - 18	Editar información chat
Descripción	El sistema de chat permitirá editar el nombre y avatar de los chats grupales.

Tabla 18. Descripción Requisito funcional 18: Editar información del chat

RF - 19	Otorgar permisos en los chats
Descripción	El sistema de chat permitirá dar permisos a los integrantes de un chat grupal seleccionados.

Tabla 19. Descripción Requisito funcional 19: Otorgar permisos en los chats

RF - 20	Ver tus asignaturas
Descripción	La aplicación permitirá ver a los profesores y usuarios las asignaturas que están cursando o que han cursado en años anteriores.

Tabla 20. Descripción Requisito funcional 20: Ver tus asignaturas

3.3. REQUISITOS NO FUNCIONALES

Los requisitos no funcionales para cumplir con los objetivos de nuestro proyecto son los siguientes:

RNF - 1	Usabilidad
Descripción	La aplicación debe ser de uso sencillo.

Tabla 21. Descripción Requisito no funcional 1: Usabilidad

RNF - 2	Interfaz
Descripción	La aplicación debe tener una interfaz atractiva y fácil de usar para cualquier usuario con unos conocimientos mínimos de internet.

Tabla 22. Descripción Requisito no funcional 2: Interfaz

RNF - 3	Integridad
Descripción	La información proporcionada y usada ha de ser cuidada para protegerse contra la corrupción y estados inconsistentes.

Tabla 23. Descripción Requisito no funcional 3: Integridad

RNF - 4	Rendimiento
Descripción	Los tiempos de respuesta de la aplicación deben ser los mínimos posibles.

Tabla 24. Descripción Requisito no funcional 4: Rendimiento

RNF - 5	Portabilidad
Descripción	La aplicación puede ser usada por distintos navegadores y dispositivos con diferente tamaño de ventana.

Tabla 25. Descripción Requisito no funcional 5: Portabilidad

RNF - 6	Confidencialidad
Descripción	El acceso a la aplicación estará restringido sólo para los usuarios registrados en el sistema.

Tabla 26. Descripción Requisito no funcional 6: Confidencialidad

RNF - 7	Disponibilidad
Descripción	Se garantiza el acceso en todo momento a los usuarios del sistema.

Tabla 27. Descripción Requisito no funcional 7: Disponibilidad

3.4. CASOS DE USO

Los casos de uso son una descripción de la secuencia de acciones a realizar por los usuarios también llamado actores para realizar una determinada tarea en el sistema.

3.4.1. ACTORES

En esta sección se describen los actores implicados en el sistema:

Actor	Usuario anónimo
Descripción	Usuario que aún no ha sido autentificado en el sistema.

Tabla 28. Descripción Actor 1: Usuario anónimo

Actor	Administrador
Descripción	Usuario autentificado con el rol de administrador, se encarga de la gestión general de la aplicación.

Tabla 29. Descripción Actor 2: Administrador

Actor	Profesor
Descripción	Usuario autentificado con el rol de profesor.

Tabla 30. Descripción Actor 3: Profesor

Actor	Estudiante
Descripción	Usuario autentificado con el rol de estudiante.

Tabla 31. Descripción Actor 4: Estudiante

3.4.2. DIAGRAMAS DE CASO DE USO

En esta sección se muestran los diagramas de casos de uso para cada uno de los actores de la aplicación.

En la figura siguiente, encontramos el diagrama de casos de uso posibles para un usuario anónimo, en ella, podemos observar que tiene dos casos posibles: iniciar sesión y recuperar contraseña.

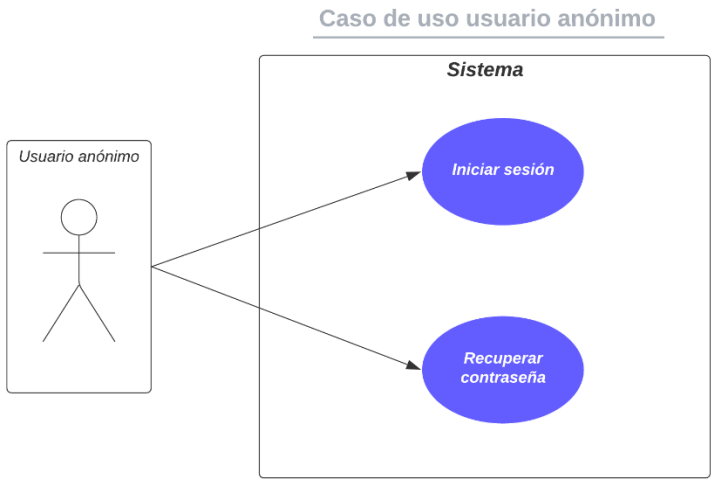


Ilustración 31. Diagrama Casos de uso usuario Anónimo

En el segundo diagrama que se muestra a continuación aparecen los casos de uso para el usuario administrador que tiene acceso a todas las funcionalidades para gestionar la aplicación.

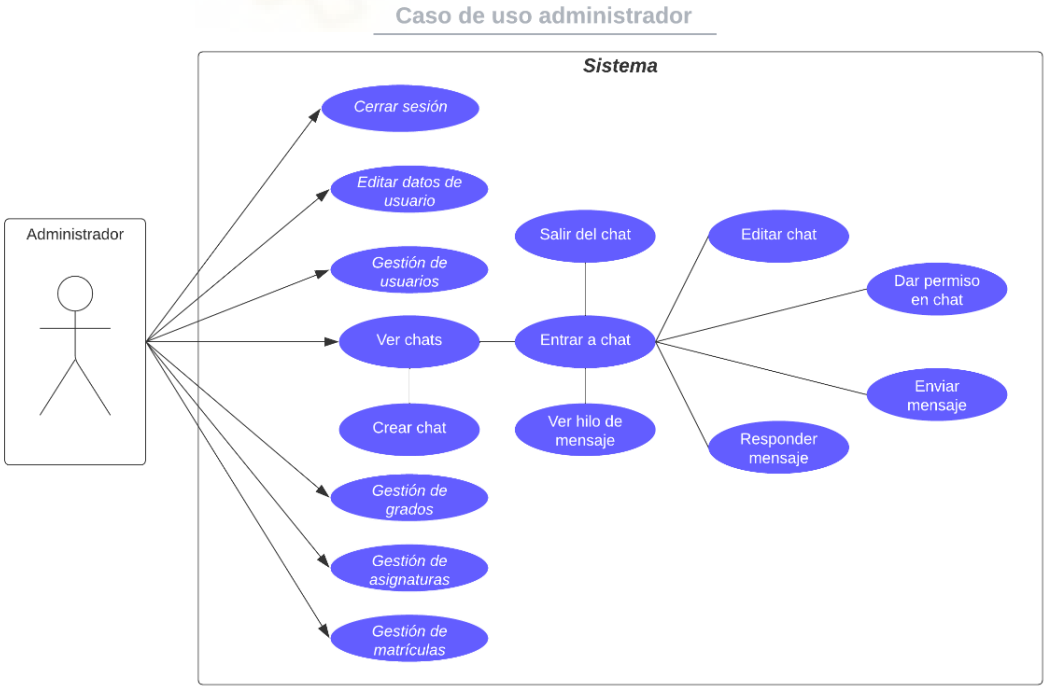


Ilustración 32. Diagrama Casos de uso usuario Administrador

En el diagrama de la figura siguiente se muestran todos los casos de uso para el usuario profesor.

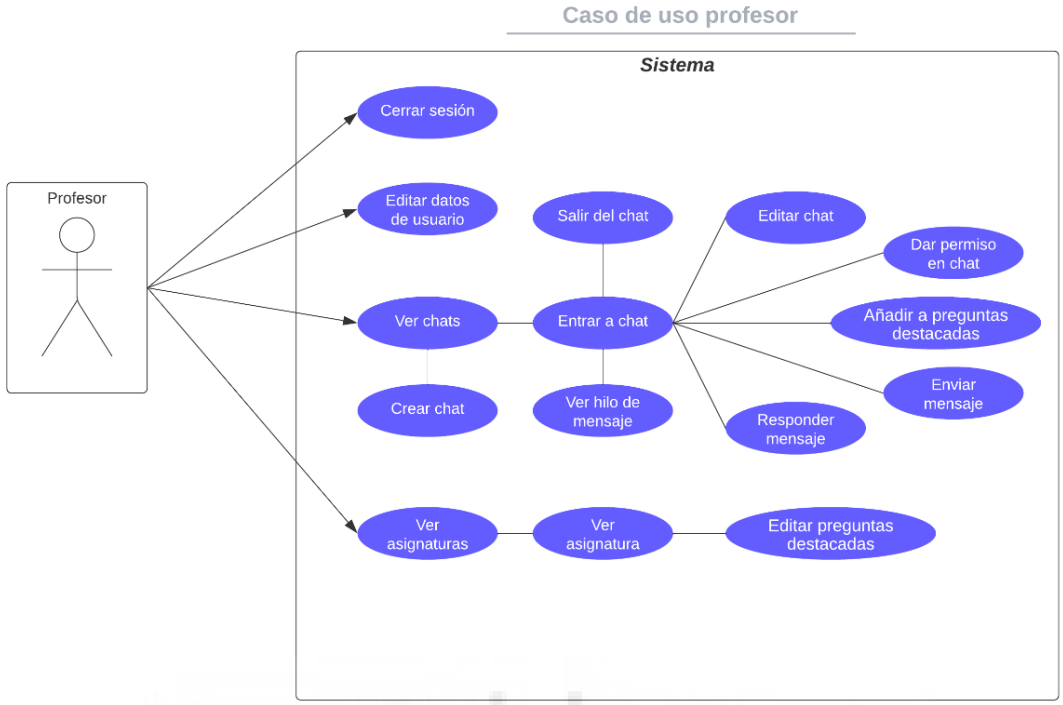


Ilustración 33. Diagrama Casos de uso usuario Profesor

Por último, en la ilustración 34 se muestran los casos de uso para el usuario Estudiante.

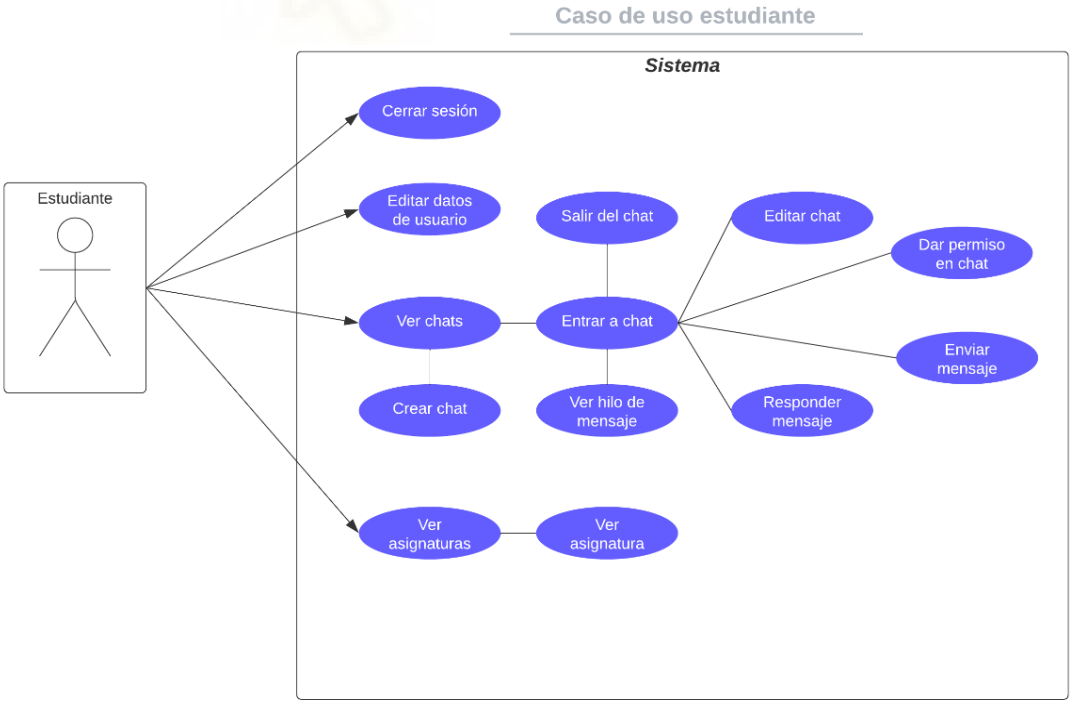


Ilustración 34. Diagrama Casos de uso usuario Estudiante

3.4.3. DESCRIPCIÓN DE LOS CASOS DE USO

CU-1	Iniciar sesión
Descripción	Los usuarios se autentifican en el sistema median un formulario de inicio de sesión con los campos de email y contraseña.
Actores	Usuario administrador, profesor y estudiante.
Dependencias	
Precondición	El usuario debe estar sin autentificar y registrado en el sistema.
Secuencia normal	1. Acceder a la pantalla de inicio de sesión. 2. Rellenar el formulario de inicio de sesión.
Postcondición	El usuario es autenticado en el sistema.
Excepciones	▪ Datos del formulario incorrectos. (Muestra mensaje de alerta)
Importancia	Alta
Estado	Completo

Tabla 32. . Descripción Caso de uso 1: Iniciar sesión

CU-2	Cierre de sesión
Descripción	Permite a un usuario autentificado cerrar la sesión en el sistema.
Actores	Usuario administrador, profesor y estudiante.
Dependencias	
Precondición	El usuario debe estar autentificado en el sistema.
Secuencia normal	1. Pulsar botón de cerrar sesión. 2. Usuario autentificado pasa a usuario anónimo.
Postcondición	
Excepciones	▪ Datos del formulario incorrectos. (Muestra mensaje de alerta)
Importancia	Alta
Estado	Completo

Tabla 33. Descripción Caso de uso 2: Cierre de sesión

CU-3	Editar datos de usuario
Descripción	Permite al usuario cambiar sus datos en el sistema, así como añadir un avatar.
Actores	Usuario administrador, profesor y estudiante.
Dependencias	
Precondición	El usuario debe estar autentificado en el sistema.
Secuencia normal	1. Ir a la pantalla de Ajustes de cuenta. 2. Rellenar el formulario con los datos a modificar. 3. Enviar formulario.
Postcondición	El usuario podrá editar sus datos del sistema.
Excepciones	▪ Datos del formulario incorrectos. (Muestra mensaje de alerta)
Importancia	Media
Estado	Completo

Tabla 34. Descripción Caso de uso 3: Editar datos de usuario

CU-4	Gestión de usuarios
Descripción	Permite listar, modificar, crear y eliminar usuarios del sistema.
Actores	Usuario administrador.
Dependencias	
Precondición	El usuario debe estar autentificado en el sistema y ser administrador.
Secuencia normal	1. Ir a las pantallas de gestión de usuarios. 2. Ingresar datos necesarios para la acción seleccionada.
Postcondición	El sistema se configura de acuerdo con la acción seleccionada.
Excepciones	▪ Datos del formulario incorrectos. (Muestra mensaje de alerta)
Importancia	Baja
Estado	Completo

Tabla 35. Descripción Caso de uso 4: Gestión de usuarios

CU-5	Gestión de grado
Descripción	Permite listar, modificar, crear y eliminar grados del sistema.
Actores	Usuario administrador.
Dependencias	
Precondición	El usuario debe estar autentificado en el sistema y ser administrador.
Secuencia normal	1. Ir a las pantallas de gestión de grados. 2. Ingresar datos necesarios para la acción seleccionada.
Postcondición	El sistema se configura de acuerdo con la acción seleccionada.
Excepciones	▪ Datos del formulario incorrectos. (Muestra mensaje de alerta)
Importancia	Baja
Estado	Completo

Tabla 36. Descripción Caso de uso 5: Gestión de grado

CU-6	Gestión de asignaturas
Descripción	Permite listar, modificar, crear y eliminar asignaturas del sistema.
Actores	Usuario administrador.
Dependencias	
Precondición	El usuario debe estar autentificado en el sistema y ser administrador.
Secuencia normal	1. Ir a las pantallas de gestión de asignaturas. 2. Ingresar datos necesarios para la acción seleccionada.
Postcondición	El sistema se configura de acuerdo con la acción seleccionada.
Excepciones	▪ Datos del formulario incorrectos. (Muestra mensaje de alerta)
Importancia	Baja
Estado	Completo

Tabla 37. Descripción Caso de uso 6: Gestión de asignaturas

CU-7	Gestión de matrículas
Descripción	Permite listar, modificar, crear y eliminar matrículas del sistema.
Actores	Usuario administrador.
Dependencias	
Precondición	El usuario debe estar autentificado en el sistema y ser administrador.
Secuencia normal	1. Ir a las pantallas de gestión de matrículas. 2. Ingresar datos necesarios para la acción seleccionada.
Postcondición	El sistema se configura de acuerdo con la acción seleccionada.
Excepciones	▪ Datos del formulario incorrectos. (Muestra mensaje de alerta)
Importancia	Baja
Estado	Completo

Tabla 38. Descripción Caso de uso 7: Gestión de matrículas

CU-8	Ver chats
Descripción	Permite al usuario ver los chats en los que está registrado en el sistema.
Actores	Usuario administrador, profesor y estudiante.
Dependencias	
Precondición	El usuario debe estar autentificado en el sistema.
Secuencia normal	1. Ir a las pantallas de chats.
Postcondición	El sistema permite la visualización de la lista de chats del usuario.
Excepciones	
Importancia	Alta
Estado	Completo

Tabla 39. Descripción Caso de uso 8: Ver chats

CU-9	Crear chats
Descripción	Permite al usuario crear un nuevo chat privado, grupal o de asignatura si el usuario que realiza la acción es un profesor.
Actores	Usuario administrador, profesor y estudiante.
Dependencias	
Precondición	El usuario debe estar autentificado en el sistema.
Secuencia normal	1. Ir a las pantallas de chats. 2. Pulsar el botón de crear chat 3. Rellenar el formulario.
Postcondición	El sistema crea un nuevo chat con las condiciones impuestas por el usuario.
Excepciones	▪ Datos del formulario incorrectos. (Muestra mensaje de alerta)
Importancia	Alta
Estado	Completo

Tabla 40. Descripción Caso de uso 9: Crear chats

CU-10	Entrar a un chat
Descripción	Permite al usuario entrar y visualizar los mensajes del chat seleccionado.
Actores	Usuario administrador, profesor y estudiante.
Dependencias	CU-8
Precondición	El usuario debe estar autentificado en el sistema y pertenecer al chat que se quiere entrar.
Secuencia normal	1. Seleccionar chat
Postcondición	El sistema muestra el chat con los mensajes.
Excepciones	
Importancia	Alta
Estado	Completo

Tabla 41. Descripción Caso de uso 10: Entrar a chat

CU-11	Salir de un chat
Descripción	Permite al usuario abandonar el chat seleccionado siempre que no sea un chat privado.
Actores	Usuario administrador, profesor y estudiante.
Dependencias	CU-9
Precondición	El usuario debe estar autentificado en el sistema y el chat debe ser un chat grupal o de asignatura.
Secuencia normal	1. Pulsar en el botón de salir del chat
Postcondición	El sistema saca al usuario del chat seleccionado.
Excepciones	
Importancia	Baja
Estado	Completo

Tabla 42. Descripción Caso de uso 11: Salir de un chat

CU-12	Editar chat
Descripción	Permite al usuario editar el nombre y avatar del chat siempre que tenga permisos de administración en el chat y sea un chat privado.
Actores	Usuario administrador, profesor y estudiante.
Dependencias	CU-9
Precondición	El usuario debe estar autentificado en el sistema, tener permisos de administración en el chat y ser un chat grupal o de asignatura.
Secuencia normal	1. Pulsar el botón de editar. 2. Rellenar el formulario con los campos que se desean modificar.
Postcondición	El usuario cambia los datos del chat.
Excepciones	▪ Datos de formulario incorrectos. (Muestra mensaje de alerta) ▪ Permisos insuficientes en el chat.
Importancia	Baja
Estado	Completo

Tabla 43. Descripción Caso de uso 12: Editar chat

CU-13	Dar permisos en el chat
Descripción	Permite conceder permisos de administración a los integrantes de un chat grupal.
Actores	Usuario administrador, profesor y estudiante.
Dependencias	CU-9
Precondición	El usuario debe estar autentificado en el sistema, tener permisos de administración en el chat y el chat debe ser grupal o de asignatura.
Secuencia normal	1. Pulsar el botón de conceder permisos al usuario.
Postcondición	Se añade a la lista de usuarios con permisos del chat.
Excepciones	
Importancia	Baja
Estado	Completo

Tabla 44. Descripción Caso de uso 13: Dar permiso en el chat

CU-14	Enviar mensaje en un chat
Descripción	Permite al usuario enviar un mensaje en el chat seleccionado.
Actores	Usuario administrador, profesor y estudiante.
Dependencias	CU-9
Precondición	El usuario debe estar autentificado en el sistema.
Secuencia normal	1. Rellenar el formulario para crear un mensaje. 2. Enviar el formulario.
Postcondición	El mensaje se registra en el chat.
Excepciones	▪ Datos del formulario incorrectos. (Muestra mensaje de alerta)
Importancia	Alta
Estado	Completo

Tabla 45. Descripción Caso de uso 14: Enviar mensaje en un chat

CU-15	Responder mensaje
Descripción	Permite al usuario responder un determinado mensaje del chat.
Actores	Usuario administrador, profesor y estudiante.
Dependencias	CU-9
Precondición	El usuario debe estar autentificado en el sistema.
Secuencia normal	1. Elegir un mensaje del chat. 2. Enviar mensaje de respuesta.
Postcondición	El mensaje se registra en el chat y se añade como respuesta al mensaje seleccionado.
Excepciones	▪ Datos del formulario incorrectos. (Muestra mensaje de alerta)
Importancia	Alta
Estado	Completo

Tabla 46. Descripción Caso de uso 15: Responder mensaje

CU-16	Ver hilo de mensajes
Descripción	Permite al usuario ver un hilo dentro de un chat del mensaje seleccionado.
Actores	Usuario administrador, profesor y estudiante.
Dependencias	CU-9
Precondición	El usuario debe estar autentificado en el sistema.
Secuencia normal	1. Elegir un mensaje del chat que contenga respuestas.
Postcondición	El sistema muestra el hilo del mensaje seleccionado.
Excepciones	
Importancia	Alta
Estado	Completo

Tabla 47. Descripción Caso de uso 16: Ver hilo de mensajes

CU-17	Añadir mensaje a preguntas destacadas de la asignatura
Descripción	Permite al usuario profesor seleccionar un hilo del chat para añadirlo a la sección de “Preguntas destacadas” de la web de la asignatura.
Actores	Usuario profesor.
Dependencias	CU-9
Precondición	El usuario debe estar autentificado en el sistema y el chat debe ser de asignatura.
Secuencia normal	1. Elegir un mensaje del chat que contenga respuestas. 2. Pulsar el botón de añadir a favoritos. 3. Rellenar el formulario
Postcondición	El sistema añade el hilo de mensajes a la sección de preguntas destacadas de la web de la asignatura.
Excepciones	▪ Formulario con datos erróneos (Muestra mensaje de alerta)
Importancia	Alta
Estado	Completo

Tabla 48. Descripción Caso de uso 17: Añadir mensaje a preguntas destacadas

CU-18	Ver asignaturas
Descripción	Permite al usuario ver la lista de sus asignaturas que está cursando o ha cursado.
Actores	Usuario profesor y estudiante.
Dependencias	
Precondición	El usuario debe estar autentificado en el sistema.
Secuencia normal	1. Pulsar Mis asignaturas. 2. Elegir curso.
Postcondición	El sistema muestra las asignaturas que este cursando o impartiendo el usuario
Excepciones	
Importancia	Media
Estado	Completo

Tabla 49. Descripción Caso de uso 18: Ver asignaturas

CU-19	Ver asignatura
Descripción	Permite al usuario ver una asignatura de su lista de asignaturas.
Actores	Usuario profesor y estudiante.
Dependencias	CU-17
Precondición	El usuario debe estar autentificado en el sistema.
Secuencia normal	1. Seleccionar la asignatura de la lista
Postcondición	El sistema muestra l asignatura seleccionada
Excepciones	
Importancia	Media
Estado	Completo

Tabla 50. Descripción Caso de uso 19: Ver asignatura

CU-20	Editar preguntas destacadas de asignatura
Descripción	Permite al usuario eliminar o editar las preguntas destacadas de la asignatura.
Actores	Usuario profesor.
Dependencias	C-18
Precondición	El usuario debe estar autentificado en el sistema y ser docente de la asignatura.
Secuencia normal	1. Ir a las pantallas de preguntas destacadas. 2. Ingresar datos necesarios para la acción seleccionada.
Postcondición	El sistema muestra las asignaturas que esté cursando o impartiendo el usuario
Excepciones	
Importancia	Baja
Estado	Completo

Tabla 51. Descripción Caso de uso 20: Editar pregunta destacada

4. DISEÑO DE LA APLICACIÓN

Este capítulo se especificarán las tecnologías y herramientas utilizadas para el desarrollo de la aplicación web, así como la arquitectura final de la aplicación que nos permita complacer los requisitos y objetivos expuestos anteriormente.

4.1. TECNOLOGÍAS APLICADAS

En esta sección se explicarán las principales tecnologías y herramientas que se han usado durante el desarrollo y las pruebas de la aplicación web, tanto las usadas para el desarrollo del frontend como del backend, así como los servicio cloud que han utilizado para su despliegue.

4.1.1. TECNOLOGÍAS DEL LADO DEL CLIENTE.

A continuación, explicaremos las tecnologías que hemos usado para el desarrollo del frontend de la aplicación.

4.1.1.1. REACT

React es una librería de código abierto de JavaScript creada por Facebook para el desarrollo de interfaces web interactivas. Permite el diseño de vistas simples formadas por componentes con estados propios (datos), los cuales React se encarga de renderizar y actualizar de forma eficiente cuando sus estados cambian, además su programación declarativa ayuda a que el código sea más predecible y fácil de depurar.

Ilustración 35. Librería de React

Nos hemos decantado por esta librería para realizar el frontend de la aplicación web por su baja curva de aprendizaje, el gran número de librerías de terceros existentes para añadir funcionalidades o componentes y por la gran cantidad de información y tutoriales disponibles.

A continuación, se mostrarán algunas de las librerías de terceros utilizadas con ReactJS para el desarrollo de la aplicación web:

- **React Router**: es una librería de enrutamiento estándar para ReactJS.
- **React Infinite Scroll**: es una librería que nos proporciona poder deslizarnos por una ventana, sin necesidad de interactuar con elementos de paginación y navegación.
- **React Toastr**: es una librería JavaScript para mostrar notificaciones web visuales, interactivas y con muchas opciones.
- **React tranistion group**: es una librería que nos proporciona de manera fácil realizar animaciones cuando un componente React ingresa o sale del DOM.
- **Formik & Yup**: es un grupo de librerías que facilita el uso de formularios y su validación en ReactJS.
- **React icons**: es una librería que nos permite incluir iconos en nuestra aplicación.
- **Axios**: es una librería JavaScript que permite hacer peticiones HTTP.
- **Emoji Picker**: es una librería que proporciona un selector de emojis.
- **Laravel Echo & Pusher**: es una librería JavaScript para conectarnos al servidor de websocket (Pusher o Laravel websockets) indicando qué canal y evento queremos escuchar.

4.1.1.2. TAILWIND CSS

Tailwind CSS es un framework de desarrollo para el lado del cliente, mediante el cual podemos disponer de una serie de clases de CSS predefinidas para el rápido desarrollo del diseño de la aplicación, además este framework ofrece una gran integración con la librería de ReactJS y ofrece clases CSS para crear un diseño responsive fácilmente.

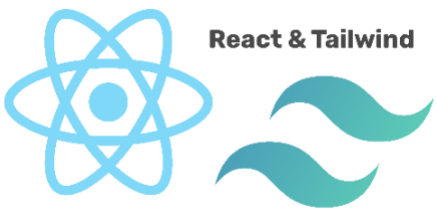


Ilustración 36. Logo Tailwind CSS con React

4.1.1.3. SURGE

Para facilitar y automatizar el despliegue del frontend realizado con ReactJS se ha decidido usar Surge, que es un servicio para implementar y alojar sitios web y aplicaciones estáticas con un par de líneas de comando, además surge nos proporciona un dominio y alojamiento gratuito.

Ilustración 37. Logo Surge

4.1.2. TECNOLOGÍAS DEL LADO DEL SERVIDOR

En esta sección se expondrán las tecnologías utilizadas en el lado de servidor o backend para el desarrollo de la aplicación y despliegue de esta.

4.1.2.1. PHP

A la hora de elegir un lenguaje interpretado para el lado del servidor se han barajado varías tecnologías como NodeJS, PHP, Ruby o Python. Al final nos hemos decantado por usar PHP (Hypertext Preprocessor), que se trata del lenguaje de código abierto más popular para el desarrollo web y nos permitirá usar Laravel, que es uno de los frameworks de desarrollo web más usados y con más comunidad de desarrolladores de la actualidad.

PHP sirve para crear aplicaciones web dinámicas con acceso a información almacenada en una base de datos y que permite incorporar código en las páginas HTML de forma sencilla, se ejecuta en el lado del servidor generando el HTML y enviándolo al cliente, además cuenta con una extensa biblioteca de funciones y una gran comunidad de desarrolladores. [19]

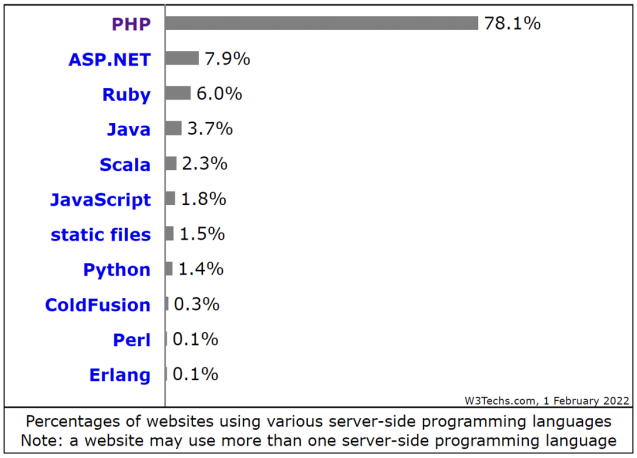


Ilustración 38. Porcentaje de servidores web usando PHP

4.1.2.2. LARAVEL

Laravel es un marco de trabajo en PHP de código abierto que sigue el patrón MVC (modelo-vista-controlador) desarrollado y mantenido por Taylor Otwell con el objetivo de facilitar el desarrollo de aplicaciones y servicios web a través de herramientas y funciones integradas. [20]

Las razones por la que nos hemos decantado por Laravel por encima de los otros frameworks de desarrollo para nuestra aplicación en el lado del servidor es por la gran cantidad de herramientas y funciones que incorpora, como por ejemplo, un completo sistema de autentificación integrado, motor de enrutamiento, inyección de dependencias, además de contar con Eloquent que es un ORM que permite interactuar con la base de datos de una manera orientada a objetos que facilita las gestiones con la base de datos.

Ilustración 39. Logo Laravel

Otra de las razones por las que hemos elegido Laravel es por su facilidad de uso y aprendizaje debido a su gran documentación, que cuenta con ejemplos de uso

para cada una de las herramientas y funciones que nos proporciona. Asimismo, Laravel proporciona una gran cantidad de paquetes propios y de terceros de fácil instalación, que nos facilita el desarrollo de cualquier tipo de aplicación o servicio web, para nuestra aplicación la lista de paquetes utilizados es la siguiente:

- **Laravel websockets**: es un paquete que nos permite establecer un canal de comunicación abierto entre el cliente y el servidor que nos proporcionará respuestas instantáneas en los chats de la aplicación a través del protocolo de comunicación websockets.

- **FFMPEG**: es un paquete que proporciona utilidades para editar videos e imágenes como por ejemplo comprimir, escalar, cortar o cambiar el formato que nos permitirá ahorrar espacio en el disco del servidor, así como crear thumbnails de los videos para ahorrar recursos de red en la parte del Frontend.

4.1.2.3. MYSQL

Para tener una persistencia de los datos en nuestra aplicación es necesaria una base de datos. Como se dijo en capítulos pasados las BBDB se dividen en dos grandes grupos: bases de datos relacionales (SQL) y bases de datos no relacionales (NoSQL).

Puesto que usaremos Laravel como framework para la realización de nuestro backend la elección sobre el tipo de base de datos es sencilla, ya que Laravel no ofrece soporte por defecto para una base de datos no relacional.

Por tanto, la tecnología elegida para realizar y gestionar la base de datos en nuestro proyecto es MySQL, que se trata de gestor de base de datos relacional de código abierto de fácil instalación y uso en los sistemas operativos Linux.

Ilustración 40. Logo MySQL

4.1.2.4. APACHE

Nuestra aplicación tiene que poder servir de alguna manera los archivos a sitios web en Internet, por ello necesitamos un servidor web, para nuestro proyecto se ha elegido Apache por encima de Nginx porque ya teníamos experiencia previa utilizando este servidor web.

Ilustración 41. Logo Apache Web Server

4.1.2.5. AMAZON WEB SERVICES

Para desplegar nuestra API de Laravel, base de datos en internet se ha decidido hacer uso de Amazon Web Services (AWS) que es una plataforma proveedora de servicios en la nube, que ofrece más de 200 servicios, en los cuales entre ellos están almacenamiento, recursos de computación, aplicaciones móviles, bases de datos, servidores privados virtuales (VPS), etc. [21]

Ilustración 42. Logo Amazon Web Services

Se ha elegido esta plataforma para el despliegue porque ofrece un plan gratuito para un servidor privado virtual en la nube de 1GiB de memoria RAM, 1vCPU, 12GB de almacenamiento y sistema operativo Linux, que es más que suficiente para cubrir los recursos necesarios para el correcto funcionamiento de la API, base de datos y servidor de websockets.

4.1.3. HERRAMIENTAS DE DESARROLLO

En esta sección hablaremos sobre las herramientas que se han utilizado para el desarrollo y pruebas de la aplicación web.

4.1.3.1. VISUAL STUDIO CODE

Visual Studio Code es un editor de código fuente multiplataforma que permite trabajar con distintos lenguajes de programación, control de versiones con Git e incluso crear tus propios atajos de teclado y refactorizar código. Es de código abierto y nos proporciona la utilidad para descargar y gestionar extensiones con la que podemos personalizar y potenciar esta herramienta. [22]

Ilustración 43. Logo Visual Studio Code

Las extensiones que hemos utilizado en Visual Studio Code para el desarrollo de este proyecto son las siguientes:

- **PHP Debug & PHP Intelephense**: Nos ayuda a encontrar errores, resalta el código de PHP y autocompletar fragmentos de código de PHP.
- **Laravel Snippets**: Proporciona autocompletado para código del framework de Laravel.
- **ES7+React/Redux/React-Native snippets**: Proporciona autocompletado para código de la librería de ReactJS y ECMAScript 7.
- **Tailwind CSS IntelliSense**: Proporciona autocompletado de código para el framework de Tailwind CSS.
- **Material Icon Theme**: Personaliza los iconos dentro del editor de Visual Studio Code.
- **Community Material Theme**: Personaliza la interfaz de Visual Studio Code.

En este proyecto se ha decidido utilizar Visual Studio Code como el editor de código para el desarrollo tanto del backend con Laravel y del frontend con React por sus extensiones que ayudan y agilizan la creación de código con estas tecnologías.

4.1.3.2. MYSQL WORKBENCH

MySQL Workbench es una herramienta de diseño de bases de datos que integra desarrollo de software, administración de bases de datos, diseño de bases de datos, gestión y mantenimiento para el sistema de base de datos MySQL mediante un interfaz de usuario que ayuda a agilizar estas funciones. [23]

Su uso en este proyecto únicamente será para el diseño de la base de datos y creación de un modelo de datos que posteriormente será integrado en Laravel.

Ilustración 44. Logo MySQL Workbench

4.1.3.3. TABLEPLUS

TablePlus es una herramienta que permite administrar bases de datos como PostgreSQL, MySQL, MariaDB, SQL Server y más con una interfaz de usuario elegante y sencilla. A su vez, permite la conexión a bases de datos remotas por medio de diferentes tipos de conexiones.

Durante el desarrollo de este proyecto ha sido usada simplemente para ver el contenido de la base de datos mediante una conexión SSH de una forma sencilla y gráfica. [24]

Ilustración 45. Logo TablePlus

4.1.3.4. POSTMAN

Postman es una aplicación que nos permite realizar pruebas API a través de un cliente HTTP que nos da la posibilidad realizar peticiones HTTP a través de una interfaz gráfica de usuario en la cual obtendremos diferentes tipos de respuestas devueltas por la API introducida de una manera rápida, fácil y visual. [25]

Durante el desarrollo de la aplicación usaremos Postman para verificar el correcto funcionamiento de los endpoints de la API REST de Laravel.

Ilustración 46. Logo Postman

4.1.3.5. MOBAXTERM

MobaXTerm es una herramienta que nos proporciona funciones diseñadas para programadores, webmasters, administradores de TI para gestionar y administrar servidores tanto remotos como locales mediante diferentes tipos de conexiones como: SSH, X11, RDP, VNC, FTP, MOSH, etc. [26]

MobaXTerm será usada durante el desarrollo de este proyecto para realizar la configuración en el servidor EC2 de Amazon AWS mediante una conexión SSH al servidor remoto.

Ilustración 47. Logo MobaXTerm

4.1.3.6. CHROME DEVTOOLS

Chrome DevTools es un conjunto de herramientas para desarrolladores web integradas en el navegador de Google Chrome que permiten entre otras características la visualización de las webs en diferentes tamaños de pantallas, así como añadir estilos a la web mientras se está ejecutando. Chrome DevTools también nos permite añadir nuevas herramientas mediante extensiones fácilmente descargables en Chrome Web Store. [27]

Ilustración 48. Logo Google Chrome DevTools

Las extensiones que hemos usado para este proyecto son:

- **React Developers Tools**: extensión que permite inspeccionar las jerarquías de componentes de React.
- **Lighthouse**: herramienta que nos permite hacer una serie de pruebas contra la página para comprobar su rendimiento.

Chrome DevTools se usará para realizar pruebas de diseño en la aplicación web directamente desde el navegador y visualizar la aplicación en diferentes tamaños de pantalla, además con las extensiones añadidas podremos depurar el código de React y hacer auditorías una vez esté finalizada la web para comprobar el rendimiento de esta.

4.1.3.7. GIT Y GITHUB

Git es una herramienta de control de versiones distribuida que puede gestionar el historial de código fuente de un proyecto de desarrollo, mientras que GitHub es una plataforma que puede mantener repositorios de código en almacenamiento basado en la nube para que varios desarrolladores puedan trabajar en un solo proyecto y ver las ediciones de cada uno en tiempo real. [28]

Ilustración 49. Logo Git y GitHub

Durante la realización del proyecto se ha utilizado el control de versiones Git y la plataforma de GitHub para poder trabajar en equipo en el desarrollo del backend y posteriormente de manera individual para llevar un control de los cambios realizados en el frontend.

4.1.3.8. NODEJS Y NPM

Node.js es un entorno de tiempo de ejecución a tiempo real de JavaScript que incluye todo necesario para ejecutar un programa escrito en JavaScript. [29]

NPM es un gestor de paquetes para Javascript, es decir, sirve para instalar y gestionar versiones de paquetes y librerías de JavaScript.

Ilustración 50. Logo NodeJS y NPM

Estas tecnologías se han usado durante el desarrollo del Frontend de la aplicación, donde hemos usado NodeJS para ejecutar la aplicación de React en nuestra máquina y el gestor de paquetes NPM para descargar herramientas y librerías durante el desarrollo.

4.2. ARQUITECTURA

En esta sección se explicarán los patrones y las técnicas utilizadas para el diseño y desarrollo de la aplicación web. Como primer punto de esta sección explicaremos cómo es la arquitectura genérica de una aplicación web y posteriormente expondremos la arquitectura final elegida para la realización de nuestro proyecto.

4.2.1. ARQUITECTURA DE LAS APLICACIONES WEB

Las aplicaciones web están basadas en una arquitectura cliente-servidor, donde los usuarios utilizan un navegador web como cliente para conectarse a un servidor mediante una conexión de red. [30] [31]

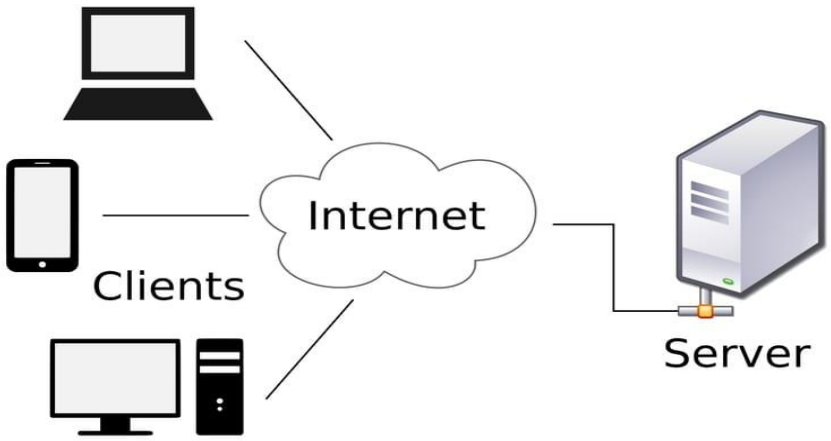


Ilustración 51. Modelo cliente-servidor

El cliente a través del navegador solicita una página web al servidor mediante una conexión de red que usa el protocolo HTTP. Una vez la petición es recibida

por el servidor web, localiza la página web en su sistema de archivos y la envía de vuelta al cliente que la solicitó. Una vez se ha entregado la página web al navegador, la conexión entre el cliente y el servidor se rompe, produciendo que el navegador web no tenga acceso directo a los recursos del servidor web.

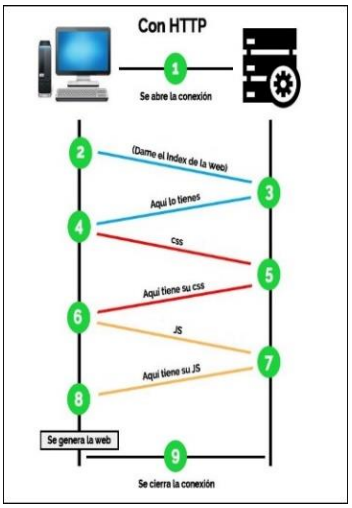


Ilustración 52. Protocolo HTTP

Tradicionalmente la arquitectura de una aplicación web constaba de dos componentes, el sistema del lado del cliente o la interfaz de usuario y un sistema en el lado del servidor que solía ser un servidor de base de datos, donde la lógica de negocio se incorporaba totalmente en la interfaz de usuario, a este modelo de la arquitectura se le conocía como modelo de dos capas o niveles. Este modelo presentaba una serie de desventajas como baja escalabilidad, reducido número de conexiones, alta carga de red, flexibilidad reducida y baja seguridad debido a la interacción directa de los usuarios con la base de datos.

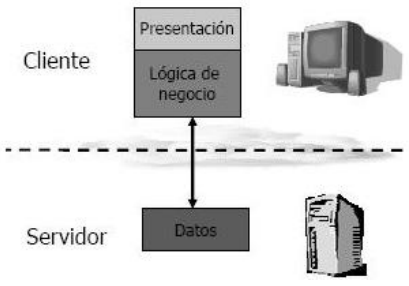


Ilustración 53. Modelo de dos capas

Para superar estas limitaciones de la arquitectura ajustada al modelo de dos capas se introduce una capa intermedia, entre la capa de presentación (interfaz de usuario) y la capa de datos (servidor de base de datos), en esta capa se

centraliza la lógica de negocio haciendo la administración más sencilla, flexible y aumentando la seguridad de la aplicación.

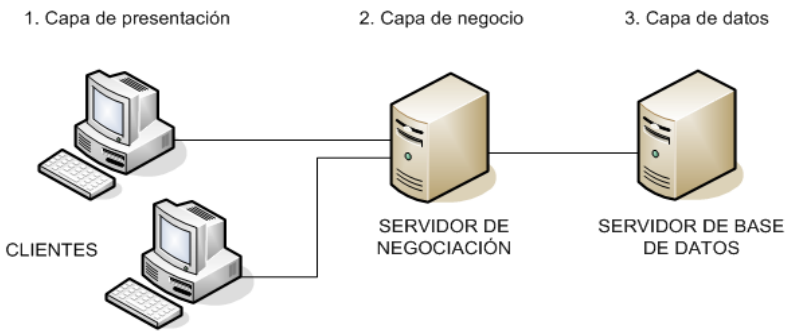


Ilustración 54. Modelo de tres capas

Este modelo es llamado modelo de tres capas o niveles y consta de:

1. **Capa de presentación,** esta capa permite a los usuarios interactuar con el servidor o el servicio del Backend a través de una interfaz de usuario en el navegador web, todo el código en esta capa reside en el navegador es decir en el lado del cliente y se encarga de recibir las solicitudes y presentar la información al usuario.

Ilustración 55. Capa de presentación

2. **Capa de negocio**, esta capa se encarga de recibir las solicitudes de los usuarios e interactuar con la capa de datos para realizar los procesos de negocio de la aplicación y mandar los resultados obtenidos a la capa de presentación o Frontend.

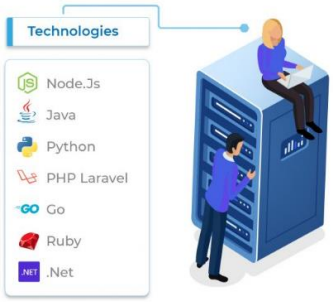


Ilustración 56. Capa de negocio

3. **Capa de datos** (Servidor de datos), esta capa se encarga de almacenar y suministrar los datos a la capa o nivel de negocio.

Ilustración 57. Capa de datos

En este modelo cada capa interactúa solamente con sus capas adyacentes, es decir la capa de presentación no puede interactuar con la capa de datos directamente, lo que permite mayor seguridad, flexibilidad, escalabilidad y facilidad en el desarrollo de la aplicación.

4.2.2. ARQUITECTURA FINAL DEL PROYECTO

Esta sección aborda la descripción detalla del diseño de la arquitectura final elegida para cumplir con los requisitos y objetivos de este proyecto. A continuación, se muestra un diagrama de la arquitectura final que seguirá la aplicación.

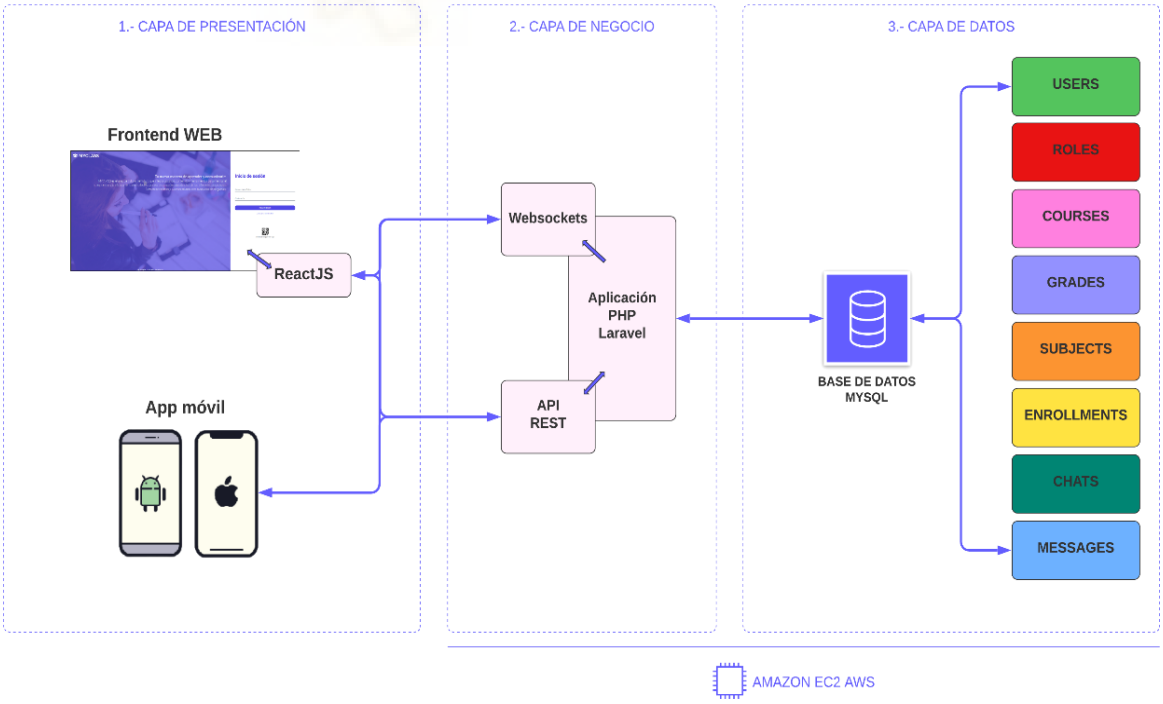


Ilustración 58. Arquitectura final del proyecto

Como podemos observar en el diagrama de la ilustración 58, la aplicación web sigue la arquitectura de 3 capas o niveles, donde la capa de presentación hace referencia al Frontend de la aplicación y la capa de negocio y la capa de datos hacen referencia a lo que actualmente se conoce como Backend.

Esta sección se dividirá en dos partes:

- **Arquitectura del Backend**. En este apartado se explicará el funcionamiento y relación de todas las tecnologías usadas en el lado del servidor, así como el diseño del modelo de datos.

- **Arquitectura del Frontend**. En este apartado se explicará la arquitectura de la tecnología usada en Frontend, así como diagrama de vistas de la aplicación en el lado del cliente.

4.2.2.1. ARQUITECTURA DEL BACKEND

Para poder comunicar nuestro Frontend ya sea una aplicación móvil o una aplicación web con nuestra base de datos se ha desarrollado una API o interfaz de programación de aplicaciones, que consiste en un conjunto de reglas que determinan como las aplicaciones y dispositivos pueden conectarse y comunicarse entre sí, en otras palabras una API permite interactuar con una computadora o un sistema para obtener datos o ejecutar una función, de manera que el sistema comprenda la solicitud y la cumpla independientemente de dónde ésta provenga. [32]

En nuestro caso desarrollaremos una API REST, que es un estilo de arquitectura de software que se utiliza para describir cualquier interfaz entre diferentes sistemas que utilice HTTP para comunicarse. El termino REST significa Representational State Tranfer (transferencia de datos representacional) que consiste en que entre dos llamadas cualesquiera el servicio no guarda los datos, es decir, cada llamada es totalmente independiente.

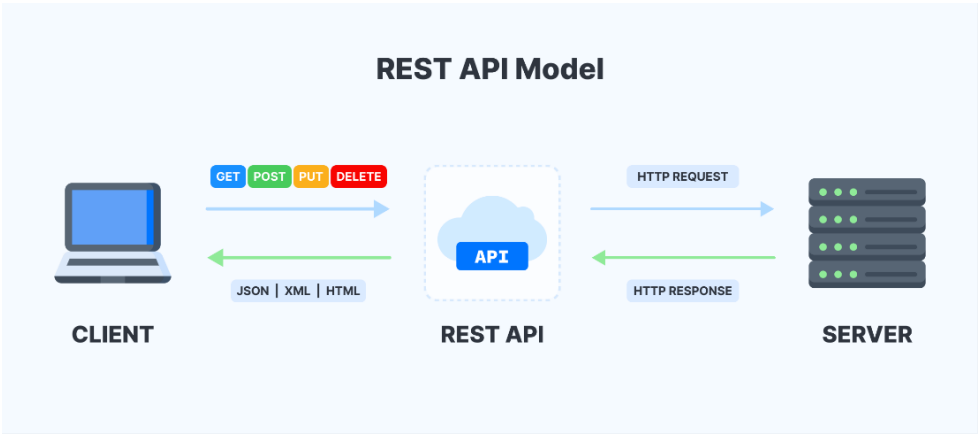


Ilustración 59. Modelo API Rest

Como se ha mencionado una API REST se comunica a través de solicitudes HTTP donde existen diferentes métodos que le dirán a la API que hacer con el recurso solicitado, estos métodos son los siguientes:

- **POST**: sirve para la creación de un recurso en la aplicación.
- **GET**: se utiliza para obtener un recurso de la aplicación.
- **PUT**: nos permite actualizar un recurso de la aplicación.
- **DELETE**: se utiliza para eliminar un recurso de la aplicación.

En pocas palabras, una API REST es un servicio del Backend que es capaz de contestar a las llamadas a una serie de URLs también llamadas endpoints normalmente en formato JSON, y que además es capaz de recibir información por medio del cuerpo de la llamada para poder realizar funciones de actualización y creación en los recursos disponibles en la aplicación.

Para la realización de la API REST se ha optado por usar el framework de código abierto **Laravel** [33], que nos ofrece multitud de herramientas y funcionalidades fácilmente incorporables a través de su gestor de paquetes Composer. Asimismo, Laravel ofrece un desarrollo fácil, rápido y modular gracias a la implementación casi inmediata en la estructura de la aplicación del patrón MVC (Modelo-Vista-Controlador).

El **patrón MVC** permite separar la aplicación en tres módulos claramente identificables y con funcionalidad bien definida: El Modelo, las Vistas y el Controlador. [34]

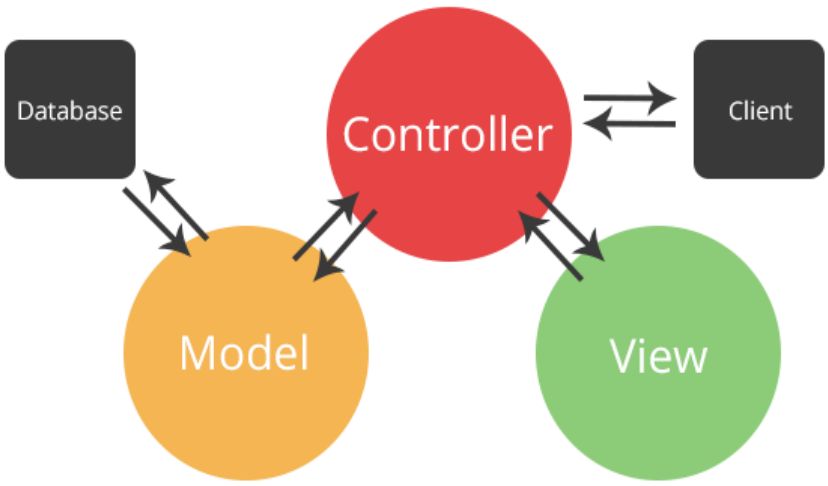


Ilustración 60. Arquitectura Modelo-Vista-Controlador

- **Modelo**: esta capa se encarga de trabajar con los datos y contiene los mecanismos que permiten acceder y modificar los datos de nuestra base de datos.

- **Controlador**: en esta capa se implementa la lógica de negocio de la aplicación, es decir los procedimientos, algoritmos y rutinas necesarias para el funcionamiento de la aplicación. Actúa como intermediario entre la capa del modelo y la capa de vista aplicando las transformaciones y lógica necesarias.

- **Vista**: esta capa se responsabiliza de la representación de la información por medio de una interfaz de usuario que por lo general representan los datos provenientes del modelo o estos transformados en un proceso en el controlador.

Para el caso de este proyecto las vistas no serán utilizadas, ya que desde el controlador enviaremos la respuesta en formato JSON a nuestro Frontend creado con ReactJS, aunque cabe destacar que Laravel nos proporciona un motor de plantillas simples y poderoso que permite usar PHP plano en ellas y ofrece un desarrollo modular pudiendo dividir las vistas en componentes como se muestra en la figura siguiente.

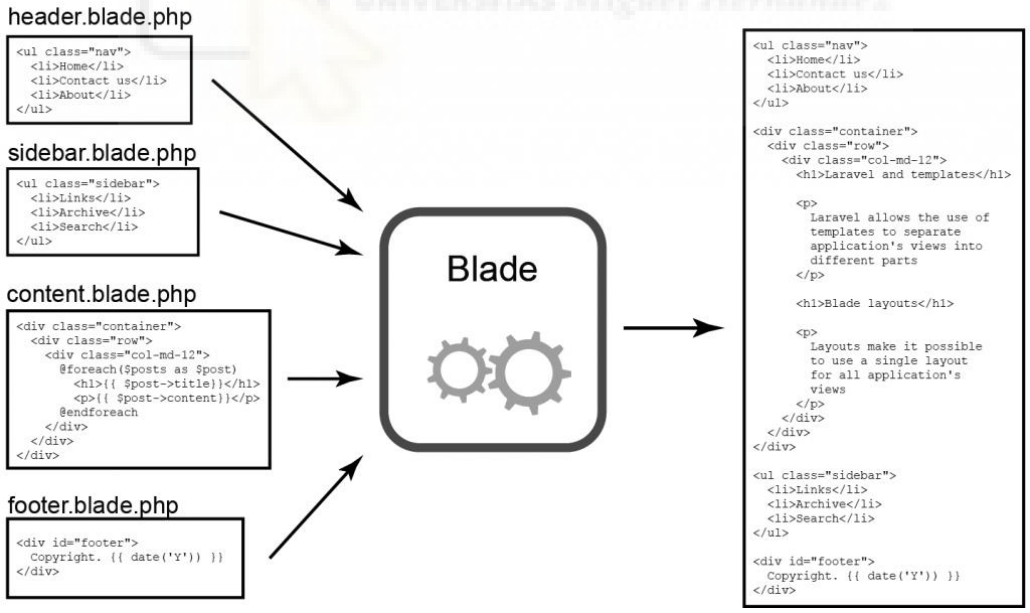


Ilustración 61. Diagrama del motor de plantillas de Laravel

A continuación, se expondrán algunas de las características del framework de Laravel que se han usado para el desarrollo e implementación de la arquitectura de la aplicación en el lado del Backend:

- **Consola Artisan**: es una interfaz de línea de comandos integrada en Laravel, que provee comandos para agilizar el desarrollo de la aplicación.

- **Eloquent ORM**: es un modelo de programación que permite mapear las estructuras de una base de datos relacional que implementa el patrón de arquitectura Active Record, es decir, que cada una de las tablas de la base de datos corresponde a un modelo de Laravel que interactúa con ella. Cada instancia del modelo en cuestión pertenece a un registro en la tabla que genera una persistencia en la base de datos, a su vez cuando se produzca una actualización de los atributos de los modelos también se producirá en la base de datos.

Ilustración 62. Laravel Eloquent ORM

- **Migraciones**: son un sistema de control de versiones de la base de datos, que permite modificar y compartir el esquema de la base de datos de la aplicación, que generalmente se combinan con el generador de esquemas de Laravel para construir el esquema de la base de datos de la aplicación.

- **Seeds y Factories**: son métodos de Laravel que nos van a permitir poblar de información de prueba la base de datos de la aplicación.

- **Sistema de enrutamiento**: Laravel proporciona un sistema de enrutamiento sencillo y fácil de desarrollar que define un mapa entre los métodos de HTTP, las URLs y los controladores.

- **Autorización con Sanctum**: Laravel Sanctum proporciona un sistema de autenticación ligero para aplicaciones de una sola página, aplicaciones móviles y API simples basadas en tokens que está completamente integrado en el marco de Laravel permitiendo tener un sistema de autentificación seguro y fiable en segundos.

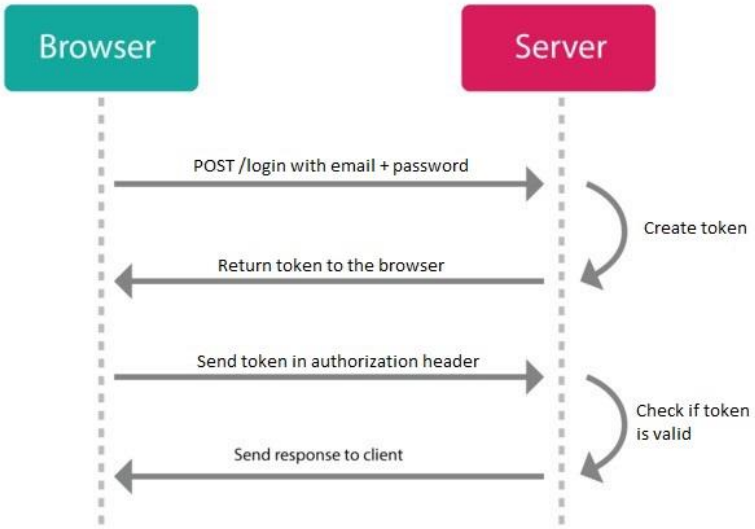


Ilustración 63. Autentificación mediante Laravel Sanctum

- **Autorización con policies**: son clases de Laravel que nos permitirán organizar la lógica de autorización para un modelo o recurso en particular.

- **Resources**: es una clase de Laravel que nos permite añadir una capa de transformación entre los modelos de Eloquent y la respuesta JSON que devolverá un controlador de la aplicación.

- **Eventos**: proporcionan una implementación de observador simple, que permite suscribirse y escuchar varios eventos que ocurren en su aplicación, en nuestro caso será usada para realizar los eventos de chat mediante una conexión websockets donde el receptor del evento será el Frontend.

Como se observó en la ilustración 58, donde se muestra la arquitectura final de la aplicación, se hará uso de una comunicación vía Websockets que servirá para conseguir una comunicación en tiempo real en la aplicación del chat.

Websockets es un protocolo de comunicación (TCP) que permite mantener una comunicación bidireccional abierta entre el cliente y el servidor, que permite que las aplicaciones web envíen y reciban datos sin tiempos de espera como por ejemplo en juegos multijugador o aplicaciones de chat como en el caso de este proyecto. [35]

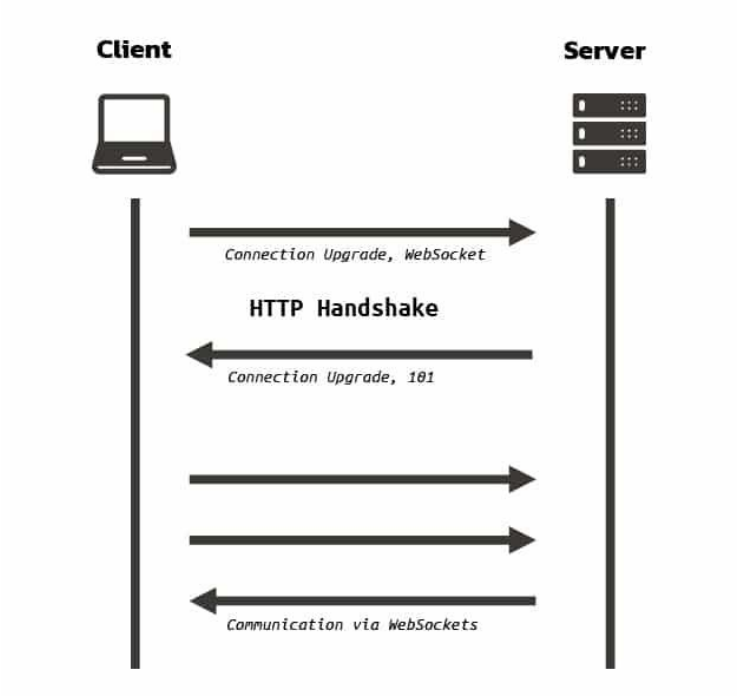


Ilustración 64. Protocolo Websockets

Para implementar de forma sencilla esta tecnología, Laravel nos proporciona un sistema de transmisión de eventos del lado del servidor al lado del cliente utilizando un enfoque basado en Websockets mediante un paquete llamado **Laravel Websockets** que utiliza la API integrada de Pusher en Laravel.

Ilustración 65. Logo Websockets

Laravel Websockets nos permite crear un servidor de Websockets en PHP totalmente funcional de una manera fácil y sencilla, donde podemos enviar eventos desde el marco de trabajo de Laravel que se transmiten por medio de “canales” que pueden ser especificados como públicos o privados y que posteriormente pueden ser consumidos fácilmente desde el cliente utilizando una librería en JavaScript llamada **Laravel Echo**.

En la ilustración siguiente se muestra el funcionamiento de la arquitectura de websockets para implementar uno de los eventos existentes dentro de la aplicación del chat.

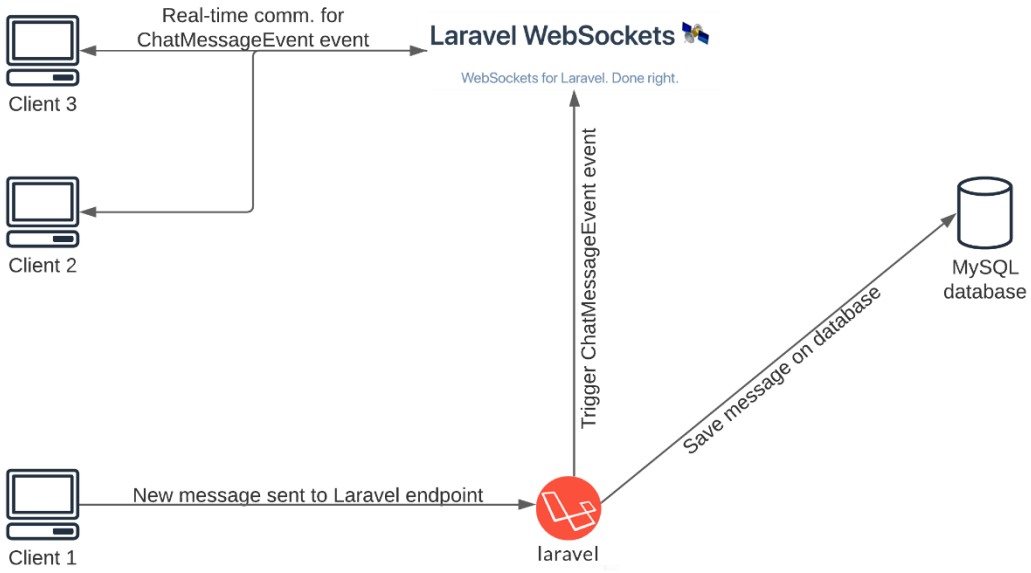


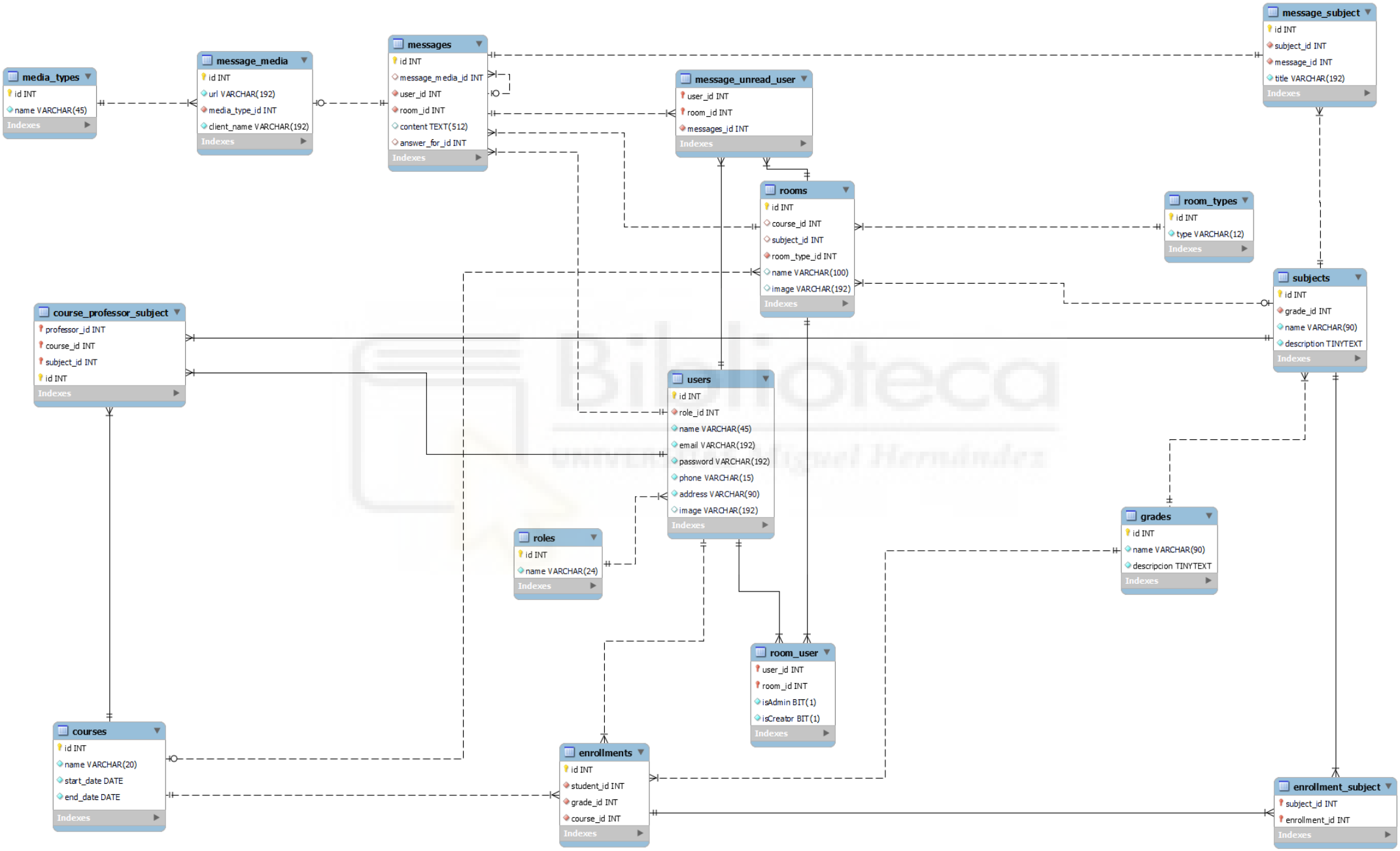
Ilustración 66. Uso de eventos con Laravel Websockets

Como se puede observar en la figura anterior el cliente 1 crea un nuevo mensaje en la aplicación haciendo una petición POST al Endpoint de la aplicación del Backend en Laravel, la cual guarda el mensaje en la base de datos para que sea persistente y envía el evento "ChatMessageEvent" al servidor Websocket creado con Laravel Websockets que a su vez retrasmite en el canal del chat el evento, que llega a todos los clientes que estén escuchan el canal del chat y el evento "ChatMessageEvent" mediante la librería de JavaScript Laravel Echo.

Como última parte en la arquitectura del Backend explicaremos el diseño de la base de datos, la cual estará conectada a nuestra aplicación de Laravel para conseguir la persistencia de los datos mediante el gestor de base de datos MySQL.

A continuación, se muestra un diagrama del modelo de la base de datos creado con MySQL Workbench donde se muestran los atributos de cada tabla y sus relaciones que serán posteriormente implementadas mediante el esquema de migraciones de Laravel y Eloquent ORM. Cabe recordar que Laravel agregará más tablas de manera automática para poder gestionar la autentificación y otras características internas que no serán mostradas en este diagrama.

Ilustración 67. Esquema modelo de datos de la aplicación



Como podemos observar la base de datos consta de 16 tablas en las que se almacenan los diferentes datos para el correcto funcionamiento de la aplicación. A continuación se muestra la estructura y las relaciones de las diferentes tablas pertenecientes a la base de datos.

Tabla roles

Tabla			
Nombre	roles		
Descripción	Almacena los roles disponibles para los usuarios de la aplicación.		
Campos			
Nombre	**Tipo**	**Descripción**	**Nulo**
id	INT	Identificador del rol.	
name	VARCHAR	Nombre del rol.	
Relaciones			
▪ Relación 1 a muchos con la tabla usuarios			

Tabla 52 . Base de datos: Roles

Tabla Users

Tabla				
Nombre	users			
Descripción	Contiene la información referente a los usuarios registrados en los establecimientos.			
Campos				
PK	**Nombre**	**Tipo**	**Descripción**	**Nulo**
✓	id	INT	Identificador del usuario.	
	role_id	INT	Identificador del rol al que pertenece el usuario.	
	name	VARCHAR	Nombre de usuario.	
	email	VARCHAR	Correo electrónico del usuario.	
	password	VARCHAR	Contraseña de usuario (Encriptada con Laravel).	
	phone	VARCHAR	Teléfono del usuario.	
	address	VARCHAR	Dirección del usuario.	
	image	VARCHAR	URL de la imagen del usuario en el servidor.	✓
Claves foráneas				
Campo	**Tabla referenciada**	**Campo referenciado**		
role_id	roles	id		
Relaciones				
▪ Relación 1 a muchos inversa con la tabla roles. ▪ Relación 1 a muchos con la tabla enrollments. ▪ Relación muchos a muchos con la tabla rooms mediante tabla pivote room_user. ▪ Relación 1 a muchos con la tabla messages. ▪ Relación 1 a muchos con la tabla message_unread_user. ▪ Relación muchos a muchos con la tabla courses y subjects mediante tabla pivote course_professor_subject.				

Tabla 53. Base de datos: Users

Tabla courses

Tabla				
Nombre	courses			
Descripción	Almacena los cursos o años lectivos.			
Campos				
PK	**Nombre**	**Tipo**	**Descripción**	**Nulo**
✓	id	INT	Identificador del curso.	
	name	VARCHAR	Nombre del curso.	
	start_date	DATE	Fecha de inicio del curso.	
	end_date	DATE	Fecha de finalización del curso.	
Relaciones				
▪ Relación muchos a muchos con la tabla users y subjects mediante table pivote course_professor_subject. ▪ Relación 1 a muchos con la tabla rooms. ▪ Relación 1 a muchos con la tabla enrollments.				

Tabla 54. Base de datos: Courses

Tabla subjects

Tabla				
Nombre	subjects			
Descripción	Almacena las asignaturas de la aplicación.			
Campos				
PK	**Nombre**	**Tipo**	**Descripción**	**Nulo**
✓	id	INT	Identificador de asignatura.	
	grade_id	INT	Identificador del grado al que pertenece la asignatura.	
	name	VARCHAR	Nombre de la asignatura.	
	description	VARCHAR	Descripción de la asignatura.	
Claves foráneas				
Campo		**Tabla referenciada**	**Campo referenciado**	
grade_id		grades	id	
Relaciones				
▪ Relación 1 a muchos inversa con la tabla grades. ▪ Relación muchos a muchos con las tablas users y courses mediante la tabla pivote course_professor_subject. ▪ Relación muchos a muchos con la tabla enrollments mediante la tabla pivote enrollment_subject. ▪ Relación 1 a muchos con la tabla rooms. ▪ Relación 1 a muchos con la tabla message_subject.				

Tabla 55. Base de datos: Subjects

Tabla enrollments

Tabla				
Nombre	enrollments			
Descripción	Almacena las matrículas de los estudiantes, es decir las asignaturas seleccionadas por curso y grado.			
Campos				
PK	**Nombre**	**Tipo**	**Descripción**	**Nulo**
✓	id	INT	Identificador de asignatura.	
	student_id	INT	Identificador del usuario al que pertenece la matrícula.	
	grade_id	INT	Identificador del grado al que pertenece la matrícula.	
	course_id	INT	Identificador del curso al que pertenece la matrícula.	

Claves foráneas		
Campo	**Tabla referenciada**	**Campo referenciado**
student_id	users	id
grade_id	grades	id
course_id	courses	id

Relaciones

- Relación 1 a muchos inversa con la tabla users.
- Relación 1 a muchos inversa con la tabla grades.
- Relación 1 a muchos inversa con la tabla courses.
- Relación muchos a muchos con la tabla subjects mediante la tabla pivote enrollment_subject.

Tabla 56. Base de datos: Enrollments

Tabla grades

Tabla				
Nombre	grades			
Descripción	Almacena los grados que hay disponible en la aplicación.			
Campos				
PK	**Nombre**	**Tipo**	**Descripción**	**Nulo**
✓	id	INT	Identificador del grado.	
	name	VARCHAR	Nombre del grado.	
	description	VARCHAR	Descripción del grado.	

Relaciones

- Relación 1 a muchos con la tabla subjects.
- Relación 1 a muchos con la tabla enrollments.

Tabla 57. Base de datos: Grades

Tabla enrollment_subject

Tabla				
Nombre	enrollment_subject			
Descripción	Tabla pivote que relaciona las matrículas con las asignaturas.			
Campos				
PK	**Nombre**	**Tipo**	**Descripción**	**Nulo**
	subject_id	INT	Identificador de la asignatura al que se relaciona con la matrícula.	
	enrollment_id	INT	Identificador de la matrícula que se relaciona con la asignatura.	
Claves foráneas				
Campo		**Tabla referenciada**	**Campo referenciado**	
subject_id		subjects	id	
enrollment_id		enrollments	id	
Relaciones				
▪ Tabla pivote que relaciona las tablas enrollments y subjects (muchos a muchos).				

Tabla 58. Base de datos: Tabla Pivote enrollment_subject

**Clave primaria compuesta por subject_id y enrollment_id*

Tabla course_professor_subject

Tabla				
Nombre	course_professor_subject			
Descripción	Tabla pivote que almacena la relación de las asignaturas que imparte un profesor por año.			
Campos				
PK	**Nombre**	**Tipo**	**Descripción**	**Nulo**
	professor_id	INT	Identificador del usuario que se relaciona con el curso y asignatura.	
	course_id	INT	Identificador del curso que se relaciona con el profesor y la asignatura.	
	subject_id	INT	Identificador de la asignatura que se relaciona con el profesor y curso.	
Claves foráneas				
Campo		**Tabla referenciada**	**Campo referenciado**	
professor_id		users	id	
course_id		courses	id	
subject_id		subjects	id	
Relaciones				
▪ Tabla pivote que relaciona las tablas users, courses y subjects (muchos a muchos).				

Tabla 59. Base de datos: Tabla pivote course_professor_subject

**Clave primaria compuesta por subject_id, course_id y subject_id*

Tabla room_types

Tabla				
Nombre	room_types			
Descripción	Almacenan los diferentes tipos de salas de chat que existen en la aplicación.			
Campos				
PK	**Nombre**	**Tipo**	**Descripción**	**Nulo**
✓	id	INT	Identificador de tipo de sala de chat.	
	type	VARCHAR	Nombre tipo sala de chat.	
Relaciones				
▪ Relación 1 a muchos con la tabla rooms.				

Tabla 60. Base de datos: Room types

Tabla rooms

Tabla				
Nombre	rooms			
Descripción	Almacena las diferentes salas del chat de la aplicación.			
Campos				
PK	**Nombre**	**Tipo**	**Descripción**	**Nulo**
✓	id	INT	Identificador de sala de chat.	
	course_id	INT	Identificador del curso al que pertenece la sala de chat.	✓
	subject_id	INT	Identificador de la asignatura al que pertenece la sala de chat.	✓
	room_type_id	INT		
	name	VARCHAR	Si el tipo de sala es grupal podemos almacenar el nombre del grupo.	✓
	image	VARCHAR	Si el tipo de sala es grupal podemos añadir una imagen de avatar a la sala.	✓

Claves foráneas		
Campo	**Tabla referenciada**	**Campo referenciado**
course_id	courses	id
subject_id	subjects	id
room_type_id	room_types	id

Relaciones

- Relación 1 a muchos inversa con la tabla subjects.
- Relación 1 a muchos inversa con la tabla room_types.
- Relación 1 a muchos inversa con la tabla courses.
- Relación 1 a muchos con la tabla messages.
- Relación 1 a muchos con la tabla message_unread_user.
- Relación muchos a muchos con la tabla users mediante la tabla pivote room_users.

Tabla 61. Base de datos: Rooms

Tabla room_users

Tabla				
Nombre	room_users			
Descripción	Tabla pivote que almacena la relación de las salas de chat con los usuarios que la componen y además indica el creador y si tiene poderes de administración de la sala de chat.			
Campos				
PK	**Nombre**	**Tipo**	**Descripción**	**Nulo**
	user_id	INT	Identificador del usuario que se relaciona con la sala de chat.	
	room_id	INT	Identificador de la sala de chat del usuario.	
	isAdmin	Boolean	Indica si el usuario en cuestión tiene poderes de administrador de la sala.	
	isCreator	Boolean	Indica si el usuario es el creador de la sala de chat.	
Claves foráneas				
Campo	**Tabla referenciada**	**Campo referenciado**		
user_id	users	id		
room_id	rooms	id		
Relaciones				
▪ Tabla pivote que relaciona las tablas users y rooms (muchos a muchos).				

Tabla 62. Base de datos: Tabla pivote room_users

**Clave primaria compuesta por user_id y room_id*

Tabla message_media

Tabla				
Nombre	message_media			
Descripción	Almacena el tipo del archivo, la URLs donde se encuentra el archivo multimedia y el nombre con el que lo subió el cliente.			
Campos				
PK	**Nombre**	**Tipo**	**Descripción**	**Nulo**
✓	id	INT	Identificador del archivo de un mensaje.	
	url	VARCHAR	Url que nos lleva a donde se encuentra almacenado el archivo.	
	media_type_id	INT	Identificador del tipo de archivo.	
	client_name	VARCHAR	Nombre del archivo cuando lo subió el usuario.	
Claves foráneas				
Campo	**Tabla referenciada**	**Campo referenciado**		
media_type_id	media_types	id		
Relaciones				
▪ Relación 1 a 1 con la tabla messages. ▪ Relación 1 a muchos inversa con la tabla media_types.				

Tabla 63. Base de datos: Message media

Tabla media_types

Tabla				
Nombre	media_types			
Descripción	Tabla que almacena los diferentes tipos que pueden tener los archivos de los mensajes.			
Campos				
PK	**Nombre**	**Tipo**	**Descripción**	**Nulo**
✓	id	INT	Identificador de tipo de archivo que contiene el mensaje.	
	name	VARCHAR	Nombre del tipo de archivo.	
Relaciones				
▪ Relación 1 a muchos con la tabla message_media.				

Tabla 64. Base de datos: Media types

Tabla messages

Tabla				
Nombre	messages			
Descripción	Almacena los mensajes de las diferentes salas de chat de la aplicación.			
Campos				
PK	**Nombre**	**Tipo**	**Descripción**	**Nulo**
✓	id	INT	Identificador de sala de chat.	
	message_media_id	INT	Identificador del archivo del mensaje.	✓
	user_id	INT	Identificador del usuario al que pertenece el mensaje.	
	room_id	INT	Identificador de la sala del chat a la que pertenece el mensaje.	
	content	TEXT	Contenido del mensaje.	✓
	answer_for_id	INT	Identificador de la pregunta a la que pertenece el mensaje.	✓

Claves foráneas		
Campo	**Tabla referenciada**	**Campo referenciado**
message_media_id	message_medias	id
user_id	users	id
room_id	rooms	id
answer_for_id	messages	id

Relaciones
▪ Relación 1 a 1 con la tabla message_media. ▪ Relación reflexiva answer_for_id con la tabla messages. ▪ Relación 1 a 1 con la tabla message_subject. ▪ Relación 1 a muchos con la tabla message_unread_user. ▪ Relación 1 a muchos inversa con la tabla rooms. ▪ Relación 1 a muchos inversa con la tabla users.

Tabla 65. Base de datos: Messages

Tabla message_unread_user

Tabla				
Nombre	message_unread_user			
Descripción	Tabla pivote que relaciona si un mensaje de una sala de chat ha sido leído por usuario.			
Campos				
PK	**Nombre**	**Tipo**	**Descripción**	**Nulo**
	message_id	INT	Identificador del mensaje que no está leído en la sala de chat por un usuario.	
	user_id	INT	Identificador del usuario que en una sala de chat tiene un mensaje sin leer.	
	room_id	INT	Identificador de la sala de chat que tiene un mensaje sin leer por un usuario.	

Claves foráneas		
Campo	**Tabla referenciada**	**Campo referenciado**
message_id	messages	id
user_id	users	id
room_id	rooms	id

Relaciones

- Tabla pivote que relaciona las tablas users, rooms y messages (muchos a muchos).

Tabla 66. Base de datos: Tabla pivote message_unread_user

**Clave primaria compuesta por user_id y room_id*

Tabla message_subject

Tabla				
Nombre	message_subject			
Descripción	Tabla que almacena los mensajes seleccionados por el profesor para que aparezcan sus hilos en la sección de preguntas destacadas de la asignatura.			
Campos				
PK	**Nombre**	**Tipo**	**Descripción**	**Nulo**
✓	id	INT	Identificador del mensaje guardado en la asignatura.	
	subject_id	INT	Identificador de la asignatura del mensaje guardado.	
	message_id	INT	Identificador del mensaje guardado por la asignatura.	
	title	VARCHAR	Titulo descriptivo del hilo del mensaje.	

Claves foráneas		
Campo	**Tabla referenciada**	**Campo referenciado**
subject_id	subjects	id
message_id	messages	id

Relaciones

- Relación 1 a 1 con la tabla messages.
- Relación 1 a muchos inversa con la tabla subjects.

Tabla 67. Base de datos: Tabla pivote message_subject

Para finalizar este apartado cabe mencionar que toda la estructura del backend será desplegada mediante amazon AWS en una instancia EC2 que nos ofrece un plan gratuito que consiste en un servidor privado virtual en la nube de 1GiB de memoria RAM, 1vCPU, 12GB de almacenamiento y sistema operativo Linux.

4.2.2.2. ARQUITECTURA DEL FRONTEND

Para realizar la capa de presentación o Frontend de nuestra aplicación se ha decidido usar una de las tecnologías más populares para la realización de interfaces de usuario dinámicas como es la librería de **ReactJS**.

ReactJS es una biblioteca de JavaScript que crea interfaces de usuario dinámicas de una manera predecible y eficiente utilizando código declarativo mediante **JSX**, que es una extensión para la sintaxis del lenguaje JavaScript, que proporciona una forma de estructurar la representación de componentes utilizando la sintaxis de HTML con todo el poder de JavaScript, es decir nos permite mezclar HTML y JavaScript. Normalmente, React es usada para crear aplicaciones web de una sola página (SPA) como es el caso de nuestra aplicación para el lado del cliente, pero además es capaz de crear aplicaciones móviles o de escritorio con ayuda de otras librerías como React Native o ElectronJS. [36]

En ReactJS el elemento más importante son los **componentes**, que son pequeñas piezas de código reutilizables que permiten dividir la aplicación en bloques separados que actúan de forma independiente entre sí, los cuales reciben una serie de propiedades e implementan su propia lógica de renderizado mediante el uso de hooks de ReactJS.

Los hooks son funciones que nos permiten utilizar el manejo de estados y otros métodos de la librería de ReactJS que nos facilitaran la creación de componentes y manejo del DOM como veremos en el siguiente capítulo de esta memoria, algunos de los hooks más usados en las aplicaciones de ReactJS son:

- **useState**: permite crear y cambiar el estado de un componente.
- **useEffect**: permite realizar efectos secundarios cuando se produce un cambio en un determinado estado dentro de componente.
- **useReducer**: permite gestionar estados complejos con un reducer que es una función que toma el estado anterior y una acción para devolver un nuevo estado.
- **useContext**: permite tener estados compartidos entre diferentes componentes incluso cuando no son componentes hijos o hermanos.

El objetivo de ReactJS es diseñar vistas simples para cada estado de la aplicación, donde React actualizara y renderizará de manera eficiente el componente correcto cuando cambie alguno de sus datos (estados).

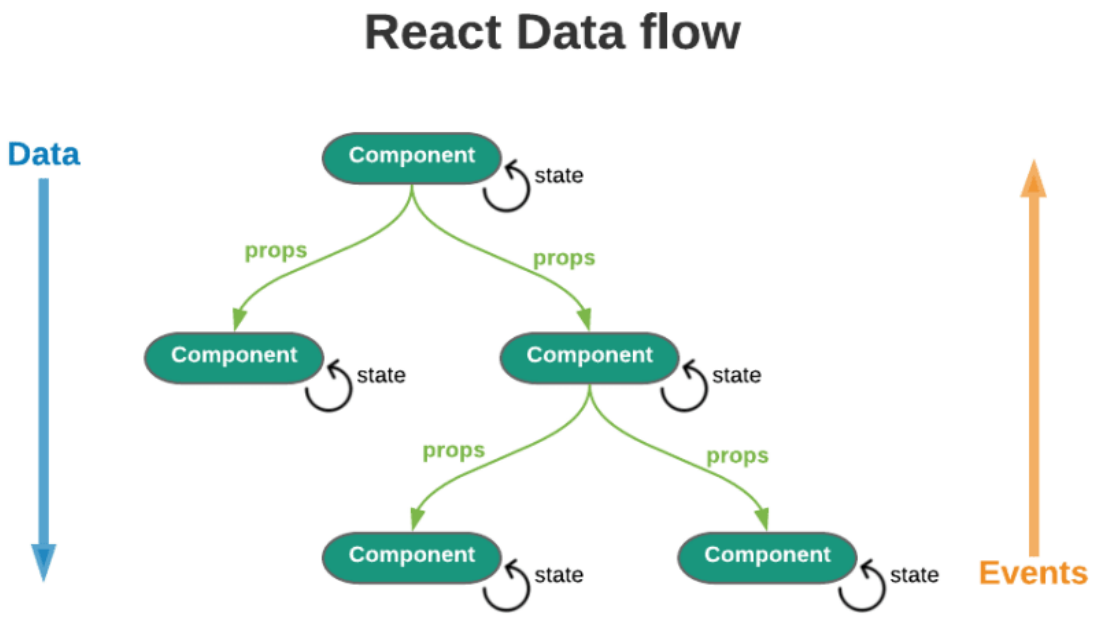


Ilustración 68. Diagrama de flujo información React

Como podemos observar en el diagrama de la figura anterior, una aplicación de React está compuesta por múltiples componentes, cada uno de los cuales es responsable de representar una pequeña pieza de HTML reutilizable, estos componentes se pueden anidar dentro de otros componentes para permitir que se construyan interfaces de usuario complejas a partir de bloque de construcción simples. Estos bloques simples se conocen como componentes y pueden contener un estado interno (hooks) que se utiliza para realizar una renderización eficiente del componente cuando se produzca algún cambio en los datos, esto produce interfaces dinámicas, rápidas, altamente escalables y personalizables.

Para realizar estas renderizaciones tan rápidas y eficientes ReactJS hace uso de un **Virtual DOM**, pero antes de ver que es un Virtual DOM explicaremos que es el DOM.

El DOM (Document Object Model) es un modelo o representación gráfica del documento HTML de una aplicación web creado por el navegador, que define como los programas pueden acceder para modificar su estructura, estilo y contenido. El DOM da una representación del documento como un grupo de nodos y objetos estructurados que tienen propiedades y métodos. [37]

Todo el proceso de carga, actualización y renderizado del DOM puede llegar a ser muy costoso en cuanto a recursos en el lado del cliente, sobre todo en aplicaciones grandes, ya que en cada actualización que sea necesaria en la aplicación el DOM volverá a renderizar cualquier elemento en el que existan cambios junto a todos sus nodos descendientes sea necesario o no.

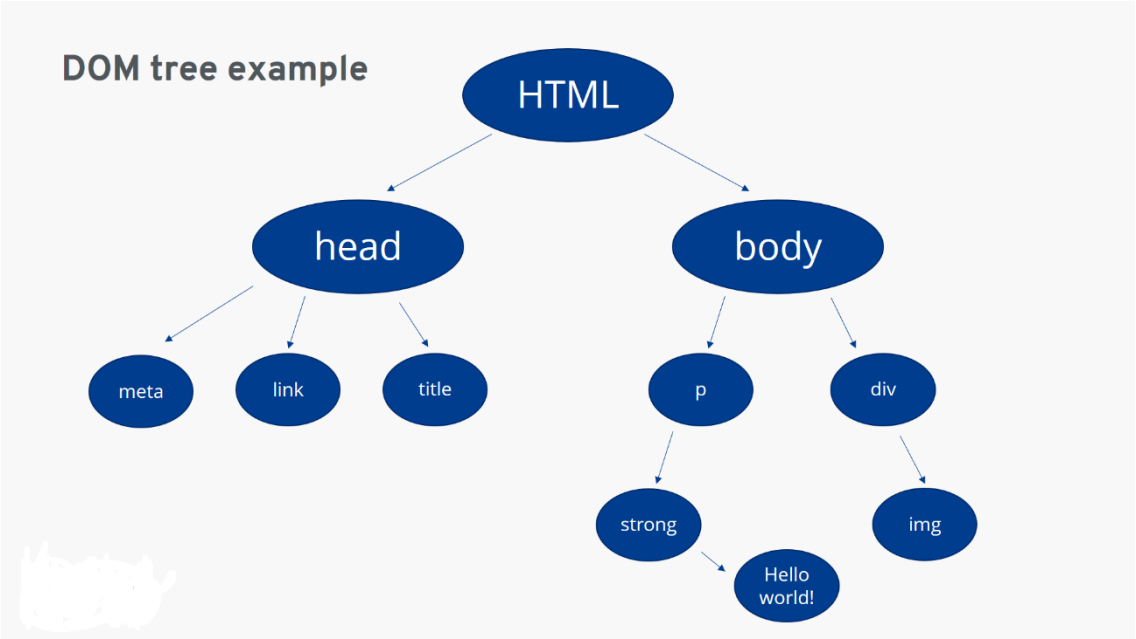


Ilustración 69. Ejemplo árbol de componentes HTML

Por ello ReactJS para optimizar el renderizado de los componentes hace uso de un Virtual DOM que consiste en una copia del DOM guardada en memoria, que actúa de intermediario entre el estado de la aplicación y los estados del DOM, el Virtual DOM interpreta dichos cambios y calcula la manera más eficiente de actualizar el DOM para que se rendericen la menor cantidad de cambios posibles, ahorrando recursos de procesamiento en el cliente y brindando una experiencia de usuario más fluida a nuestra aplicación web. [38]

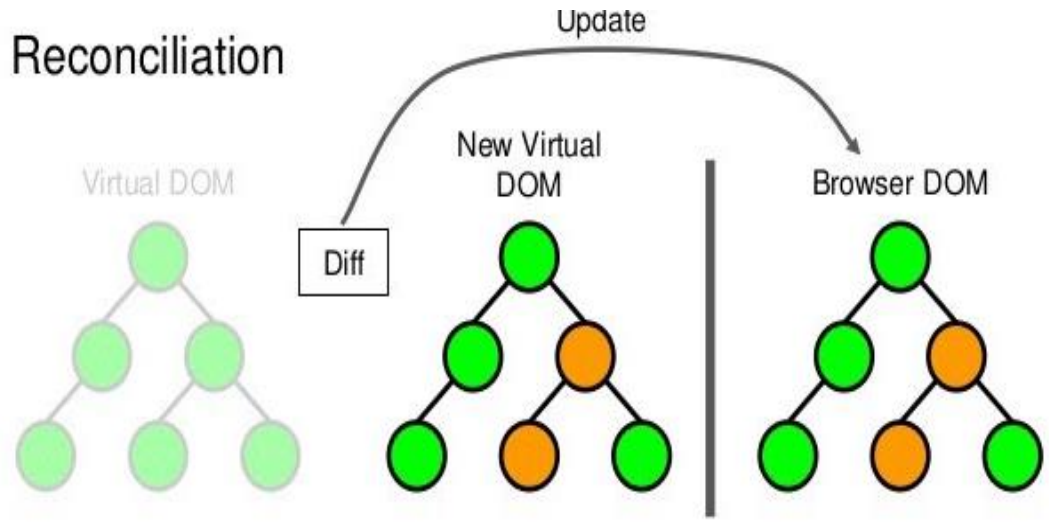


Ilustración 70. Ejemplo de Funcionamiento de renderizado de React

Como se ha comentado anteriormente React es una librería y no un marco de trabajo completo, por ello haremos uso de librerías de terceros que la acompañen para facilitar el desarrollo de la aplicación como por ejemplo **Laravel Echo** que nos servirá para conectarnos a nuestro servidor websocket, **Axios** para conectarnos a nuestra API o **React Router** que nos proveerá de un sistema de enrutamiento, etc. Además, para facilitar la creación del diseño de nuestra aplicación se utilizará un framework de CSS llamado **TailwindCSS** que se complementa muy bien con ReactJS.

Una vez entendido el funcionamiento básico de ReactJS para empezar a desarrollar nuestra aplicación del Frontend nos hemos creado unos diagramas que muestran el flujo y las diferentes vistas o páginas que queremos crear en nuestra interfaz de usuario.

Arquitectura de vistas para los usuarios con el rol de estudiante y profesor

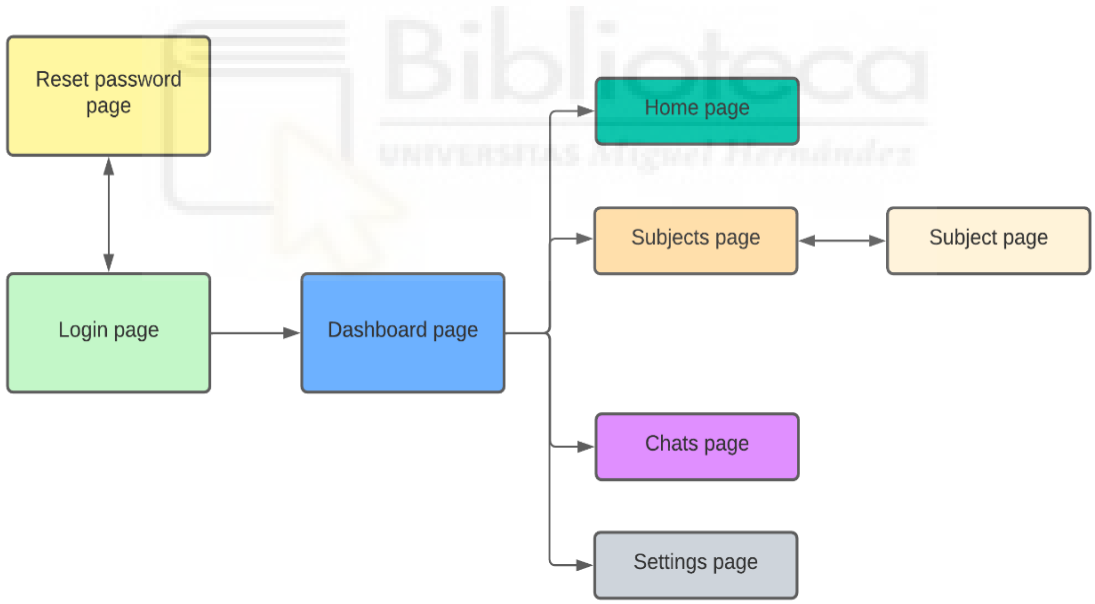


Ilustración 71. Diagrama de vistas de la aplicación para roles de profesor y estudiante

Arquitectura de vistas para los usuarios con el rol de administrador

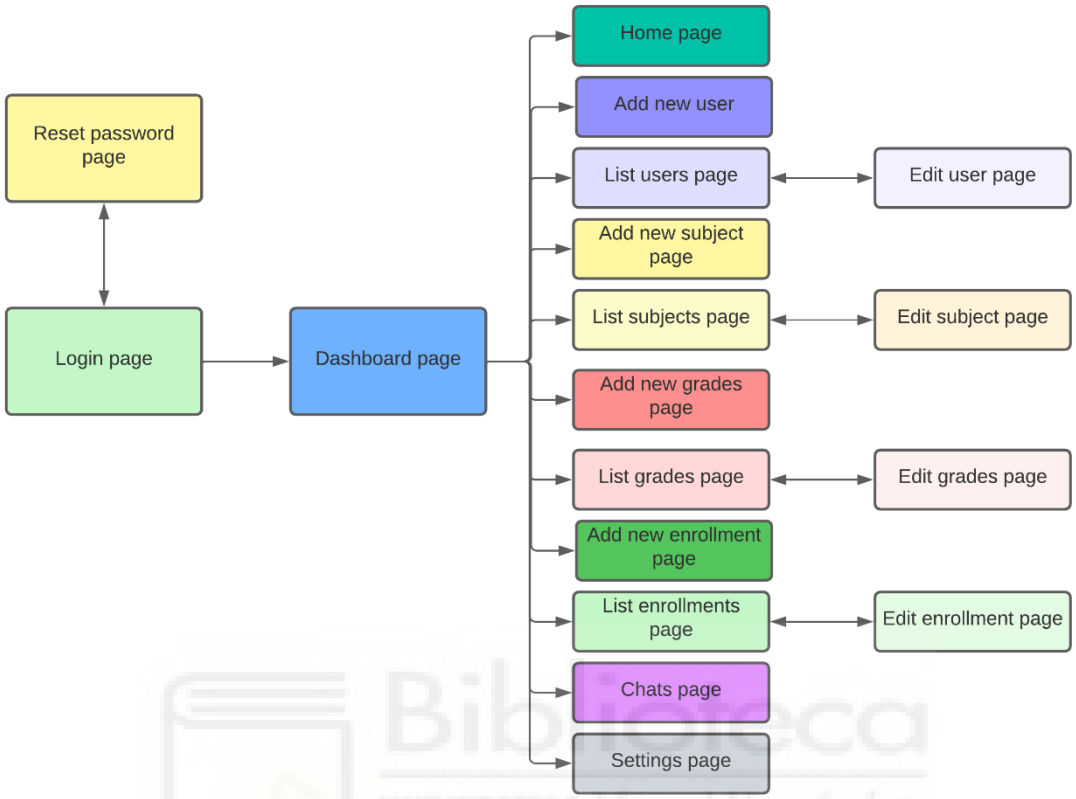


Ilustración 72. Diagrama de vistas de la aplicación para rol administrador

Para finalizar esta sección simplemente quiero recordar que para el despliegue de la parte del Frontend se va a usar Surge.sh que es un servicio para implementar y alojar sitios web y aplicaciones estáticas de una manera sencilla y rápida. Además nos entrega un dominio gratuito. [39]

5. IMPLEMENTACIÓN

En este capítulo de implementación se explicará cómo ha sido desarrollado la aplicación tanto para el lado de servidor como para el lado del cliente con ejemplos de código para los componentes más importantes. Este capítulo se dividirá en dos partes:

- **Implementación del Backend**. En esta sección se explicará la estructura de una aplicación de Laravel y se expondrán ejemplos de código de los elementos más importantes para comprender su funcionamiento.

- **Implementación del Frontend**. En este apartado se expondrá la estructura de una aplicación de React y se explicará con ejemplos de código los elementos más importantes para realizar una aplicación en React.

5.1. IMPLEMENTACIÓN BACKEND

Una vez configurado nuestro servidor con el stack de Lamp (Linux, Apache, MySQL y PHP) [Anexo A] para crear una aplicación de Laravel es necesario instalar **Composer** [40], que es un manejador de paquetes para PHP que proporciona un estándar para administrar, descargar e instalar dependencias y librerías. Además, es necesario instalar las siguientes extensiones de PHP si no se encuentran instaladas en el servidor: *BCMath*, *Ctype*, *DOM PHP*, *Fileinfo*, *JSON*, *Mbstring*, *OpenSSL*, *PCRE*, *PDO*, *Tokenizer*, *XML*.

Una vez tenemos instalado todo lo necesario, ejecutamos desde la consola el siguiente comando para crear la aplicación de Laravel con el nombre “backend-myclass”**:**

```
> php composer create-project laravel/laravel backend-myclass
```

5.1.1. ESTRUCTURA DE UNA APLICACIÓN LARAVEL

En este sección se expondrá la estructura de una aplicación de Laravel al iniciar el proyecto describiendo para que se usa cada directorio y los archivo más importantes que contiene.

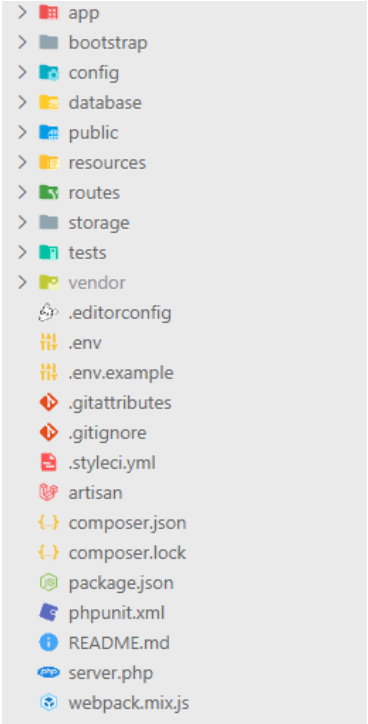


Ilustración 73. Estructura de carpetas Laravel

- **/app**: este directorio contiene el código principal de la aplicación. En esta carpeta entre otros se encuentran los modelos y controladores de la aplicación.

    - **app/http/controllers**: dentro de esta carpeta se encuentran los controladores de la aplicación que se encargan de administrar las peticiones y llamar a los modelos y las vistas necesarias.
    - **app/http/middleware**: en ella se encuentran los filtros para las peticiones que deciden si dejan llegar las peticiones a los controladores.
    - **app/console**: en esta carpeta contiene todos los comandos personalizados de Artisan para su aplicación.
    - **App/models**: en este directorio se encuentran todas las clase de los modelos de Eloquent de la aplicación.
    - **app/exceptions**: se encuentran los archivos para gestionar las excepciones de la aplicación.
    - **app/providers**: se encuentran los proveedores de servicios de la aplicación.

- **/bootstrap**: en este directorio se encuentra el código para inicializar la aplicación y optimizar su rendimiento.

- **/config**: contiene todos los archivos de configuración de la aplicación.

- **/database**: contiene las migraciones de la base de datos, los Factories y las semillas, además, también se puede usar este directorio para incluir una base de datos SQLite.

    - **database/migrations**: contiene las clases para gestionar las tablas de la base de datos mediante el sistema de migraciones de Laravel.
    - **database/factoreis**: en este directorio se encuentran clases para crear datos aleatorios de los modelos de la aplicación.
    - **database/seeders**: contiene las clases que sirven para poblar las base de datos.

- **/public**: este directorio es el punto de entrada de todas las solicitudes que ingresan a la aplicación, contiene el index.php y alberga la imágenes públicas, JavaScript y CSS compilados de la aplicación.

- **/resources**: contiene las vistas de la aplicación, así como el JavaScript y CSS sin compilar.

    - **resources/lang**: contiene los ficheros con las traducciones de los textos de la aplicación.
    - **resources/css**: contiene el css que se usaran en las vistas de su aplicación sin compilar.
    - **resources/js**: en este directorio están los archivos de JavaScript usados por las vistas sin compilar.
    - **resources/views**: contiene las plantillas de Blade que generan las vistas de la aplicación.

- **/routes**: contiene todos los archivos de definición de las rutas para la aplicación.

- **/storage**: este directorio almacena información necesaria para la ejecución de la web como caché o archivos de sesión y además permite el almacenamiento de archivos por parte de los usuarios de la aplicación.

- **/test**: contiene los archivos de pruebas automatizadas de la aplicación.

- **/vendor**: en este directorio están las dependencias de Composer.

- **.env**: archivo donde se encuentran las variables de entorno de la aplicación.

A medida que se añadan nuevos componentes a la aplicación de Laravel se irán añadiendo nuevos directorios de forma automática, por lo que la estructura final de carpetas puede ser distinta al finalizar la aplicación.

5.1.2. DESARROLLO CON LARAVEL

En esta sección se explicará con más detalle y con ejemplo de código los componentes fundamentales usados en nuestro proyecto.

Como ya se dijo anteriormente Laravel es un marco de trabajo que nos facilita la creación de aplicaciones o servicios web con la ayuda de funciones y herramientas integradas o de terceros para conseguir un desarrollo rápido y seguro, por ello quería comentar que para el desarrollo de la autentificación de la aplicación se ha hecho uso de la herramienta integrada en Laravel llamada **Laravel Sanctum** [41], la cual nos crea automáticamente todos los archivos necesarios (controladores, rutas, migraciones y middlewares) para tener una autentificación robusta y segura en nuestra API mediante tokens personales.

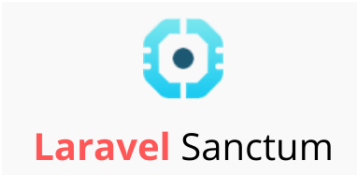


Ilustración 74. Logo Laravel Sanctum

Lo primero para comenzar con el desarrollo en una aplicación de Laravel es hacer la conexión de la base de datos con el proyecto de Laravel. Para ello tenemos que crearnos una nueva base de datos en nuestro gestor de base de datos MySQL mediante la consola de Ubuntu y colocar la configuración de nuestra base de datos en las variables de entorno que se encuentra en la raíz del proyecto en el archivo **.env**.

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

Ilustración 75. Ejemplo variable de entorno Laravel para base de datos

Una vez lista la conexión de la base de datos, el siguiente paso sería pasar nuestro modelo de datos creado con anterioridad en MySQL Workbench a Laravel mediante el uso del sistema de migraciones.

Migraciones

Las migraciones son clases especiales que heredan de la clase Migration que nos permiten crear tablas, modificar o eliminarlas y establecer relaciones entre ellas. Para crear las migraciones usaremos la consola de comandos Artisan con el siguiente comando:

```
> php artisan make:migration create_subjects_table
```

Este comando nos creará una nueva migración con el nombre “create_subjects_table” en el directorio **./database/migrations**, una vez creada simplemente tenemos que añadir los campos y relaciones necesarias especificadas en el modelo de datos como se muestra a continuación:

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateSubjectsTable extends Migration
{
    /* Run the migrations. */
    public function up()
    {
        Schema::create('subjects', function (Blueprint $table) {
            /* campo clave primaria id */
            $table->id();
            /* campo clave foránea a la tabla grades al atributo id
               con eliminación y actualización en cascada */
            $table->foreignId('grade_id')
                  ->constrained()
                  ->onUpdate('cascade')
                  ->onDelete('cascade');
            /* campo name tipo VARCHAR */
            $table->string('name', 90);
            /* campo description tipo TEXT */
            $table->text('description');
            /* campos de marcas temporales */
            $table->timestamps();
        });
    }

    /* Reverse the migrations. */
    public function down()
    {
        Schema::dropIfExists('subjects');
    }
}
```

Como se puede observar, crear las tablas para el modelo de datos usando las migraciones de Laravel es bastante sencillo, sólo hay que definir los campos necesarios y sus relaciones entre las diferentes tablas. Una vez creadas todas las tablas en nuestro proyecto Laravel el directorio de migraciones quedaría como se muestra en la imagen siguiente.

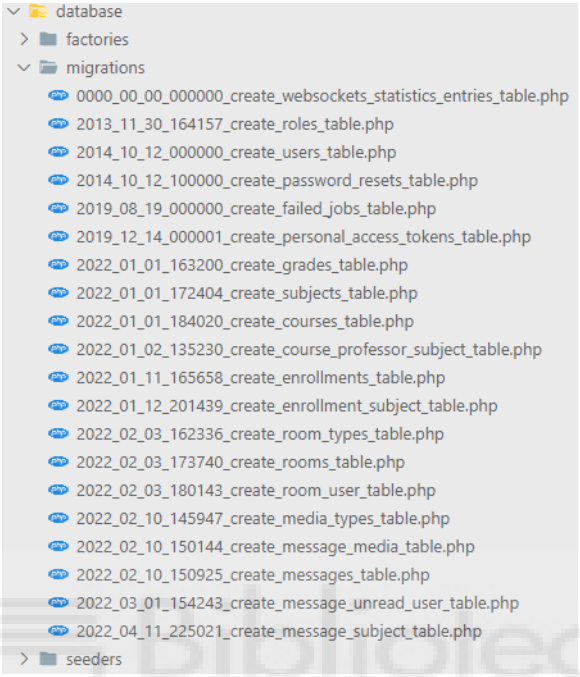


Ilustración 76. Ejemplo carpeta de migraciones Laravel

Una vez tengamos las tablas en nuestro proyecto de Laravel las insertamos en nuestra base de datos mediante la consola de comandos Artisan con el siguiente comando:

```
> php artisan migrate
```

El siguiente paso en nuestro proyecto de Laravel es crear los modelos que realizarán las consultas a la base de datos por medio de **Eloquent ORM**.

Modelos

El marco de trabajo de Laravel incorpora Eloquent ORM que consiste en un mapeador de objetos que facilita enormemente interactuar con la base de datos. Para usar Eloquent ORM hay que crear un “Modelo” por cada tabla que no sea una tabla pivote en la base de datos.

Los Modelos son clases especiales que heredan de la clase Model que se utilizan para interactuar con la base de datos mediante Eloquent ORM. Para crear un

modelo para una tabla en particular, pongamos como ejemplo la tabla “Subjects”, tenemos que llamar al modelo con el singular del nombre de la tabla, que para nuestro caso de ejemplo sería “Subject”, esto hará que Laravel conecte automáticamente el modelo con la tabla en la base de datos.

Para crear un modelo usaremos la consola de Artisan nuevamente con el siguiente comando que creará un modelo llamado “Subject” en **./app/models**:

```
> php artisan make:model Subject
```

A continuación, se mostrará un ejemplo de un modelo de nuestro proyecto de Laravel, donde se muestra la configuración de los diferentes campos que permiten el almacenamiento masivo de datos, además también podemos encontrar mutators para modificar el valor del atributo a la hora de guardar en la base de datos o visualizar en otra clase y se establecen las diferentes relaciones con otros modelos como por ejemplo con el método “grade()”, que accede a la relación establecida en la base de datos y obtiene de la tabla “Grades” el registro indicado y lo devuelve como un modelo (objeto).

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Subject extends Model
{
    use HasFactory;

    /* Campos que se le permite almacenamiento masivo en la base de datos */
    protected $fillable = [
        'grade_id',
        'name',
        'description'
    ];

    /* Mutators de los atributos del modelo */
    public function setNameAttribute($valor){
        $this->attributes['name'] = mb_strtolower($valor, "UTF-8");
    }

    public function getNameAttribute($valor){
        return ucwords($valor);
    }
```

```php
    /* Obtiene el grado al que pertenece la asignatura */
    public function grade(){
        return $this->belongsTo(Grade::class);
    }


    /* Obtiene los profesores que estan impartiendo la asignatura */
    public function professors(){
        return $this->belongsToMany(User::class,'course_professor_subject')
                    ->withPivot('course_id');
    }

    /* Obtiene las matrículas de la asignatura */
    public function enrollments(){
        return $this->belongsToMany(Enrollment::class,'enrollment_subject');
    }

    /* Obtiene la relacion de los mensajes seleccionados de la asignatura */
    public function messages(){
        return $this->belongsToMany(Message::class, 'message_subject')
                    ->withPivot('title');
    }

    /* Obtiene las salas del chat de la asignatura */
    public function rooms()
    {
        return $this->hasMany(Room::class);
    }

}
```

Una vez creados todos los modelos para nuestra aplicación de Laravel, el directorio **./app/models** quedaría de la siguiente manera:

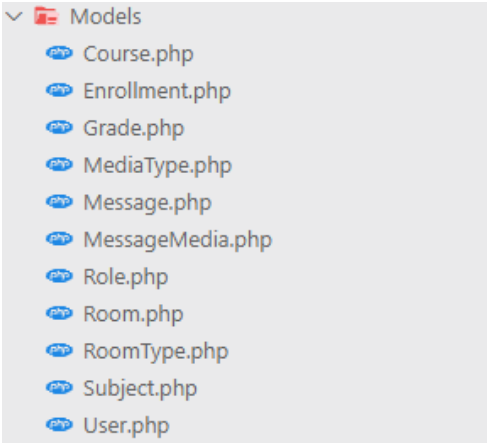


Ilustración 77. Ejemplo carpeta modelos de Laravel

En la siguiente apartado de esta sección explicaremos como definir las rutas del proyecto de Laravel, explicaremos los diferentes tipos de rutas que se han usado en el proyecto y los tipos de middleware que se han utilizado.

Rutas

Laravel incorpora un sistema de enrutamiento totalmente integrado desde el inicio del proyecto. Los archivos para definir las rutas se encuentran en el directorio **./routes** de la raíz del proyecto y se dividen en:

- **api.php**: en este archivo se definirán las rutas o endpoints de la API.
- **web.php**: en este archivo se definen las rutas web de la aplicación.
- **channels.php**: sirve para registrar los canales de retransmisión y la autorización por los que se transmitirán los eventos.
- **console.php**: en este archivo se definen los nombres de los comandos personalizados de la consola.

Para nuestro proyecto solamente haremos uso del archivo *api.php* para crear los endpoints de la API a los que se conectará nuestro Frontend y también se hará uso del archivo *channels.php* que nos ayudará a definir los canales de transmisión por los que se transmiten nuestros eventos, en el cual nos centraremos en un apartado más adelante.

Para añadir un endpoint en nuestro proyecto simplemente tenemos que ir al archivo api.php y definir una ruta de la siguiente manera:

Ilustración 78. Ejemplo de uso ruta de Laravel

Como podemos observar para crear una ruta básica en Laravel bastan con decir el tipo de petición HTTP (GET, POST, PUT, DELETE), el nombre de la ruta en la que se va a recibir la petición y por último el controlador y la acción que la van a manejar.

A continuación, se muestra el contenido de api.php donde se encuentran definidas todas las rutas disponibles en el API de nuestra aplicación.

```php
/* Rutas sin proteger */
/* Ruta de inicio de sesión */
Route::post('/signin', [AuthenticationController::class, 'signin'])
     ->middleware('throttle:5,1');
/* Rutas para restablecer la contraseña */
Route::post('/password/email', [PasswordResetController::class,
            'sendPasswordResetLink'])
     ->middleware('throttle:5,1')->name('password.email');
Route::post('/password/reset', [PasswordResetController::class, 'reset-
Password'])
     ->middleware('throttle:5,1')->name('password.reset');

/* Rutas protegidas por el middleware de sanctum */
Route::middleware(['auth:sanctum'])->group(function () {
    /* Rutas para obtener datos del usuario autentificado */
    Route::get('/user', [AuthenticationController::class, 'user']);
    /* Rutas para cerrar sesión */
    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 de usuario */
    Route::get('users/subjects', [UserController::class, 'getUserSubjects'
]);
    Route::apiResource('users', UserController::class);
    /* Rutas de grados */
    Route::apiResource('grades', GradeController::class);
    /* Rutas de asignaturas */
    Route::apiResource('subjects', SubjectController::class);
    Route::get('subjects/{subject}/users', [SubjectController::class,
               'getUsersOnSubject']);
    Route::get('subjects/{subject}/faqs', [SubjectController::class,
              'getfaqsSubject']);
    Route::get('subjects/{subject}/faqs/{item}', [SubjectController::class,
              'getfaqResponseSubject']);
    Route::delete('subjects/{subject}/faqs/{item}/delete',
                 [SubjectController::class, 'deletefaqSubject']);
    Route::get('subjects/{subject}/faqs/{item}/download',
                 [SubjectController::class, 'getMediafaq']);
    /* Rutas de cursos */
    Route::apiResource( 'courses', CourseController::class)->except(['upda-
te']);
    /* Rutas de 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, 'getRoomAva-
tar']);
    Route::get('rooms/{room}/read', [RoomController::class, 'markAsRea-
dRoom']);
    /* Rutas de mensajes  */
    Route::apiResource( 'messages', MessageController::class )
         ->except(['update']);
    Route::get('messages/{message}/download', [MessageController::class,
               'getMessageFile']);
    Route::get('messages/{message}/thumbnail', [MessageController::class,
               'getMessageFileThumbnail']);
    Route::post('messages/{message}/faqs/add', [MessageController::class,
                'addMessageToSubjectFaqs']);
});
```

Como podemos observar en el fragmento de código anterior, se encuentran definidas todas las rutas de nuestra API, pero en algunas nos encontramos que son del tipo “apiResource”, que es un método que nos proporciona Laravel para simplificar nuestro archivo de rutas, con este método Laravel internamente está creando todos los métodos HTTP (GET, PUT, POST Y DELETE) para el endpoint especificado y que son manejados con el controlador especificado como segundo parámetro de la función. Para entender mejor el funcionamiento de este método veamos el ejemplo para el endpoint del recurso grados:

```
/* Rutas recurso grados */
Route::apiResource('grades', GradeController::class);
```

Lo que realmente está haciendo Laravel al definir una ruta con este método es crear todas las rutas que se encuentran en la siguiente tabla y que llaman al controlador “GradeController” con la acción especificada en la tabla.

Tipo de petición	Ruta endpoint	Acción controlador	Utilidad
GET	/grades	index	Muestra todos los grados.
POST	/grades	store	Crear un nuevo grado.
GET	/grades/{id}	show	Ver un grado.
PUT	/grades/{id}	update	Editar un grado.
DELETE	/grades/{id}	destroy	Eliminar un grado.

Tabla 68. Relaciones controlador y tipo de petición

En el código anterior, que muestra todas las rutas de la API, también podemos ver que hay un grupo de rutas protegidas mediante un middleware llamado “auth:sanctum” que es proporcionado por Laravel Sanctum, que como ya hemos comentado anteriormente es un paquete que Laravel integra para realizar la autenticación de la aplicación web. Este middleware se encarga de filtrar las peticiones HTTP comprobando si el usuario está autentificado en la aplicación web.

En nuestro siguiente apartado explicaremos qué es un controlador y de qué manera se implementan en el framework de Laravel.

Controladores

Los controladores son uno de los componente más importante de Laravel que se encarga de agrupar la lógica de manejo de las peticiones HTTP relacionadas, es decir, un controlador se encarga de realizar la lógica de negocio necesaria para las peticiones que le llegan. Los controladores son los intermediarios entre los modelos y las vistas (arquitectura MVC), que en el caso de nuestro proyecto

enviarán una respuesta en formato JSON con la información solicitada al Frontend.

En nuestro proyecto se encuentran un total de 11 controladores:

- **AuthenticationController**. Este controlador se encarga de la autentificación del usuario en la aplicación, tanto inicio de sesión como de su cierre y además nos permite obtener la información del usuario autenticado.
- **CourseController**. Este controlador nos permite realizar todas las acciones CRUD con el recurso de Course.
- **EnrollmentController**. Contiene todos los métodos CRUD para el recurso Enrollment.
- **GradeController**. Este controlador nos permite realizar todas las acciones CRUD con el recurso de Grade.
- **ListProfessors**. Este controlador muestra la lista de profesores de la aplicación.
- **MessageController**. Este controlador nos permite realizar todas las acciones CRUD con el recurso de Message y además nos permite obtener los archivos enlazados a un mensaje o añadirlos a los mensajes favoritos de una determinada asignatura.
- **ObtainAvatar**. Devuelve la imagen de avatar de un determinado usuario.
- **PasswordResetController**. Contiene todas las acciones necesarias para restablecer la contraseña.
- **RoomController**. Contiene todos los métodos CRUD para el recurso Room y otras acciones, como añadir participantes a una sala de chat, salir de la sala de chat, etc.
- **SubjectController**. Este controlador contiene todos los métodos CRUD para el recurso Subejct y contiene acciones para obtener los mensajes destacados de una asignatura, obtener los usuarios que la han cursado o están cursándola, etc.
- **UserController**. Contiene todas los métodos CRUD para el recurso User y permite obtener las asignaturas que está cursando el estudiante o impartiendo el profesor.

Para crear los controlador usaremos la consola de comandos Artisan con el comando siguiente:

```
> php artisan make:controller API/GradeController --api
```

Notar que hemos añadido en el nombre del controlador la carpeta donde queremos que se almacene, en este caso el controlador será creado en **./app/Http/Controllers/API** con el nombre de “GradeController”, además hemos

usado una de las opciones disponibles de la consola de comandos de Artisan para crear el controlador colocando el flag " –api", esto creará un controlador con una plantilla que contiene las acciones de index, store, update, destroy y show que son usadas por las rutas con el método "apiResource" como se indica en la tabla 68.

```php
<?php

namespace App\Http\Controllers\API;

use App\Http\Controllers\Controller;
use Illuminate\Http\Request;

class GradeController extends Controller
{
    /* Display a listing of the resource. */
    public function index(){ }

    /* Store a newly created resource in storage. */
    public function store(Request $request){ }

    /* Display the specified resource. */
    public function show(Grade $grade){ }

    /* Update the specified resource in storage. */
    public function update(Request $request, Grade $grade){ }

    /* Remove the specified resource from storage. */
    public function destroy (Grade $grade){ }
}
```

A continuación, mostraremos la lógica de negocio utilizada para algunas de las acciones del controlador "GradeController".

Index

```php
public function index(Request $request)
{
    /* Query param */
    $search = $request->input('search');
    /* Eloquent filter by grade name */
    $query = Grade::where( function($query) use($search){
        $query->where('name', 'like', '%'.$search.'%');
    });
    $grades = $query->get();
    /* Response transform with GradeResource to JSON */
    return $this->handleResponse('Lista de grados mostrada con exito',
                    GradeResource::collection( $grades ) );
}
```

Este método es llamado cuando se quiere obtener todos los grados disponibles en la aplicación. Como podemos observar en el código también obtenemos un parámetro de la petición (Request) llamado "search" que usaremos para filtrar por nombre mediante el modelo Grade y Eloquent ORM, devolviendo así una respuesta JSON con los grados filtrados y formateados por medio de un

componente de Laravel llamado Resource (“GradeResource”) que ya veremos más adelante en este capítulo.

Store

```
public function store(Request $request)
{
    /* Authorization using Laravel policies */
    if (Auth::User()->cannot('create', Grade::class) ) {
        return $this->handleError('No tienes autorización para realizar
                                   esta acción.', 403);
    }
    /* Request validation */
    $validated = $request->validate([
        'name' => 'bail|required|min:5|max:90',
        'description' => 'bail|required|min:5',
    ]);

    /* Create an instance of the grade resource with Eloquent ORM */
    $grade = Grade::create($validated);
    /* Return response in JSON */
    return $this->handleResponse('Grado creado con exito', 201);
}
```

Esta acción se utiliza para guardar en la base de datos un nuevo grado, en la cual recibimos una solicitud por medio del método POST que contiene la información para crear el grado en el cuerpo de la petición (Request), antes de validar esta información se comprueba si el usuario está autorizado a realizar esta acción por medio de las Policies de Laravel (las veremos en el siguiente apartado), si el usuario está autorizado y la información es válida se creará una nueva instancia del modelo Grade en la base de datos y se enviará de vuelta una respuesta JSON confirmando la creación del grado.

Show

```
public function show(Grade $grade)
{
    /* Response transform with GradeResource to JSON */
    return $this->handleResponse('Grado mostrado con exito',
                                    new GradeResource($grade) );
}
```

Esta acción es llamada para mostrar la información de un determinado grado, para especificar el grado que queremos obtener únicamente tenemos que especificar el id en la ruta en el endpoint como se indica en la tabla 68, es decir, para obtener el grado con el número de id 2 tenemos que hacer una solicitud a **/grades/2** y esta acción del controlador “GradeController” obtendrá el grado de la base de datos por medio de la inyección de dependencias de Laravel y

devolverá una respuesta con la información transformada a JSON mediante un Resource.

Policies

Laravel además de ofrecer un servicio de autentificación integrado también proporciona una forma sencilla de autorizar al usuario a realizar acciones en un recurso determinado, para ello Laravel nos proporciona dos mecanismos: políticas y puertas, en el caso de nuestro proyecto sólo hemos usado las políticas, que son clases que agrupan la lógica de autorización en torno a un modelo o recurso en particular.

Para crear los política usaremos la consola de comandos Artisan con el comando siguiente:

```
> php artisan make:policy GradePolicy --model=Grade
```

En el comando hemos especificado el nombre de la política que para el caso de ejemplo será “GradePolicy” y le hemos añadido la opción *“—model”* para que Laravel enlace esta política con el modelo o recurso “Grade”. Este comando creará una política con el nombre “GradePolicy” en el directorio **./app/Policies** de la aplicación que contine los siguientes métodos que se corresponden con los métodos del controlador según la tabla siguiente.

Método del controlador	Método política
index	viewAny
show	view
store	create
update	update
destroy	delete

Tabla 69. Relación métodos políticas y controlador

En cada método de la política hay que devolver el valor de verdadero o falso según si el usuario en cuestión está autorizado a usar el método en el controlador como se muestra en el ejemplo siguiente para la política del modelo “Grade”.

```php
<?php

namespace App\Policies;

use App\Models\Grade;
use App\Models\User;
use Illuminate\Auth\Access\HandlesAuthorization;

class GradePolicy
{
    use HandlesAuthorization;

    /* Comprueba si el usuario es administrador y
    permite realizar todas las acciones de la política */
    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, Grade $grade)
    {
        return true;
    }

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

    public function update(User $user, Grade $grade)
    {
        return false;
    }

    public function delete(User $user, Grade $grade)
    {
        return false;
    }
}
```

Como podemos observar en esta política también se ha añadido el método “before” que permite autorizar todas los métodos para el usuario con el rol de administrador, este método tiene preferencia sobre todos los demás métodos siguientes de la política. En esta política cualquier usuario puede acceder a los métodos de viewAny (index) y view (show) pero sólo los usuarios con el rol de administrador pueden realizar los métodos de create (store), update (update) y delete (destroy) del recurso “Grade”.

Para utilizar las políticas simplemente tenemos que colocarlas al principio del método del controlador en el que se quiera utilizar de la siguiente manera:

```
public function store(Request $request)
{
    /* Authorization using Laravel policies */
    if (Auth::User()->cannot('create', Grade::class) ) {
        return $this->handleError('No tienes autorización para realizar
                                   esta acción.', 403);
    }
    /* Resto del código del controlador */
    /* ... */
}
```

Resources

Al crear una API, Laravel nos permite crear una capa de transformación que se encuentra entre el modelo de Eloquent y las respuesta JSON que se devuelve en el controlador. Por ejemplo, podemos querer mostrar ciertos atributos o mostrar siempre ciertas relaciones en la respuesta JSON que devolvemos, para realizar esto Laravel tiene unos componentes llamados Resources que permiten transformar de manera expresiva y sencilla los modelos de Eloquent o colecciones de ellos en JSON.

Para crear un Resource usaremos la consola de Artisan con el siguiente comando, que nos creará en la carpeta **./app/Http/Resources** un Resource llamado “GradeResource”:

```
> php artisan make:resource GradeResource
```

A continuación, se muestra como ejemplo el código para “GradeResource”:

```
<?php

namespace App\Http\Resources;
use Illuminate\Http\Resources\Json\JsonResource;

class GradeResource extends JsonResource
{
    /**
     * Transform the resource into an array.
    */
    public function toArray($request)
    {
        return [
            'id' => $this->id,
            'name' => $this->name,
            'description' => $this->description
        ];
    }
}
```

Esto recibirá un modelo de “Grade” y devolverá una respuesta en formato JSON con los atributos que se muestran en el return de la función.

Para usar los resources simplemente tenemos que pasarle el modelo o la colección como se muestra en el siguiente fragmento de código:

```
/* Response transform with GradeResource to JSON */
return $this->handleResponse('Grado actualizado con exito.',
                             new GradeResource($grade), 200 );
```

Eventos

Los eventos de Laravel proporcionan una implementación del patrón de observador simple, que permite suscribirse y escuchar eventos que ocurren dentro de nuestra aplicación a tiempo real.

En nuestro proyecto de Laravel los eventos se están utilizando principalmente para conseguir transmitir en tiempo real a través de Laravel websockets cualquier acción realizada en las salas de chat de la aplicación, como por ejemplo escribir un nuevo mensaje, actualizar la imagen de avatar de una sala de chat o notificar sobre mensajes no leídos.

Para ello, como se ha comentado en el capítulo de diseño de la aplicación se está haciendo uso de Laravel websockets, que es un sistema de transmisión de eventos del lado del servidor (Laravel) al lado del cliente (React) empleando un enfoque basado en Websockets utilizando la API integrada de **Pusher** [42] en Laravel.

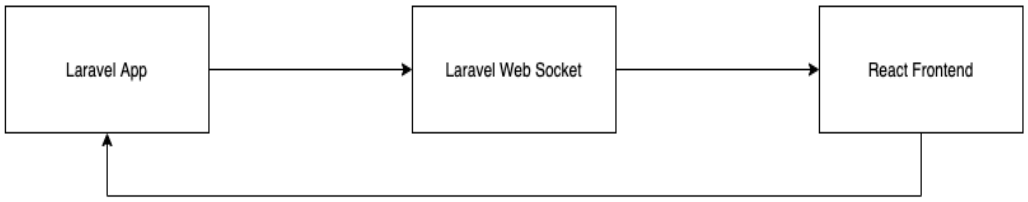


Ilustración 79. Diagrama conexión websocket frontend

Para habilitar esta función de transmisión de eventos del lado del servidor al lado del cliente lo primero que hay que hacer es instalar el paquete de **Laravel websockets** [43] como se indica en su documentación.

Una vez hemos instalado el paquete de Laravel websockets correctamente debemos configurar las variables de entorno de Pusher, que se encuentran en el archivo .env de la aplicación, estas variables serán completamente inventadas y se utilizarán para conectar el Frontend con nuestro servidor de websockets en el Backend.

```
## Configuración de pusher
PUSHER_APP_ID=54TmwEk3KVTFTjYsycdtNrYTzKvxHz-ID
PUSHER_APP_KEY=54TmwEk3KVTFTjYsycdtNrYTzKvxHz-KEY
PUSHER_APP_SECRET=54TmwEk3KVTFTjYsycdtNrYTzKvxHz-SECRET
PUSHER_APP_CLUSTER=mt1
```

Ilustración 80. Ejemplo variable de entorno para configuran Laravel websockets

Además, en este mismo archivo de las variables de entorno, debemos cambiar el valor a la variable de “BROADCAST_DRIVER” con “pusher” para que Laravel comience a utilizar Laravel websockets para retransmitir los eventos.

Para crear un evento en Laravel usaremos la consola de comandos de Artisan con el siguiente comando:

```
> php artisan make:evento ChatMessageEvent
```

Esto nos creará un nuevo evento llamado “ChatMessageEvent” en el directorio **./app/Events** de la aplicación, que nos servirá para transmitir la existencia de un nuevo mensaje en una sala de chat de la aplicación.

A continuación mostraremos como ejemplo el evento “ChatMessageEvent” de nuestra aplicación de Laravel:

```php
<?php

namespace App\Events;

use Illuminate\Broadcasting\Channel;
use Illuminate\Broadcasting\InteractsWithSockets;
use Illuminate\Broadcasting\PresenceChannel;
use Illuminate\Broadcasting\PrivateChannel;
use Illuminate\Contracts\Broadcasting\ShouldBroadcast;
use Illuminate\Foundation\Events\Dispatchable;
use Illuminate\Queue\SerializesModels;

use App\Http\Resources\MessageResource;
use App\Models\Message;

class ChatMessageEvent implements ShouldBroadcast
{
    use Dispatchable, InteractsWithSockets, SerializesModels;

    public $message;

    /* Create a new event instance. */
    public function __construct( Message $message )
    {
        $this->message = $message;
    }

    /* Get the channels the event should broadcast on. */
    public function broadcastOn()
    {
        return new PrivateChannel('chat.'.$this->message->room_id);
    }

    /* Data carried by the event. */
    public function broadcastWith()
    {
        return (new MessageResource($this->message))->resolve();
    }
}
```

Como se puede observar en el código del evento, éste recibe una instancia del mensaje que se quiere transmitir en el constructor de la clase, asimismo se observan dos métodos que Laravel utiliza internamente para transmitir el evento por el canal especificado (broadcastOn) y para transformar los datos del modelo que transportara el evento a JSON donde nuevamente se usan los Resources que nos proporciona Laravel (broadcastWith).

Al usar un canal privado para la transmisión del evento tenemos que configurar la lógica de acceso al canal en cuestión desde el archivo **./routes/channels.php** como se muestra en el siguiente fragmento de código:

```php
Broadcast::channel('chat.{room}', function ($user, Room $room) {
    return $room->users()->where('user_id', $user->id)->exists();
});
```

Una vez creado el evento y configurada la lógica de acceso al canal solamente queda disparar el evento desde cualquier acción de un controlador, para nuestro caso de ejemplo será desde el controlador “MessageController” y desde la acción de store que se utiliza para crear un nuevo mensaje para una sala de chat de la siguiente manera:

```
/*Disparamos eventos de nuevo mensaje para los que están conectados a la sala de chat */
broadcast(new ChatMessageEvent($message))->toOthers();
```

Con esto cada vez que se añada un nuevo mensaje a una sala de chat será transmitido a todos los usuarios que estén conectados a esta.

Factories

Los Factories nos permiten crear registros de prueba, ya sea para cargar nuestra base de datos con información falsa o información de prueba para crear las condiciones necesarias para probar la aplicación.

Para crear un factory usaremos la consola de comandos Artisan que nos creará un nuevo archivo en el directorio **./database/factories** con el siguiente comando:

```
> php artisan make:factory UserFactory
```

A continuación mostraremos un ejemplo del “UserFactory” usado en nuestra aplicación que se encargará de proporcionar información falsa mediante la biblioteca de **Faker** en PHP para crear usuarios de prueba, además en este ejemplo hemos creado dos métodos para poder seleccionar el rol del usuario.

```
<?php

namespace Database\Factories;

use Illuminate\Database\Eloquent\Factories\Factory;
use Illuminate\Support\Str;

class UserFactory extends Factory
{
    /* Define the model's default state. */
    public function definition()
    {
        $name = $this->faker->name();
        $name_parts = explode(" ", $name);
        return [
            'name' => $name,
            'email' => $this->faker->unique()->safeEmail(),
            'password' => '$2y$10$92IXUNpkjO0rOQ5byMi.C/.og/at2.uheWG/igi',
            'phone' => $this->faker->phoneNumber(),
```

```
            'address' => $this->faker->address(),
            'image' => $this->faker->randomElement([null, "1.jpg","2.jpg" ]),
        ];
    }

    public function student(){
        return $this->state( function( array $attributes ){
            return [
                'role_id' => 1
            ];
        });
    }

    public function professor(){
        return $this->state( function( array $attributes ){
            return [
                'role_id' => 2
            ];
        });
    }

    public function administrator(){
        return $this->state( function( array $attributes ){
            return [
                'role_id' => 3
            ];
        });
    }
}
```

Los factories, normalmente, serán utilizados junto a los seeders, que serán explicados en el siguiente apartado. Se usarán de la siguiente manera:

```
/* Creamos 140 usuarios con el rol de estudiante */
User::factory()->count(140)->student()->create();
```

Seeds

Los seeders son un componentes del marco de trabajo de Laravel que sirven para inicializar las tablas con datos.

Para crear un seeder utilizaremos la consola de comandos Artisan con el siguiente comando que creará un nuevo archivo llamado “UserSeeder” en **./database/seeders**:

```
> php artisan make:seeder UserSeeder
```

A continuación se muestra como ejemplo el seeder “UserSeeder” que iniciará la base de datos con usuarios de prueba.

```php
<?php

namespace Database\Seeders;

use Illuminate\Database\Seeder;
use App\Models\User;

class UserSeeder extends Seeder
{
    /* Run the database seeds. */
    public function run()
    {
        /* Creamos 140 usuarios con el rol de estudiante */
        User::factory()->count(140)->student()->create();
        /* Creamos 20 usuarios con el rol de profesor */
        User::factory()->count(20)->professor()->create();
        /* Creamos 3 usuarios con el rol de administrador */
        User::factory()->count(3)->administrator()->create();
    }
}
```

Para ejecutar el seeder llamado “UserSeeder” y poblar nuestra base de datos con los usuarios de prueba falsos creados mediante los factories ejecutamos el siguiente comando:

```
> php artisan db:seed  --class=UserSeeder
```

En la siguiente sección veremos la forma en la que se implementa la aplicación en el lado del Frontend, donde tendremos ejemplos de código de cómo se ha creado la interfaz de usuario usando React, TailwindCSS y las diferentes librerías de terceros usadas para el desarrollo del proyecto.

5.2. IMPLEMENTACIÓN FRONTEND

Para el desarrollo del Frontend de la aplicación se ha utilizado **React** [44], que es una librería JavaScript para el desarrollo de interfaces de usuario dinámicas y aplicaciones de una sola página (SPA).

Además de React se han utilizado otras librerias de terceros como complemento para ahorrar tiempo en el desarrollo de la aplicación como ya se comentó en capítulos anteriores de la memoria, asimismo se ha utilizado TailwindCSS que es un framework de CSS para dotar de estilo de una forma fácil y rápida al frontend de la aplicación mediante clases predefinidas, se ha decidido usar este framework de CSS ya que se integra de una forma fácil y rápida con las aplicaciones desarrolladas con la librería de React.js.

Para crear una aplicación en React actualmente existen muchos métodos, pero nosotros utilizaremos el método que se indica en el tutorial de la página oficial de React.js para crear aplicaciones de una sola página usando la herramienta de Create React App, que consiste en una interfaz de línea de comandos que permite crear una aplicación de React rápidamente y sin necesidad de configuraciones adicionales.

A continuación expondremos los pasos necesarios para instalar la herramienta de Create React App en nuestro entorno de desarrollo y crearemos una aplicación de React con TailwindCSS.

El primer paso a seguir sería instalar en nuestro entorno de desarrollo la versión más reciente de NodeJS y el administrador de paquetes npm desde la página oficial de NodeJS. [45]

Ilustración 81. Logo NodeJS y NPM

Una vez instalado NodeJS y NPM desde la línea de comandos de nuestra máquina ejecutamos el siguiente comando, que instalará la herramienta de Create React App:

```
> npm install -g create-react-app
```

Con el entorno de Create React App instalado solo queda ejecutar el siguiente comando en el directorio donde queramos almacenar el proyecto de React:

```
> create-react-app frontend
```

Esto creará una carpeta llamada frontend que contendrá los archivos necesarios para el desarrollo de una aplicación con React.

Una vez tenemos la aplicación de React creada entramos en el directorio del proyecto y ejecutamos los siguientes comandos desde la consola de comandos de nuestra máquina para instalar TailwindCSS:

```
npm install -D tailwindcss postcss autoprefixer
```

```
npx tailwindcss init -p
```

Una vez instalado todo, solo quedaría realizar la configuración de TailwindCSS para React, que podemos encontrar en la documentación oficial de la página de **TailwindCSS** [46]. Para ello, como se indica en la documentación, editamos el

archivo “tailwind.config.js” que se encuentra en la raíz de nuestro proyecto de React de la siguiente forma:

```
module.exports = {
    content: [
      "./src/**/*.{js,jsx,ts,tsx}",
    ],
    theme: {
      extend: {},
    },
    plugins: [],
}
```

Por último añadimos estas directivas al archivo de estilo globales de la aplicación de React que se encuentra en **./src/index.css**:

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Con todo esto ya tendríamos una aplicación de React creada y configurada con el framework de TailwindCSS que podríamos ejecutarla de una forma muy sencilla en nuestro entorno de desarrollo simplemente ejecutando en la raíz del proyecto el siguiente comando:

```
> npm start
```

Este comando hace que el entorno de Create React App construya y ejecute la aplicación en modo desarrollo y sea visible en el navegador en la dirección **http://localhost:3000/** en la cual podemos ver a tiempo real todos los cambios que vamos realizando en el proyecto, por lo que es muy útil para realizar en desarrollo de la aplicación.

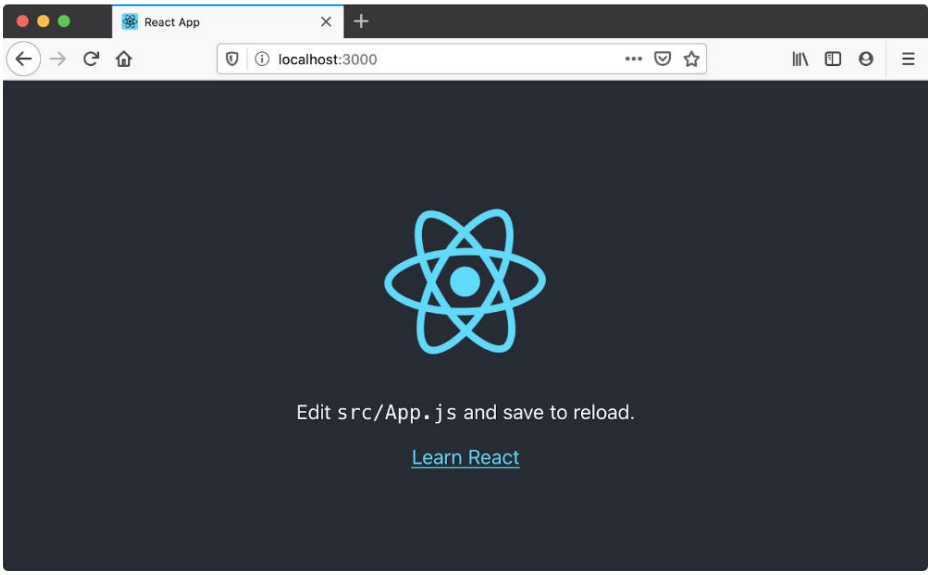


Ilustración 82. Página de inicio proyecto React

En el punto siguiente veremos cómo hemos estructurado la aplicación de React y explicaremos para que serán utilizados cada uno de los directorios.

5.2.1. ESTRUCTURA DE UNA APLICACIÓN REACT

En esta sección se explicará la estructura de carpetas elegida para la creación de nuestro proyecto React, describiendo los directorios y archivos más importantes.

Antes de exponer la estructura de directorios de nuestra aplicación he de comentar que React nos da la libertad de organizar nuestra aplicación con la estructura que deseemos, por lo que cada proyecto puede tener una estructura de directorios diferentes dependiendo del programador. A continuación mostramos la estructura elegida para crear nuestro proyecto.

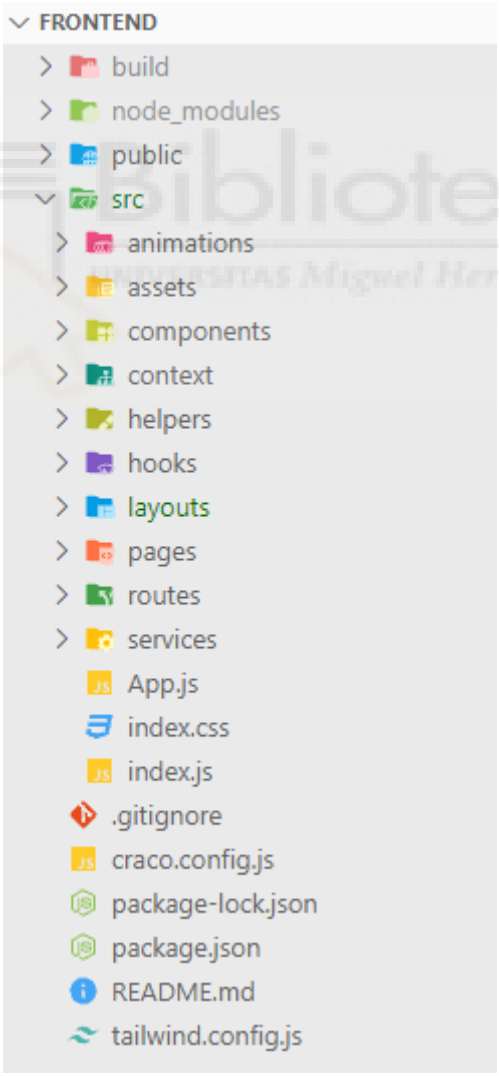


Ilustración 83. Ejemplo estructura carpetas React

- **/build:** en este directorio se almacenará la construcción de la aplicación web para producción mediante el comando “npm build”.

- **/node_modules:** este directorio almacena las diferentes librerías y módulos que la aplicación podría utilizar.

- **/public:** en esta carpeta se almacenan los archivos estáticos de la aplicación y contiene el archivo “index.html”.

- **/src:** este directorio contiene el código fuente de nuestra aplicación. En esta carpeta entre otros se encuentran los archivos de “index.js” e “index.css” que son los puntos de entrada de nuestra aplicación de React.

    - **src/animations:** este directorio contiene animaciones creadas en css.
    - **src/assets:** en esta carpeta almacenaremos los recursos de video, imágenes y sonidos usados por la aplicación.
    - **src/components:** en este directorio se encuentra el código de los diferentes componentes de React que forman la interfaz de usuario.
    - **src/context:** en esta carpeta se encuentran los archivos de context creados para nuestra aplicación de React.
    - **src/helpers:** en este directorio almacenamos funciones de utilidad que serán usadas por diferentes componentes.
    - **src/hooks:** esta carpeta contiene nuestros hooks personalizados creados para la aplicación.
    - **src/layouts:** en este directorio se encuentran los diferentes layouts que usaremos en la aplicación.
    - **src/pages:** en esta carpeta se encuentran las diferentes páginas que nuestra aplicación de React.
    - **src/routes:** en este directorio se encuentran las definiciones de las rutas para los diferentes tipos de usuario de nuestra aplicación.
    - **src/services:** en esta carpeta se encuentran los archivos que se encargan de realizar solicitudes a los endpoints del Backend.
    - **src/index.css:** este archivo contiene los estilos globales de nuestra aplicación.
    - **src/index.js:** este archivo es el punto de entrada para nuestra aplicación**.**
    - **src/app.js:** este archivo se trata del componente raíz de nuestra aplicación de React.

- **.gitignore:** es un archivo de texto que contiene los archivos o carpetas a ignorar del proyecto para el sistema de versiones Git.

- **tailwind.config.js:** este archivo contiene la configuración del framework TailwindCSS.

- **package.json:** contiene información sobre nuestro proyecto, que Node.js y npm usarán para instalar las librerías y módulos necesarios en nuestra aplicación.

5.2.2. DESARROLLO CON REACT

En esta sección detallaremos los principios básicos de desarrollo para una aplicación de React para así conseguir una idea general de las características y técnicas utilizadas durante la implementación de este proyecto.

Como ya hemos comentado en la memoria, React permite crear interfaces de usuario complejas y dinámicas con una sintaxis declarativa, eficiente y flexible mediante pequeñas piezas de códigos llamadas “componentes”. Estos componentes permiten separar la interfaz de usuario en piezas independientes y reutilizables que nos ayudan a pensar en cada pieza de forma aislada.

Para comenzar a entender cómo funciona React, analicemos un componente sencillo que hemos realizado para mostrar el logo de la aplicación en el interfaz de usuario:

```
import React from 'react'
import { GiGraduateCap } from 'react-icons/gi';

export default function Logo({sizeIcon, className}){
    return(
        <span className={`inline-flex items-center gap-2 ${className}`}>
            <GiGraduateCap size={sizeIcon}/>
            MYCLASS
        </span>
    );
}
```

Como podemos ver en el código anterior, después de realizar las importaciones de las librerías necesarias tenemos una función que se exporta por defecto, esta función representa un componente en React y solamente acepta un argumento como objeto, el cual se denomina “**Props**”, que contienen los datos necesarios para renderizar el componente y permite pasar datos de componentes padres a los componentes hijos.

Si nos centramos en el interior de la función podemos ver que devuelve una especie de código HTML que en realidad es JSX que es una extensión de la sintaxis de JavaScript que nos permite de una manera amigable crear y mantener el HTML junto con JavaScript.

Para renderizar este componente simplemente tenemos que importarlo como se indica en el código siguiente y colocarlo como si fuera una etiqueta *HTML*, a la cual le pasamos como propiedades de la etiqueta los parámetros indicados anteriormente, que para el caso de nuestro ejemplo son el tamaño del icono y las clases css adicionales.

```
import React from 'react';
import Logo from '../components/shared/Logo';
/* ... imports */

export default function Login(){

    return(
        <main className="min-h-screen flex items-stretch">
            {/* ... */}
            <Logo sizeIcon={48} className="text-4xl" />
            {/* ... */}
        </main>
    );
}
```

Como podemos ver en este pequeño ejemplo, React se basa en la creación de interfaces complejas por medio del uso de pequeños componentes que van formando componentes más complejos, donde estos componentes pueden ser reutilizados en muchas parte de la aplicación gracias a la versatilidad que se ofrece mediante las “Props” de los componentes.

Por otro lado, los componentes de React permiten tener un estado interno que se usa para contener datos o información del componente que pueda variar en el tiempo, cuando este estado cambia se vuelve a renderizar el componente mantendo así siempre la vista de este actualizada para el usuario.

Para implementar los estados en un componente se usan los “**Hooks**” de React, que son una API de la librería que permite usar los estados y otras características integradas como el manejo del ciclo de vida de los componentes. Para usar los estados en un componente de React haremos uso del hook **useState** como se muestra a continuación.

```
/* Componente que muestra el avatar de un usuario, si no tiene imagen muestra
sus iniciales */

/* Imports … */
export default function Avatar({username, image, className}){

    /* Definición de estados del componente */
    const [src, setSrc] = useState(null)
    const [loading, setLoading] = useState(false)

    /* Llamada a la API mediante el hook useEffect */
    useEffect(() => {
        if(image){
            setLoading(true)
            UsersService.getAvatar(image)
            .then( (data) => {
                var reader = new window.FileReader();
                reader.readAsDataURL(data);
                reader.onload = function () {
                    setSrc(reader.result);
                }
            })
            .catch( (error) => console.error(error.response.data.error) )
            .finally( () => setLoading(false) );
        }
    }, [image])

    /* Obtenemos las inciales del nombre de usuario */
    const initials = username.split(' ')
                             .reduce((acc, subname) => acc + subname[0], '')
    /* Clases por defecto */
    const classNames = `rounded-full bg-gray-400 border border-white
                        font-medium uppercase flex justify-center
                        items-center ${className}`
    /* Color del background según iniciales */
    const colorBg = ColorsLetter(username);

    return(
        loading ?
        <div className={`${classNames} animate-pulse`}  />
        :
        <div className={classNames}
              style={{backgroundColor: !src ?  colorBg : 'white'}} >
                { src && <img src={src} className="rounded-full object-cover"
                              style={{height: 'inherit', width: 'inherit'}}
                              alt={`Imagen de perfil de ${username}`} />
                }
                { !src && initials }
        </div>
    );
}
```

Como vemos en el ejemplo anterior de un componente que muestra el avatar de usuario, el hook useState es una función que le pasamos como argumento el valor del estado inicial y nos devuelve una pareja de valores: el estado actual y una función para actualizar el estado. Es importante señalar que el estado sólo puede cambiar a través de esta función para que se realice un nuevo render del componente.

```
/* Definición de estados del componente */
const [loading, setLoading] = useState(false)
```

Para seguir con la explicación del componente anterior explicaremos otro de los hooks más importantes en React, el hook **useEffect**. Este hook acepta como argumento una función y esta función se ejecuta la primera vez que se renderiza el componente o cuando cambia una de las variables indicadas al final del hook. Este hook es muy utilizado, ya que normalmente se usa para realizar las solicitudes a la API y obtener los datos iniciales que se tienen que mostrar en el componente.

```
    /* Llamada a la API mediante el hook useEffect */
    useEffect(() => {
        if(image){
            setLoading(true)
            UsersService.getAvatar(image)
            .then( (data) => {
                var reader = new window.FileReader();
                reader.readAsDataURL(data);
                reader.onload = function () {
                    setSrc(reader.result);
                }
            })
            .catch( (error) => console.error(error.response.data.error) )
            .finally( () => setLoading(false) );
        }
    }, [image])
```

Como se puede observar en el fragmento de código anterior donde se muestra el uso del hook useEffect en el componente “Avatar”, este se encarga que al hacer el primer renderizado del componente si el usuario tiene imagen de avatar se haga una solicitud al servidor mediante el servicio “UserService”, que una vez resuelta actualizará los estados del componente que harán que se vuelva a renderizar y se actualice la vista con la imagen de usuario obtenida desde la API.

Para hacer la solicitud a la API se ha hecho uso de un servicio, que es un objeto que ofrece un conjunto de métodos para comunicarse con el servicio API para un recurso determinado, en nuestro proyecto estos objetos están almacenados en el directorio **src/services**. La ventaja que nos otorga utilizar archivos de servicio es que nos permite tener toda la lógica de solicitud de peticiones de un recurso en un mismo archivo centralizado que permite su reutilización en cualquier componente de la aplicación.

A continuación se muestra como ejemplo uno de los servicios creados en el proyecto en el que se utiliza la librería de Axios para facilitar hacer llamadas a la API.

```
import axios from "axios";

/* Endpoint */
const ENDPOINT = "http://15.xx7.xx.107/api/users"
/* Headers de la aplicación */
const headers = { 'Content-Type': 'application/json'}

const getAvatar = async(image) => {
    const {data} = await axios.get(`${ENDPOINT}/get_avatar/${image}`,
                                   {responseType: "blob"});
    return data
}

const getUsers = async(params) => {
    const {data} = await axios.get(`${ENDPOINT}`,{headers, params});
    return data;
}

const createUser = async(payload) => {
    const {data} = await axios.post(`${ENDPOINT}`, payload ,{headers});
    return data;
}

const getUser = async(id) => {
    const {data} = await axios.get(`${ENDPOINT}/${id}`);
    return data
}

/* Resto de métodos para el recurso usuario ... */

const defaultExports = { getAvatar, getUsers, createUser,
                         getUser, deleteUser, updateUser, getSubjects }
export default defaultExports
```

En cuanto a los demás hooks de la librería de React disponibles hay que destacar el uso en el proyecto de los hooks **useContext** y **useCallback**.

El hook **useContext** es un método de React que nos permite crear datos globales para nuestros componentes sin la necesidad de pasar esta información manualmente, mediante las Props, a través de todos los niveles de componente hasta llegar al que realmente lo necesita.

A continuación se mostrará como ejemplo la implementación de “SessionContext”, que es un context que almacena los datos de sesión del usuario para que sean accesible de una forma sencilla para los componentes de nuestra apliación.

```
import React, { createContext, useState } from "react";
/* Creacion del context */
const Context = createContext({});
/* Creamos un componente para que sea proveedor del contexto */
export function SessionContextProvider({ children }) {
  const [authToken, setAuthToken] = useState(() =>
    JSON.parse(window.localStorage.getItem("auth_token"))
  );
  const [user, setUser] = useState(null);

  return (
    <Context.Provider value={{ authToken, setAuthToken, user, setUser }}>
      {children}
    </Context.Provider>
  );
}

export default Context;
```

Como podemos ver en el código de ejemplo anterior, lo primero que hacemos es crear un contexto con los datos iniciales, en nuestro caso un objeto vacío como se muestra a continuación.

```
/* Creacion del context */
const Context = createContext({});
```

Una vez tenemos creado el contexto nos creamos un componente en React en el mismo archivo que actuará como proveedor de este contexto, es decir, crearemos un componente que permitirá que todos sus componentes hijos tengan acceso a los datos almacenados en el context.

```
/* Creamos un componente para que sea proveedor del contexto */
export function SessionContextProvider({ children }) {
  const [authToken, setAuthToken] = useState(() =>
    JSON.parse(window.localStorage.getItem("auth_token"))
  );
  const [user, setUser] = useState(null);

  return (
    <Context.Provider value={{ authToken, setAuthToken, user, setUser }}>
      {children}
    </Context.Provider>
  );
}
```

Dentro de este componente tenemos una serie de estados, que serán almacenados en el context para que todos los componentes hijos tengan acceso a ellos, a su vez este componente devuelve el provider del context en el cual se especifican los datos a almacenar en el context mediante la propiedad *value*.

Este contexto lo usaremos en el componente raíz de la aplicación para que todos los componentes hijos puedan tener acceso a él, para ello simplemente tenemos que utilizarlo como cualquier componente de React.

```
/* imports ... */
function App() {
  return (
    <>
      <SessionContextProvider>
        /* componentes hijos */
      </SessionContextProvider>
    </>
  );
}
export default App;
```

Por último para acceder desde un componente hijo a los estados almacenados en el context se haría de la siguiente forma:

```
/* uso de los estados almacenado en el context SessionContext */
const {authToken ,setAuthToken, user, setUser} = useContext(SessionContext)
```

Por otra parte el hook **useCallback** se encarga de memorizar las funciones en memoria y de que no se destruyan y se vuelvan a crear cada vez que se renderiza el componente que la contiene. Este hook resulta muy útil cuando se transfieren funciones a componentes hijos mediante Props ya que permite que la función siempre tenga la misma referencia en memoria.

Para crear un hook useCallback es necesario pasarle como argumento la función que se va a almacenar en memoria, como se puede ver en ejemplo siguiente:

```
/* Definicion del hook useCallback */
const logout = useCallback(() => {
    window.localStorage.removeItem('auth_token')
    setUser(null)
    setAuthToken(null)
},[])
```

Por otro lado, a parte de los hooks disponibles en la librería, React nos permite crear nuestros propios hooks personalizados, que son un mecanismo para reutilizar lógica de estado, en el cual cada llamada al hook dispone de un estado y efecto completamente independiente entre sí. En nuestro proyecto estos hooks se encuentran en **src/hooks**.

A continuación mostraremos como ejemplo uno de los hooks personalizados creados en el proyecto llamado “useSession” que se encarga de ofrecer la lógica de estado de la sesión de usuario a cualquier componente del proyecto.

```
/* imports ... */

const useSession = () => {
    const {authToken ,setAuthToken, user, setUser} = useContext(SessionCon-
text)
    const [isLoading, setIsLoading] = useState(false)
    const [hasError, setHasError] = useState("")

    const login = useCallback((email, password) => {
        setHasError("")
        setIsLoading(true)
        AuthService.login(email, password).then( resp =>{
                setIsLoading(false)
                window.localStorage.setItem('auth_token',
                                    JSON.stringify(resp.data.auth_token) );
                setAuthToken(resp.data.auth_token);
        }).catch( (err) => {
                setIsLoading(false)
                setHasError(err.response.data.message)
            }
        );
    },[setAuthToken])

    const logout = useCallback(() => {
        window.localStorage.removeItem('auth_token')
        setUser(null)
        setAuthToken(null)
    },[setUser,setAuthToken])

    const getUser = useCallback(
      () => {
        AuthService.getUser().then( resp => setUser(resp.data) )
        .catch( err => console.error( err.response.data.message ) )
    },[setUser])

    return { isLoggued:Boolean(authToken), user, isLoading, hasError,
             setUser, login, logout, getUser }
}
export default useSession;
```

Para crear un hook personalizado es necesario definir el nombre de la función con el prefijo “use” para que React lo identifique como un hook personalizado, tal y como se muestra en el ejemplo anterior. Dentro de esta función podemos utilizar cualquiera de los hooks disponibles en la librería de React, para el caso del ejemplo, lo que queremos es centralizar toda la lógica del estado de sesión del usuario, por lo tanto hacemos uso del context creado anteriormente que almacenaba los datos de la sesión de usuario. Además, se han creado más estados que gestionan las solicitudes a la API y se han añadido los métodos para hacer “login”, “logout” y “getUser”.

Para usar un hook personalizado simplemente tenemos que llamarlo dentro de cualquier componente como se hacía con un hook de la librería de react, tal y como se muestra en el código siguiente.

```jsx
/* imports ... */
export default function Home() {
    /* Uso del hook personalizado useSession */
  const { user, logout } = useSession();

  return (
    <div className="h-full bg-gray-100 items-center py-6 px-6 relative">
      <ProfileCard user={user} onLogout={logout} />
      <MenuHomeList />
    </div>
  );
}
```

Una vez tenemos claro cómo se lleva a cabo el desarrollo de los componentes de React y el uso de los hooks de la librería, veremos cómo se realizan los formularios. Esta parte es muy importante en este proyecto, ya que se encarga de realizar peticiones a través del cliente a nuestra API.

Para la creación de los formularios en nuestro proyecto hemos hecho uso de dos librerías muy populares entre los desarrolladores que permiten realizar la gestión de formularios y validación de una forma muy sencilla. Estas librerías son **Yup y Formik**. Por su parte, Yup nos permite crear esquemas de validación para los campos definidos en el formulario; mientras que Formik se encarga de realizar un seguimiento de los valores, errores, campos visitados, validar y manejar el envío del formulario. [47] [48]

Ilustración 84. Logo React, Formik y Yup

Para implementar estas dos librerías veamos el siguiente código, que muestra el componente encargado de enviar los mensajes en el chat de la aplicación a través de un formulario manejado con Formik y Yup.

```
/* imports ... */

/* Validacion con yup */
const MessageSchemaValidation = Yup.object().shape(
  /* Lógica de validación de yup … */
);

export default function ChatInput({ file, setFile, closeChatAnswerPreview,
answer,}) {
  const { chatSelected, setChatSelected,
          setMessages, setChatList } = useChat();

  /* Handle form with formik */
  const formik = useFormik({
   /* Formik config … */
  });

  /* Handle emoji picker … */

  return (
    <div className="flex items-center gap-2 sm:gap-4 md:gap-6 …">

      {/* EmojiPicker */}
      <div className="relative" ref={EmojiPickerRef}>
      </div>
      {/* Formulario */}
      <form onSubmit={formik.handleSubmit} className="flex flex-1 …" >
        <input
          name="content"
          id="inputChat"
          value={formik.values.content}
          className="flex-1 h-8 w-32 focus:outline-none border …"
          onChange={formik.handleChange}
          autoComplete="off"
          placeholder={
            answer ? "Escribe la respuesta al mensaje" : "Escribe tu mensaje"
          }
        />
        <Button type="submit">
        </Button>
      </form>
    </div>
  );
}
```

Como podemos ver en el componente, éste empieza con la implementación del esquema de validación para las entradas del formulario a través de la librería Yup.

```
/* Validacion con yup */
const MessageSchemaValidation = Yup.object().shape(
  {
    content: Yup.string()
      .max(280, "El mensaje no puede tener más de 280 carácteres.")
      .when("file_attach", {
        is: (file_attach) => !file_attach,
        then: Yup.string().required("No puedes enviar un mensaje vacío."),
      }),
    file_attach: Yup.mixed().when("content", {
      is: (content) => !content || content.trim().length === 0,
      then: Yup.string().required("No puedes enviar un mensaje vacío."),
    }),
  },
  ["file_attach", "content"]
);
```

Esta librería nos permite crear una serie de condiciones que tienen que cumplir nuestras entradas para poder enviar el formulario. Este esquema es pasado dentro del objecto de configuración del hook **useFormik,** que pertenece a la librería de Formik. A través de este hook se maneja toda la lógica del formulario como se muestra en el código siguiente:

```
  /* Handle form with formik */
  const formik = useFormik({
    initialValues: {
      content: "",
      file_attach: file,
      answer_for_id: answer?.id,
      room_id: chatSelected.id,
    },
    validateOnMount: true,
    validationSchema: MessageSchemaValidation,
    enableReinitialize: true,
    onSubmit: (values, { resetForm }) => {
      ChatsService.createMessageChat(values)
        .then((resp) => {
          /* Lógica para petición exitosa …*/
        })
        .catch((err) => {
          /* Lógica para petición fallida …*/
        });
    },
  });
```

Como podemos ver en el hook useFormik le pasamos un objeto de configuración inicial como único argumento en el que indicamos los valores iniciales del formulario, el esquema de validación y la función para el envío de los valores a la API mediante el método onSubmit.

Nótese que es importante que los campos del esquema de validación, los atributos de los valores iniciales del hook useFormik y los campos HTML de formulario dispongan del mismo nombre, ya que Formik utilizará estos nombres para el manejo interno del formulario. Además, para completar la configuración de Formik necesitamos que el formulario y los campos HTML usen los métodos disponibles en Formik para realizar su manejo, como se muestra a continuación:

```
/* Formulario */
<form onSubmit={formik.handleSubmit} className="flex flex-1 …" >
    <input
    name="content"
    id="inputChat"
    value={formik.values.content}
    className="flex-1 h-8 w-32 focus:outline-none border …"
    onChange={formik.handleChange}
    autoComplete="off"
    placeholder={
        answer ? "Escribe la respuesta al mensaje" : "Escribe tu mensaje"
    }
    />
    <Button type="submit">
    </Button>
</form>
```

Con estos simples pasos ya tendríamos configurado un formulario con validación en el lado del cliente en React.

Por otro lado en el presente proyecto también se ha hecho uso de una librería esencial para su desarrollo llamada **React Router** [49] que nos ofrece un sistema de enrutamiento dinámico mediante una colección de componentes de navegación. Como ejemplo de uso de la librería vamos a mostrar uno de los componentes en los que se realiza el enrutamiento de la aplicación.

```
/* imports */
function App() {
    return (
      <>
        <SessionContextProvider>
          <React.Suspense fallback={<Loading />}>
            <Router>
              <Switch>
                <PublicRoute path="/login" restricted={true} component={Login}
/>
                <PrivateRoute
                  path="/dashboard" // ruta que renderiza en el componente
                  component={() => (
                   //componente que renderiza cuando la ruta coincide
                    <WithAxios>
                      <SidebarContextProvider>
                        <VolumeContextProvider>
                          <Layout /> // componente que renderiza el layout
                        </VolumeContextProvider>
                      </SidebarContextProvider>
                    </WithAxios>
                  )}
                />
                <Redirect exact path="/" to="/login" />
                <PublicRoute
                  path="/reset-password"
                  restricted={true}
                  component={ResetPassword}
                />
                <PublicRoute path="*" component={Page404} />
              </Switch>
            </Router>
          </React.Suspense>
        </SessionContextProvider>
      </>
    );
  }
  export default App;
```

Como podemos ver en el componente anterior estamos haciendo uso de algunos de los componentes de navegación más importantes de la librería en los que destacan los componentes Router y Switch, además hemos hecho una modificación en el componente Route de la librería de React Router mediante unos componente de orden superior (HOC) llamados PublicRoute y PrivateRoute.

El componente Router se encarga de envolver nuestra aplicación dándonos acceso a la API del historial de HTML5 para mantener los componentes de nuestra interfaz de usuario sincronizados con la URL del navegador, mientras

que el componente Switch se encarga de renderizar el primer componente hijo que coincida con la URL indicada.

Respecto a los componentes PrivateRoute y PublicRoute, son dos componentes de orden superior del componente Route de la librería de React Router que tienen lógica añadida para actuar de middleware de autentificación. Estos componentes heredan todas las propiedades del componente Route de la librería, por ello podemos usar las propiedades “path” para indicar la url en la que se renderiza el componente señalado en la propiedad “component”.

A continuación vamos a mostrar el ejemplo de la lógica que contiene nuestro componente de orden superior “PrivateRoute”.

```
/* imports ... */
const PrivateRoute = ({component: Component, children, role, ...rest}) => {
    const { isLoggued, user } = useSession();

    return (

        /* Mostrar componente si esta autentificado y autorizado
           de otra manera redirigir al login o mostrar que no tiene
           permisos para visualizar el contenido */
        <Route {...rest} render={props => (
            !isLoggued ? <Redirect to="/login" />
                    : (role && user.role !== role) ?  <Page403 />
                                                   : <Component {...props} />
        )} />
    );
};

export default PrivateRoute;
```

Para finalizar este capítulo vamos a explicar cómo se ha realizado la implementación de la librería de **Laravel Echo** y **Pusher** que son usadas para conectarse al servidor websockets disponible en el Backend y recibir los eventos emitidos por Laravel. [50]

Como primer paso hemos creado un componente (HOC) en el que configuramos la conexión al servidor websockets para que todos los componentes hijos tenga la conexión disponible.

```
/* imports … */
const WithWebsocket = ({ children }) => {
  window.Pusher = require("pusher-js");
  window.Echo = new Echo({
    broadcaster: "pusher",
    authEndpoint: "http://15.2xx.8x.1xx/api/broadcasting/auth",
    auth: {
      headers: {
        Authorization:
          "Bearer " + JSON.parse(window.localStorage.getItem("auth_token")),
      },
    },
    key: "54TmwEk3KVTFT83jYsycdtNrYTzKvxHzKEY",
    cluster: "mt1",
    wsHost: "15.237.84.107",
    wsPort: 6001,
    forceTLS: false,
    disableStats: true,
    encrypted: false,
    enabledTransports: ["ws"],
  });

  axios.interceptors.request.use((config) => {
    config.headers["X-Socket-Id"] = window.Echo.socketId() || null;
    return config;
  });

  return children;
};

export default WithWebsocket;
```

Para usar este componente simplemente tenemos que utilizarlo como una etiqueta HTML, para el caso del presente proyecto se ha utilizado en el Layout de la aplicación para ofrecer a todos los componentes de la aplicación la opción de usar el servicio websockets una vez el usuario ha iniciado sesión.

```
/* Este componente se encuentra en el layout para dar la capacidad de conexión
websockets a toda la app una vez se haya iniciado sesión */
<WithWebsocket>
 /* Componentes hijos */
</WithWebsocket>
```

Una vez tenemos implementada la conexión websocket, simplemente tenemos que ir a cualquiera de los componentes hijo y suscribirnos al canal y evento en cuestión como se indica en el fragmento de código siguiente.

```
/* Componente MessageBox
 Se encuentra en un useEffect para que solo se cargue al iniciar la aplicación */
useEffect(() => {
    const channelChat = window.Echo.channel(
      `private-chat.${chatSelectedRef.current.id}`
    );

    channelChat.listen("ChatMessageEvent", (message) => {
      /* Lógica cuando sucede el evento */
    });

    /* Nos salimos del canal cuando se destruya el componente */
    return () => {
      window.Echo.leave(`private-chat.${chatSelectedRef.current.id}`);
    };
  }, []);
```

En el código anterior se muestra cómo se hace la conexión al websockets del backend para el componente que muestra los mensajes de una sala de chat en la aplicación mediante la librería echo, con la cual nos hemos subscrito al canal de la sala de chat y estamos a la espera de que se realice un evento “ChatMessageEvent” en Laravel para que se ejecute la lógica de la función “Listen” que añadiría a tiempo real el mensaje recibido en la interfaz de usuario.

Antes de finalizar este capítulo me gustaría añadir el uso de la librería **React Infinite Scroll** [51] en el proyecto para la caja de mensajes de una sala de chat y otros componentes, esta librería nos ofrece un componente llamado “InfiniteScroll” que nos permite realizar una paginación de los mensajes simplemente usando el scroll hacia arriba. A continuación se muestra cómo usar esta librería en un componente de React.

```
<InfiniteScroll
    dataLength={messages.length}  /* cantidad de mensajes en la lista */
    next={() => loadMore()} /* Función que carga más mensajes desde la API */
    inverse={true} /* Para que carguen los mensajes desde la parte superior */
    hasMore={paginator.next} /* Indicamos si hay más mensajes que cargar */
    loader={
    <h4 className="text-center text-indigo-600 font-roboto">
        Cargando...
    </h4>
    }
    scrollableTarget="scrollableDiv">
       <ChatMessageList containerRef={containerRef}
        messages={messages}
        chat={chatSelected}
        setAnswer={answerMessage}
        setMessageSelected={setMessageSelected}/>
</InfiniteScroll>
```

En el siguiente capítulo se muestran los resultados del proyecto de este TFG, en el que se han usado las bases explicadas durante las anteriores secciones para realizar la de aplicación con la librería de React y el framework de Laravel.

6. RESULTADOS

En este capítulo se expondrá el resultado final de la aplicación web, donde se mostrará la interfaz y las diferentes funcionalidades que tiene cada una de las pantallas.

6.1. VISTAS PARA USUARIO NO AUTENTICADO

En esta sección se mostrarán las vistas o páginas disponibles para un usuario sin autentificar en la aplicación.

6.1.1. PÁGINA DE INICIO DE SESIÓN

Esta será la vista inicial que se encuentre el usuario al acceder a la URL de la aplicación, en ella se encuentra el formulario de inicio de sesión y una opción de recuperar la contraseña introduciendo el correo electrónico del usuario.

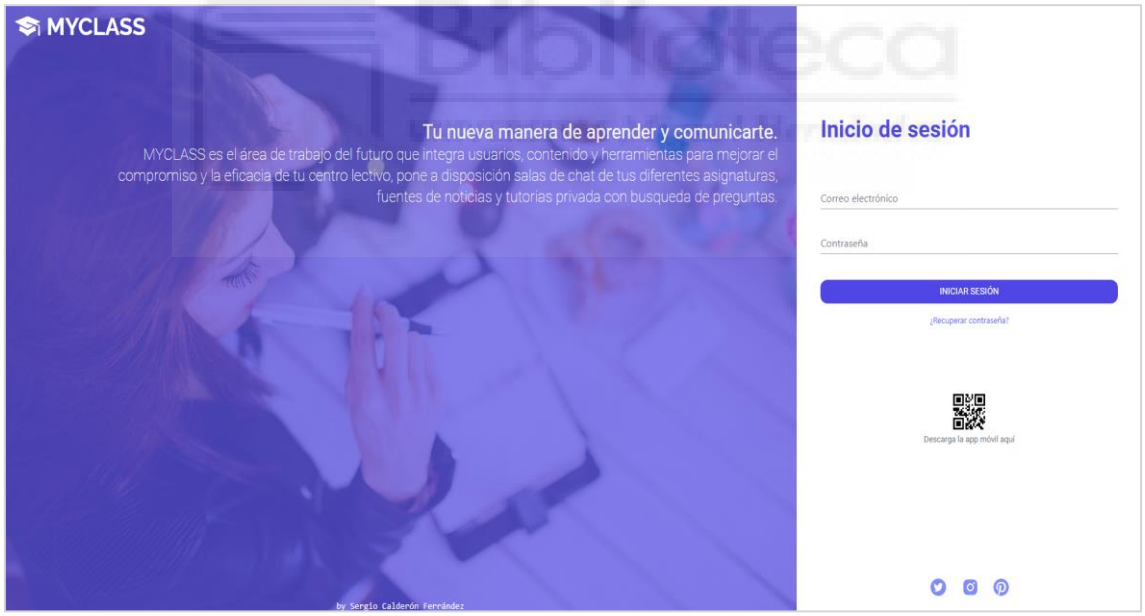


Ilustración 85. Página de inicio de sesión

Si el usuario elige la opción para recuperar la contraseña le aparecerá un modal como el que se muestra en la ilustración siguiente, en él, tiene que introducir el correo electrónico de la cuenta para la que se quiere restablecer la contraseña,

si este es válido, se le mandará un email que dispondrá de una URL que le llevará a la página de restablecimiento de contraseña.

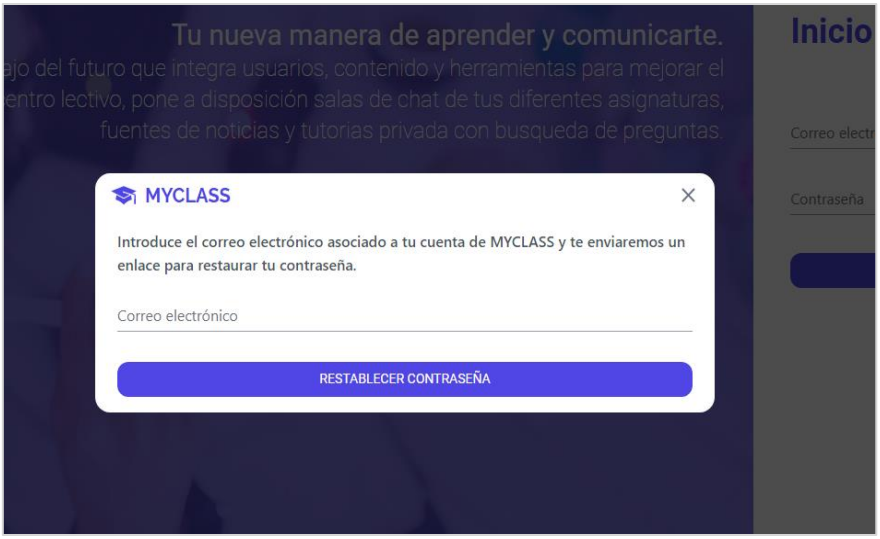


Ilustración 86. Modal restablecimiento de contraseña

6.1.2. PÁGINA PARA RESTABLECER CONTRASEÑA

Esta vista solo es accesible a través de la URL enviada a través del correo electrónico para restablecimiento de contraseña, en ella se muestra un formulario en el que se puede generar una nueva contraseña para la cuenta.

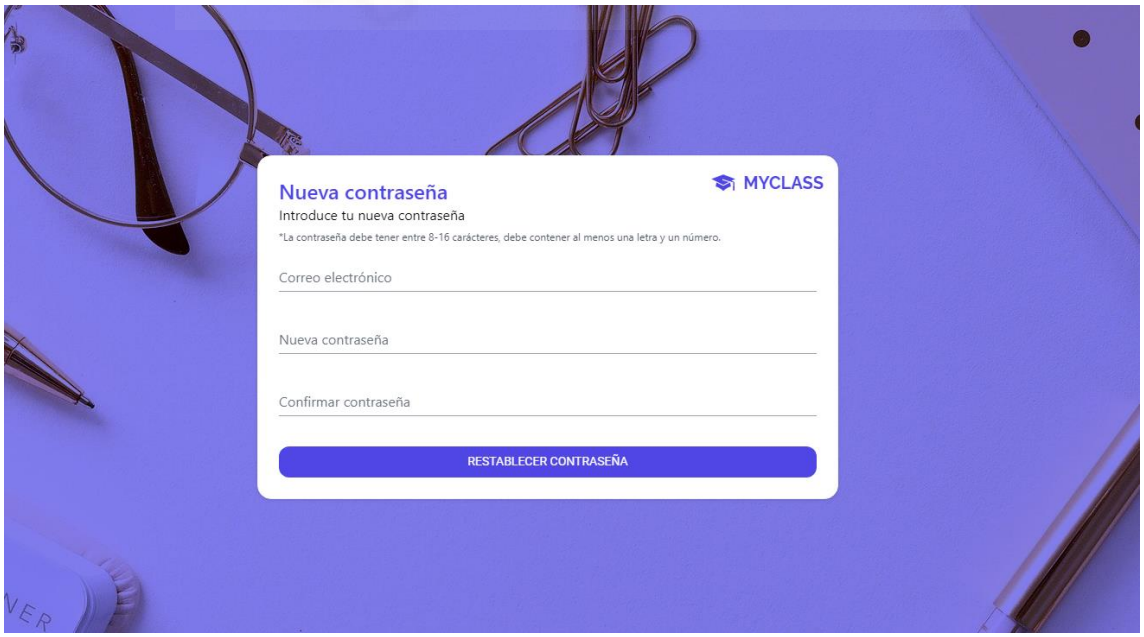


Ilustración 87. Página de restablecimiento de contraseña

6.2. VISTAS PARA USUARIO AUTENTICADO

En esta sección se expondrán las vistas que necesitan de un usuario autentificado en la aplicación. Esta vista está formada por el dashboard que dispone de una sección para el contenido principal que irá variando según la ruta donde nos encontremos, además dispone de un menú lateral que contendrá las diferentes rutas accesible por el usuario dependiendo del rol de este.

6.2.1. PÁGINA DE HOME

En esta vista se muestran la pantalla home para el rol de administrador y estudiante, para este último caso coincide con la pantalla que le aparecería a un usuario con el rol de profesor.

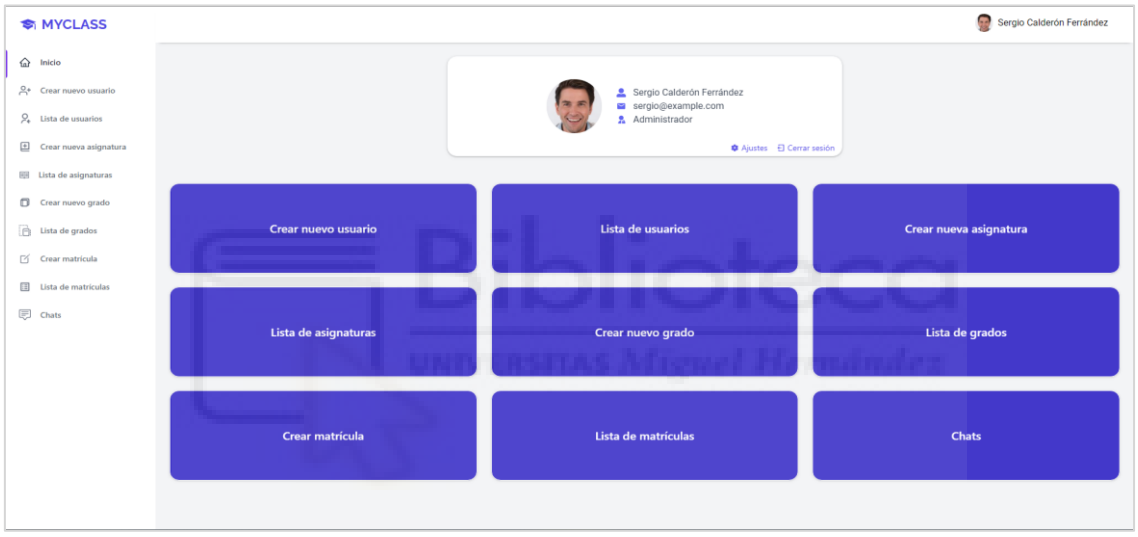


Ilustración 88. Página de home para el rol de administrador

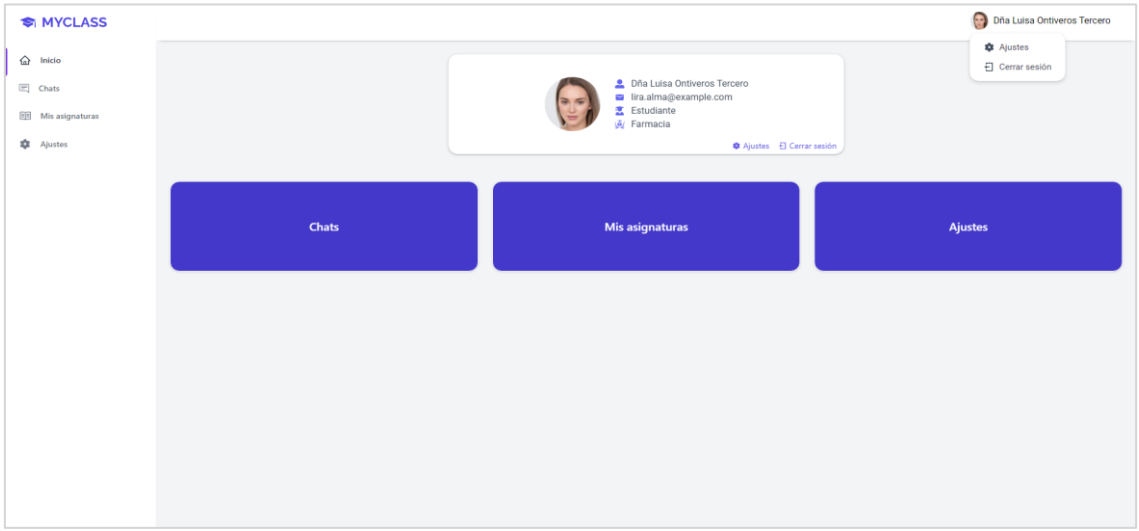


Ilustración 89. Página de home para el rol de estudiante

Esta pantalla ofrece un menú con todas las funciones disponibles en la aplicación web de una forma rápida y sencilla, nótese que estas coinciden con las opciones que nos ofrece el menú lateral del dashboard.

6.2.2. PÁGINAS PARA LA GESTIÓN DE USUARIOS

En esta sección se mostrarán las diferentes páginas para gestionar los usuarios disponibles en la aplicación. Estas páginas solo serán accesibles para los usuarios que dispongan del rol de administrador.

Crear usuario

En esta vista se muestra la página para registrar nuevos usuarios en la aplicación, en ella aparece un formulario con los datos necesarios para crear un nuevo recurso del tipo usuario, una vez se ha realizado correctamente el formulario, este generará una contraseña aleatoria que enviará por correo electrónico para informar al usuario de que ya puede comenzar a usar la aplicación web.

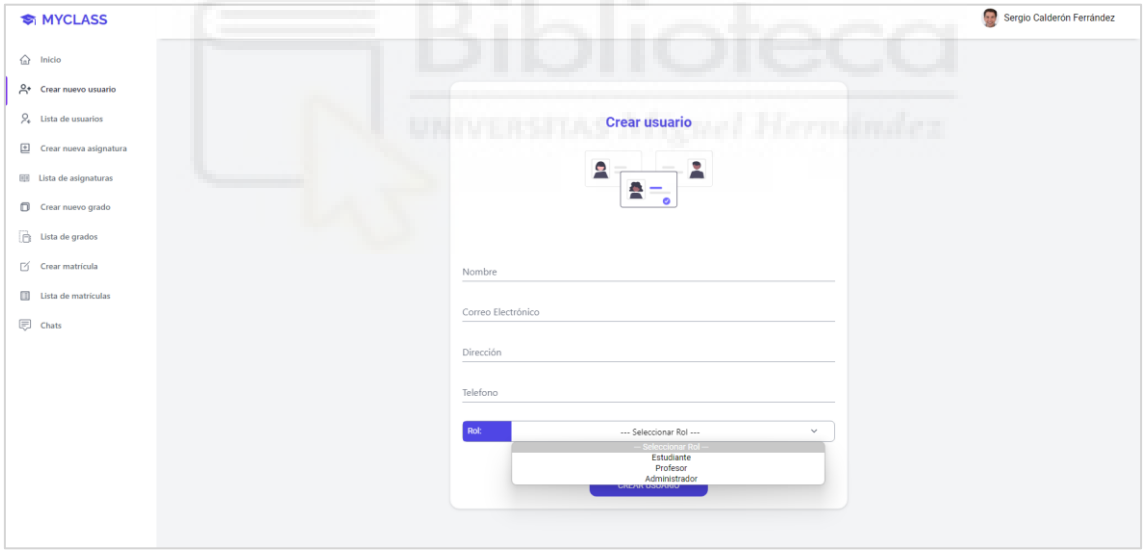


Ilustración 90. Página de creación de un usuario

Lista de usuarios

En esta página se muestra una lista de todos los usuarios de la aplicación en la que se puede filtrar por nombre de usuario, rol o email, asimismo la lista tiene una paginación basada en infinite scroll que irá cargando usuarios conforme vayamos haciendo scroll hacia abajo en la vista. Además cada usuario de la lista

dispone de un botón que nos llevará a la vista para editar el usuario seleccionado.

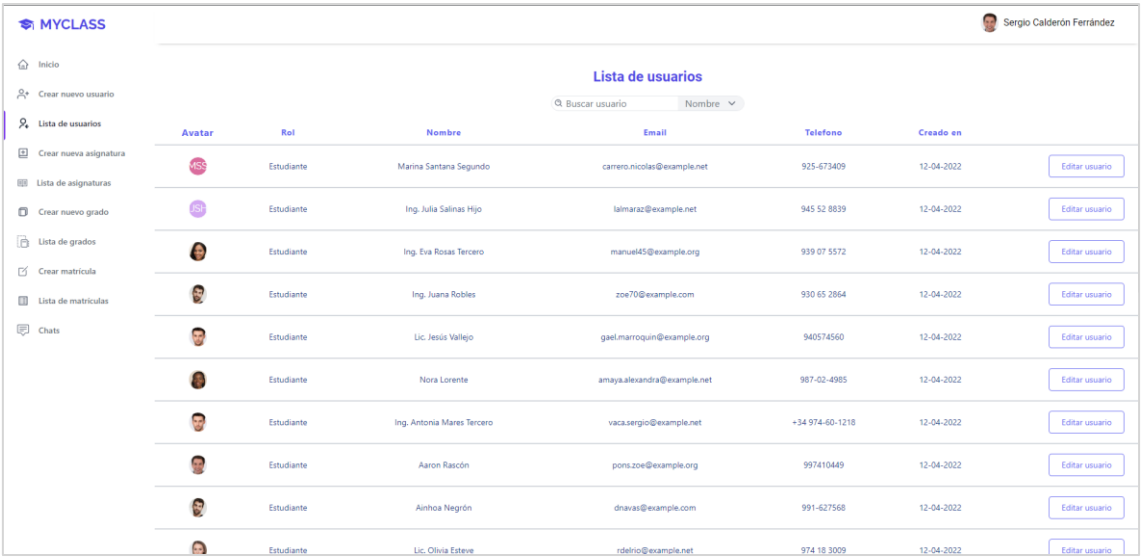


Ilustración 91. Página que muestra la lista de usuarios

Editar usuario

La ilustración siguiente muestra la vista para editar un usuario, en ella se dispone de un formulario para actualizar los diferentes campos del usuario y una opción para eliminar la cuenta de la aplicación.

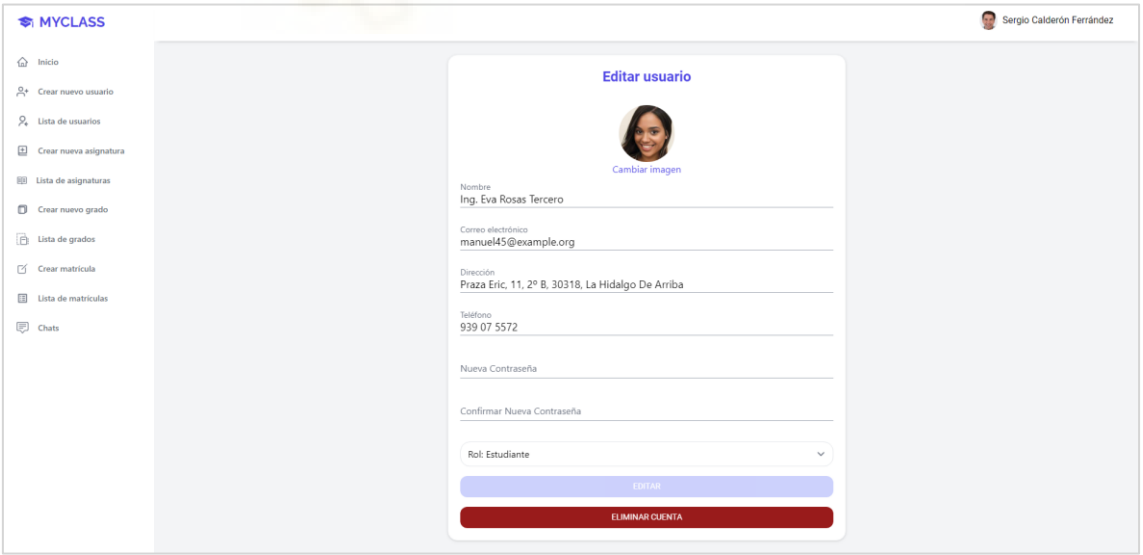


Ilustración 92. Página para editar un usuario

6.2.3. PÁGINAS PARA LA GESTIÓN DE ASIGNATURAS

En esta sección se mostrarán las diferentes páginas para gestionar las asignaturas disponibles en la aplicación. Estas páginas solo serán accesibles para los usuarios que dispongan del rol de administrador.

Crear asignatura

En esta página se muestra el formulario necesario para crear una nueva asignatura.

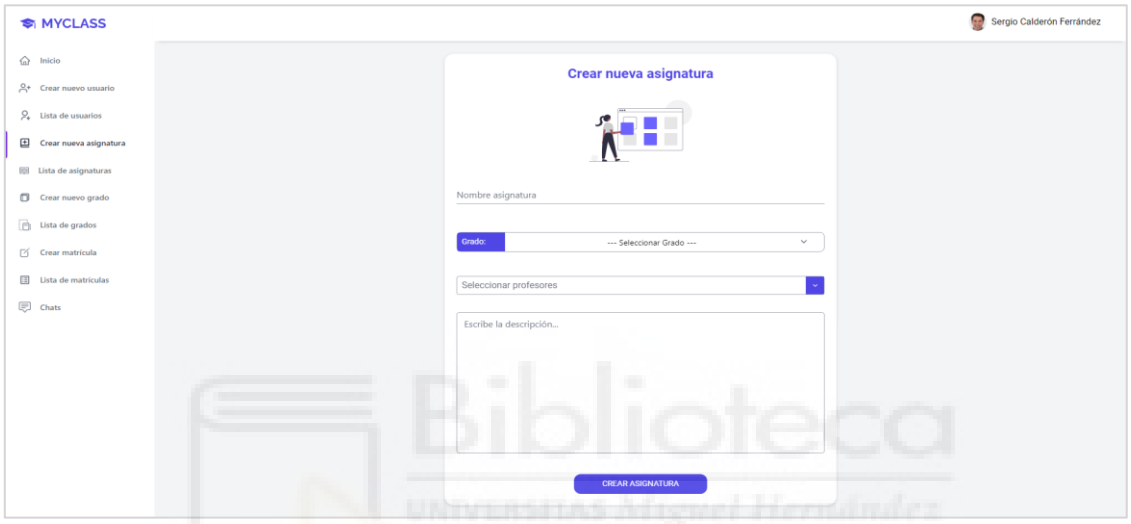


Ilustración 93. Página de creación de asignatura

Lista de asignaturas

Al igual que en la vista de la lista de usuarios, disponemos de una vista con las mismas funciones para el recurso de asignaturas de la aplicación.

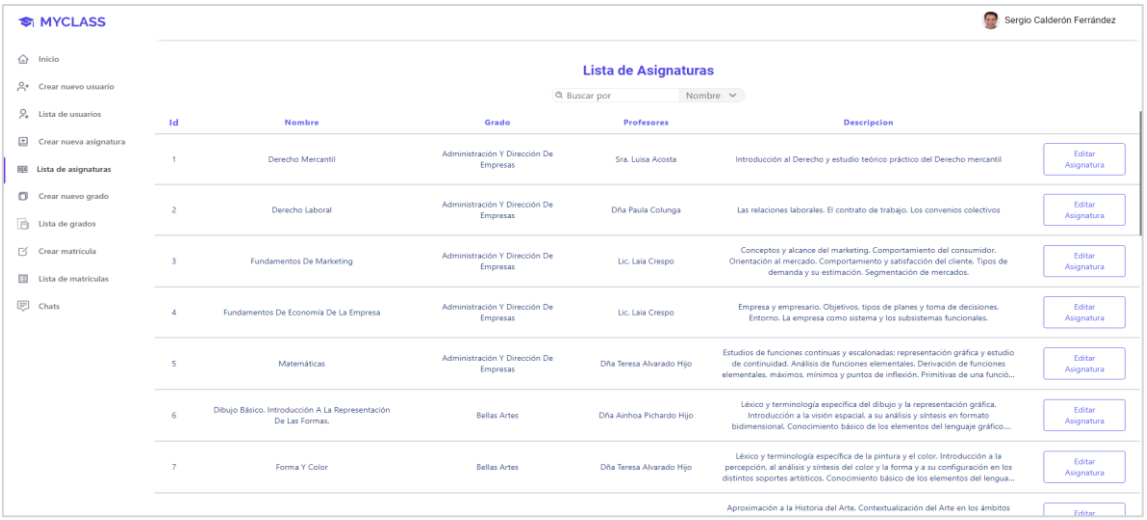


Ilustración 94. Página que muestra la lista de asignaturas

Editar asignatura

Esta página muestra el formulario para actualizar una asignatura o eliminarla del sistema.

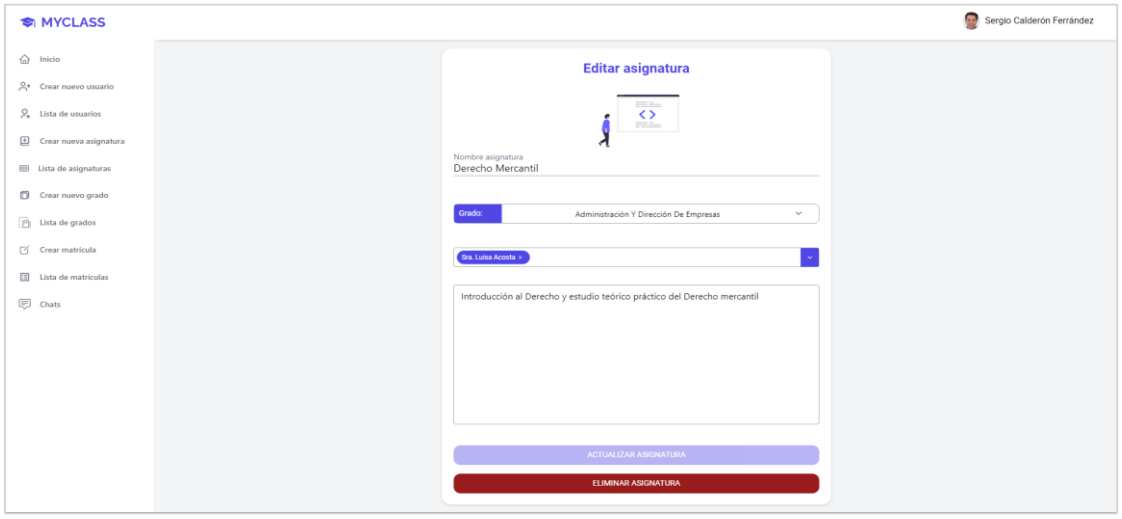


Ilustración 95. Página para editar una asignatura

6.2.4. PÁGINAS PARA LA GESTIÓN DE GRADOS

En esta sección se mostrarán las diferentes páginas para gestionar los grados disponibles en la aplicación. Estas páginas solo serán accesibles para los usuarios que dispongan del rol de administrador.

Crear grado

En esta vista se muestra el formulario necesario para crear un nuevo grado.

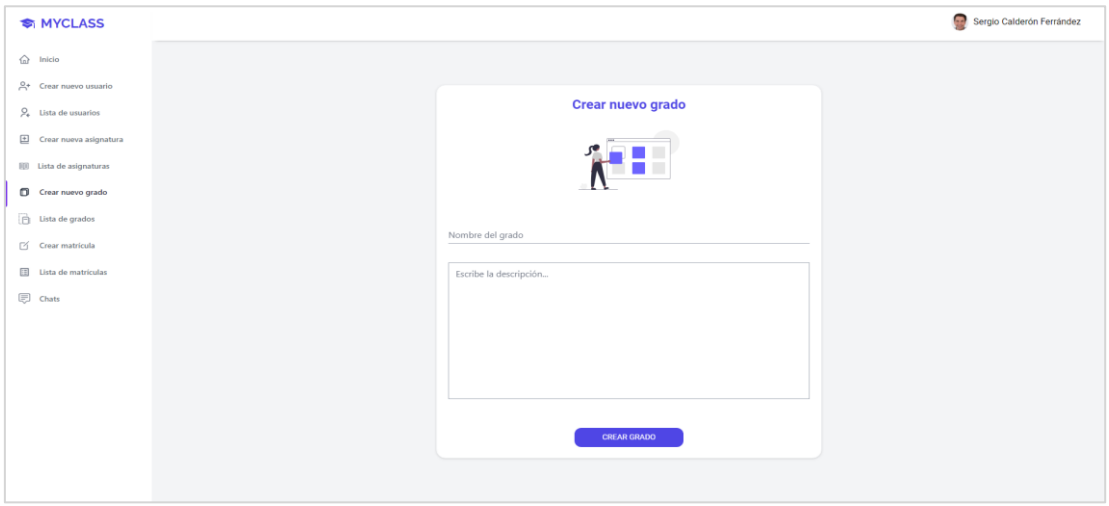


Ilustración 96. Página de creación de un grado

Lista de grados

Esta vista contiene una lista con los grados disponibles en la aplicación que dispone de un buscador para filtrar por nombre de grado, asimismo cada grado mostrado dispone de un botón que navegará a la página para editar el grado seleccionado.

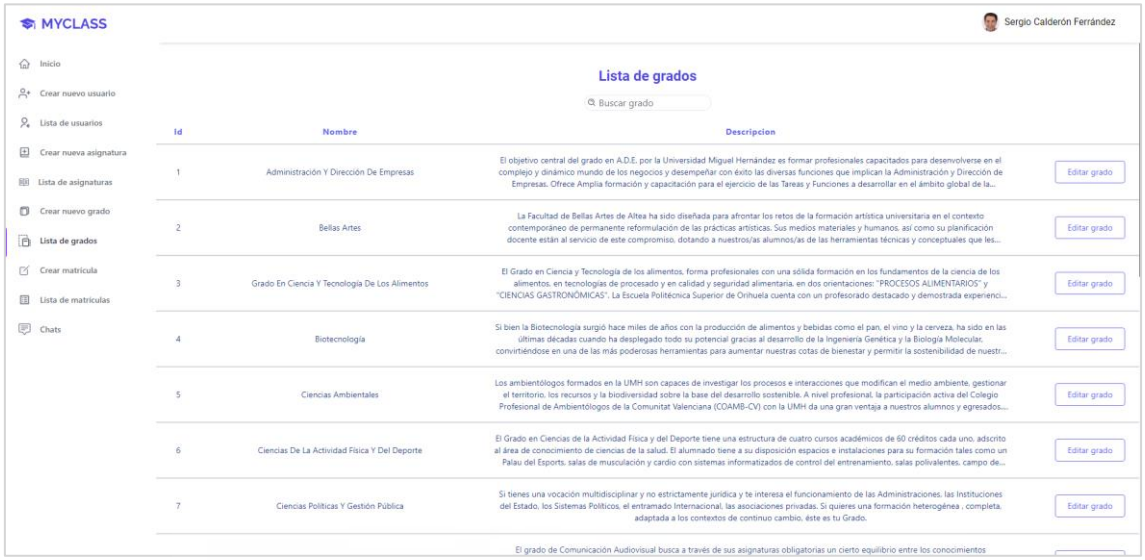


Ilustración 97. Página que muestra la lista de grados

Editar grado

En esta página se muestra el formulario para actualizar los datos de un determinado grado.

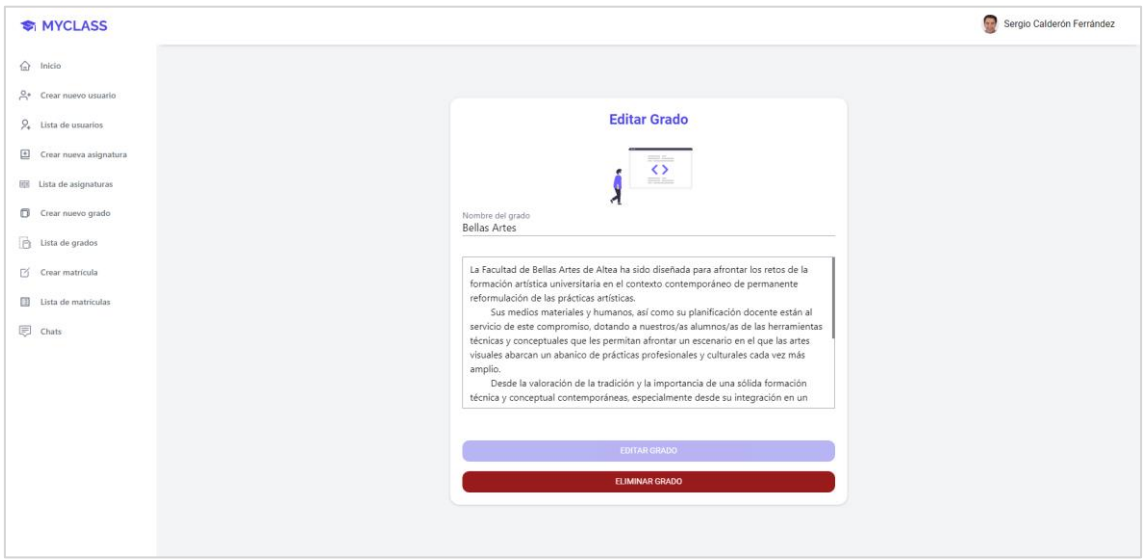


Ilustración 98. Página para editar un grado

6.2.5. PÁGINAS PARA LA GESTIÓN DE MATRÍCULAS

En esta sección se mostrarán las diferentes páginas para gestionar las matrículas disponibles en la aplicación. Estas páginas solo serán accesibles para los usuarios que dispongan del rol de administrador.

Crear matrícula

Esta vista muestra el formulario de creación para nuevas matrículas.

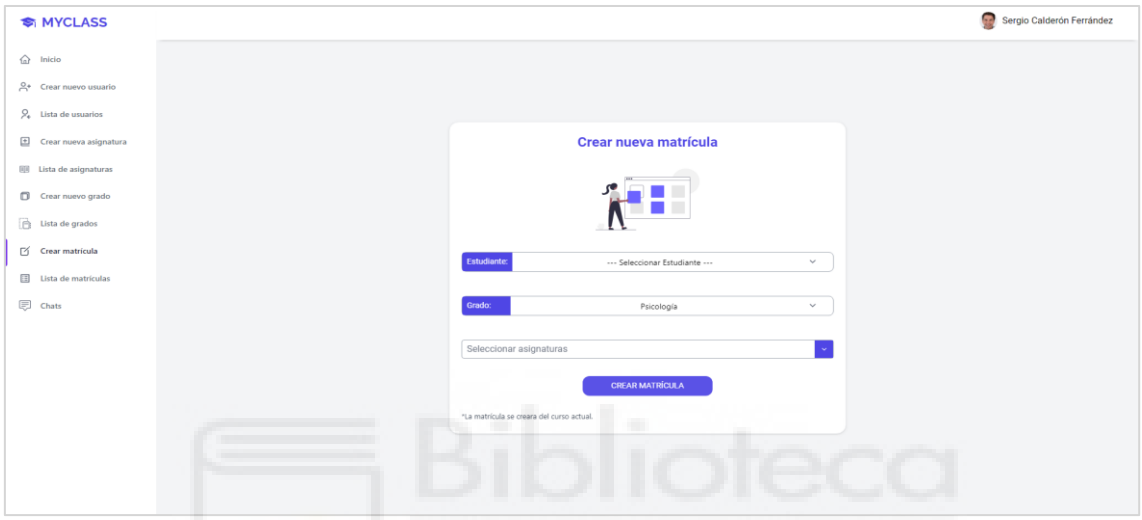


Ilustración 99. Página de creación de una matrícula

Lista de matrículas

Muestra una lista con las matrículas actuales de la aplicación que tiene la posibilidad de filtrarse mediante el buscador.

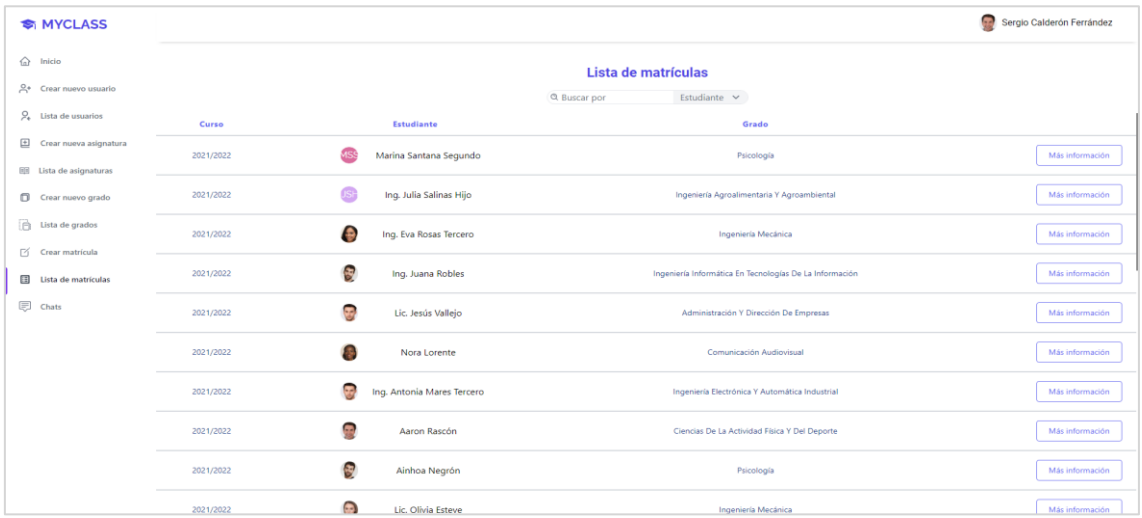


Ilustración 100. Página que muestra la lista de matrículas

Editar matrícula

Esta página muestra el formulario para actualizar los datos de una matrícula.

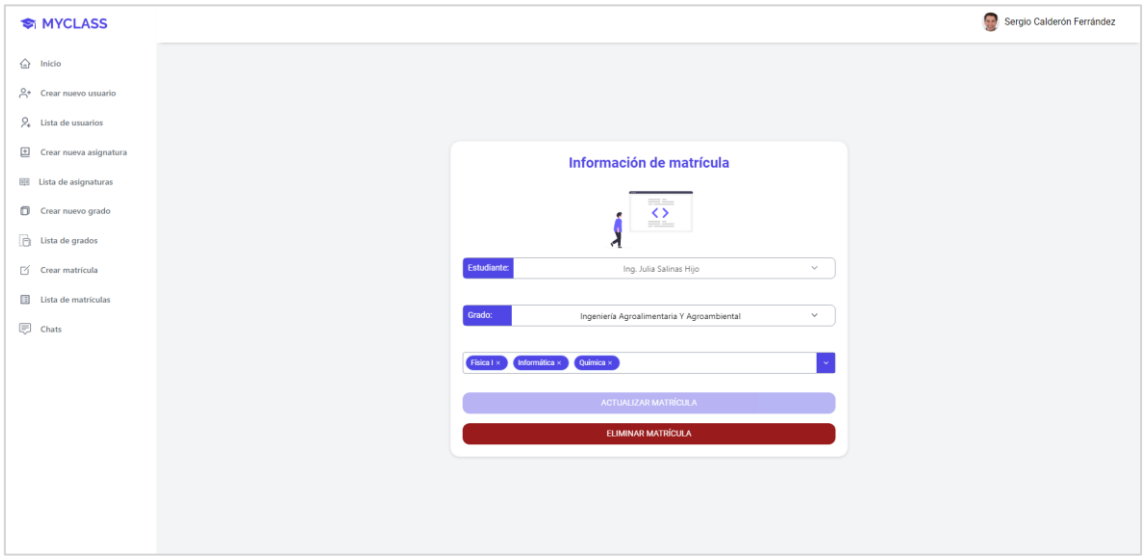


Ilustración 101. Página para editar una matrícula

6.2.6. PÁGINA DE AJUSTES

En esta sección se muestra la página o vista de ajustes de la cuenta de usuario en la que podemos actualizar nuestros datos, además de ajustar el volumen de las notificaciones del chat y cerrar sesión. Esta vista está disponible para todos los tipos de usuarios de la aplicación.

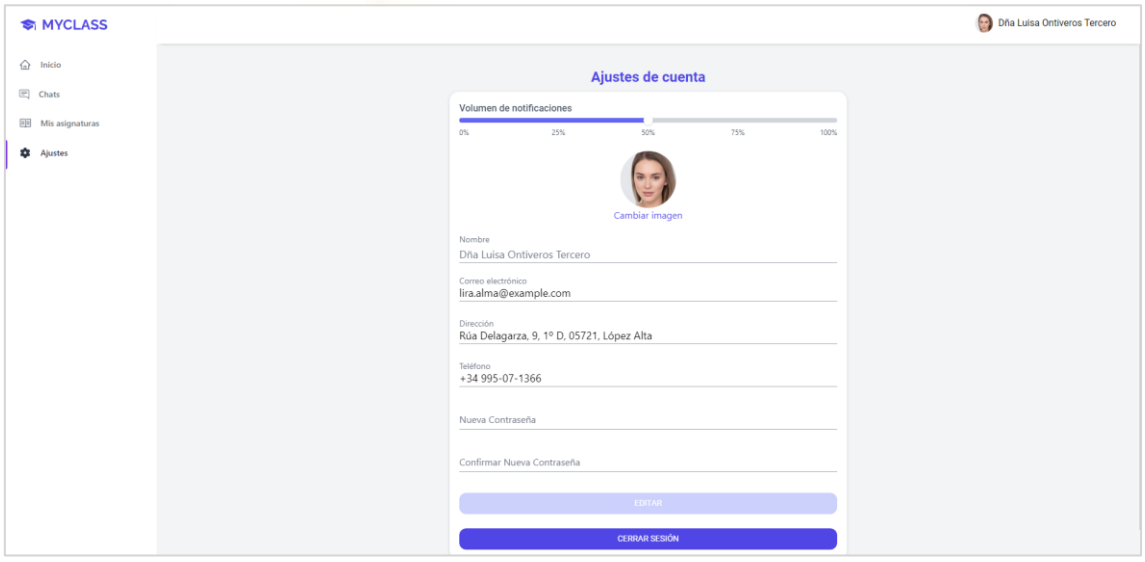


Ilustración 102. Página de ajuste de usuario

6.2.7. PÁGINA DE CHAT

En esta sección se mostrará la vista para la sala de chat y expondremos todas las funcionalidades que ofrece. Esta vista está disponible para todos los tipos de usuarios de la aplicación.

A continuación se muestra la página de chat de la aplicación, en ella se puede ver la lista de chats disponibles para el usuario, esta lista tiene la opción de filtrar por nombre los chats, asimismo también podemos filtrar los chats de asignaturas por los diferentes años lectivos. Nótese que los chats de las asignaturas se destacan por tener una estrella en el avatar del grupo.

Ilustración 103. Página de chat – Sin chat seleccionado

En esta pantalla ya estaremos teniendo una conexión a través websockets, por lo que cualquier mensaje que se envíe a cualquiera de la lista de chats será notificado por medio de un sonido y se incrementará el contador de mensajes no leídos del chat en cuestión. Desde esta lista simplemente tenemos que hacer click en un chat para entrar en la sala y poder visualizar o escribir mensajes en el chat en cuestión.

Por otro lado en esta vista disponemos de la opción para crear nuevos chats. Desde el botón de crear chats nos aparecerá un modal que contiene un formulario con diferentes pasos.

Como primer paso del formulario tenemos que seleccionar si queremos crear un chat privado o un chat grupal.

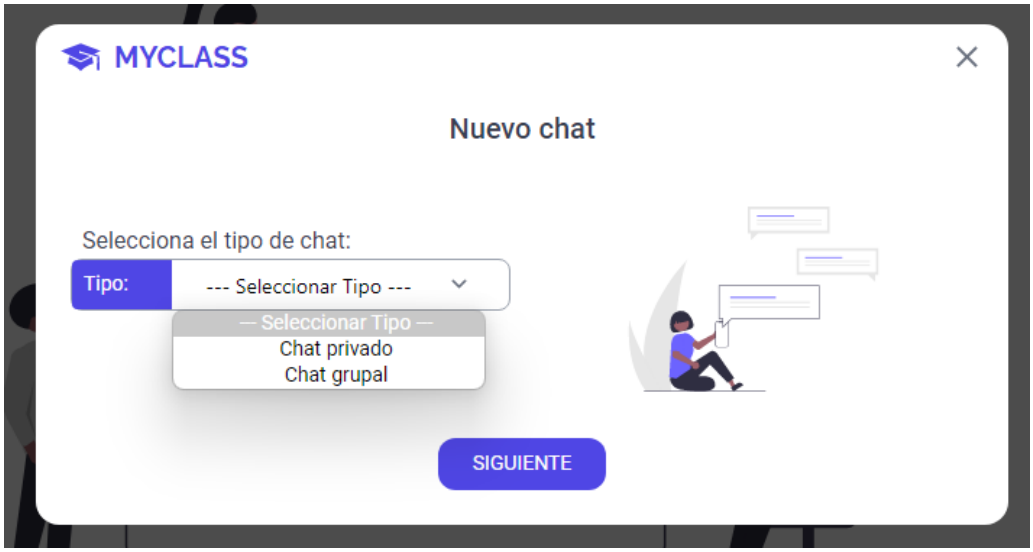


Ilustración 104. Modal para crear chat - paso 1

Si se selecciona un chat privado el siguiente paso simplemente es seleccionar al usuario y confirmar la creación del chat como se ve en la figura siguiente.

Ilustración 105. Modal para crear chat privado - paso 2

Si por el contrario seleccionamos crear un chat grupal, el siguiente paso sería elegir el nombre del grupo y el avatar, además en caso de que el usuario que quiere crear el chat grupal fuera un profesor tendría la opción de seleccionar la asignatura con la que se quiere relacionar el chat grupal, como por ejemplo para hacer grupos de prácticas de una determinada asignatura.

Ilustración 106. Modal para crear chat grupal - paso 2

Para finalizar solo quedaría elegir los usuarios que van a estar en la sala de chat, para el caso que se hubiera elegido una asignatura en la creación del chat, la lista de usuarios que aparece sería de los usuarios que están cursando la asignatura en cuestión, por el contrario si no se ha elegido ninguna asignatura la lista será de todos los usuarios que se encuentren en el sistema.

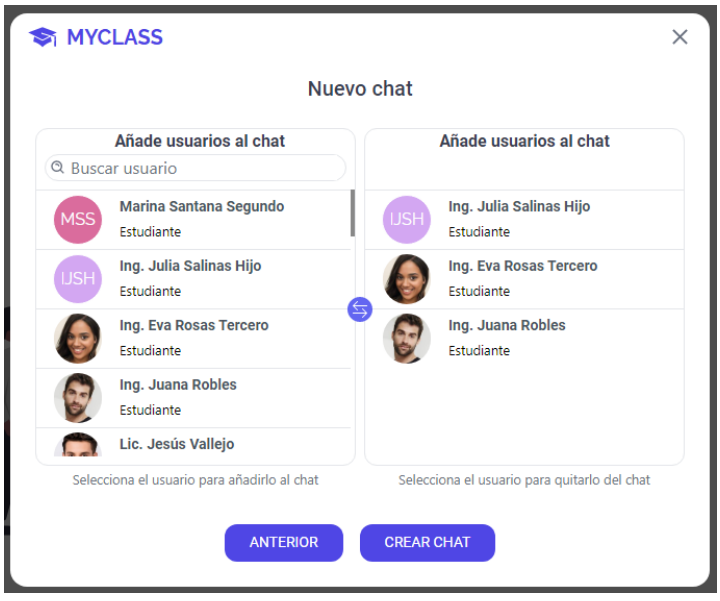


Ilustración 107. Modal para crear chat grupal - paso 3

Una vez se han seleccionado los integrantes de la sala simplemente tenemos que confirmar la creación del grupo y la aplicación nos entrará directamente en el chat creado.

En la ilustración siguiente mostraremos como es la interfaz para visualizar y escribir mensajes del chat, y expondremos algunas de sus funcionalidades.

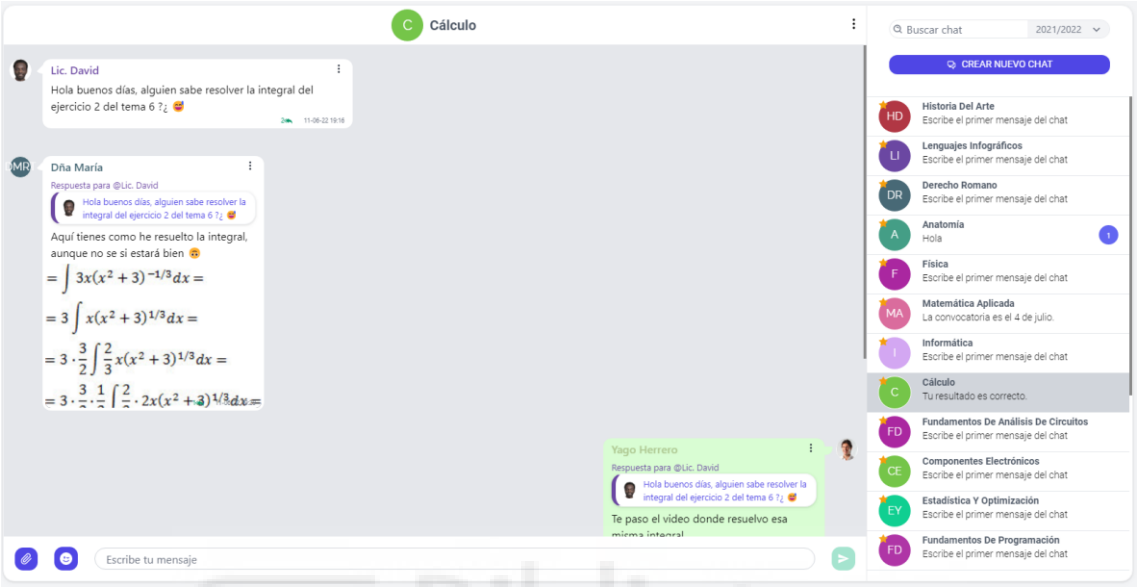


Ilustración 108. Página de chat - Chat seleccionado

La interfaz dispone de un menú en la cabecera en la que se encuentra el nombre y avatar del chat que contiene las opciones para salir del grupo y gestionar o visualizar información del chat.

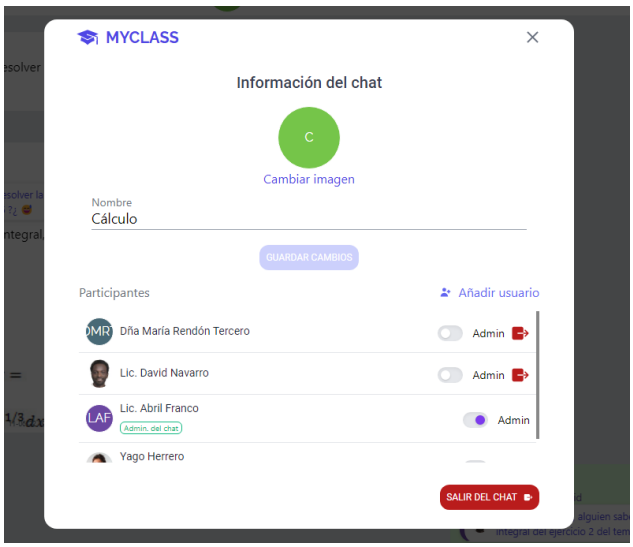


Ilustración 109. Menú información chat

Por otro lado la interfaz del chat permite adjuntar archivos a los mensajes mediante el botón inferior izquierdo que se muestra en la ilustración 107. Una vez se adjunta un fichero aparece una previsualización del archivo como se muestra a continuación.

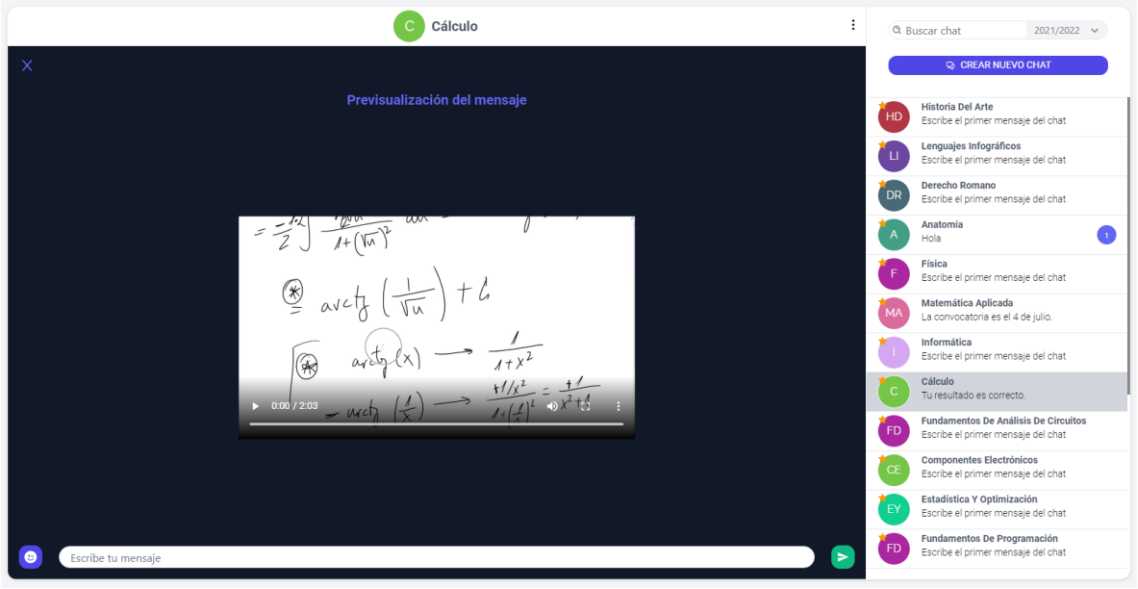


Ilustración 110. Página de chat – Previsualización archivo

Por otra parte los mensajes del chat contienen un menú como se ve en la ilustración siguiente, que permite responder un menaje, descargar el archivo adjunto de un mensaje o ver el hilo de mensajes, es decir, visualizar las respuestas del mensaje de una manera rápida y ordenada. Además tenemos otra opción disponible en las salas de chat de las asignaturas en las que si eres profesor o administrador del chat podrás añadir el mensaje y sus respuestas a la sección de preguntas destacadas, como si fuera un hilo de un foro para que sea visibles por usuarios de otros cursos.

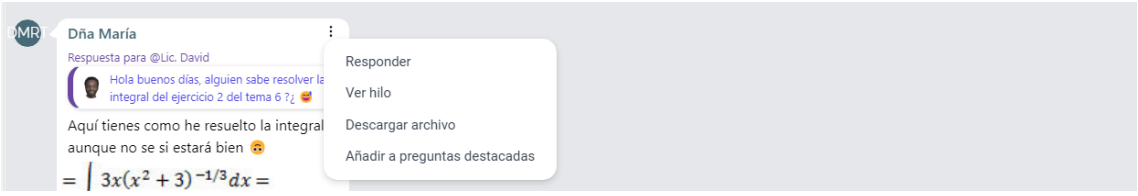


Ilustración 111. Ejemplo menú mensaje

Si seleccionamos la opción de responder el mensaje, en el formulario para escribir, nos aparecerá una alerta especificando el mensaje que tenemos seleccionado para responder, como se muestra en la ilustración siguiente.

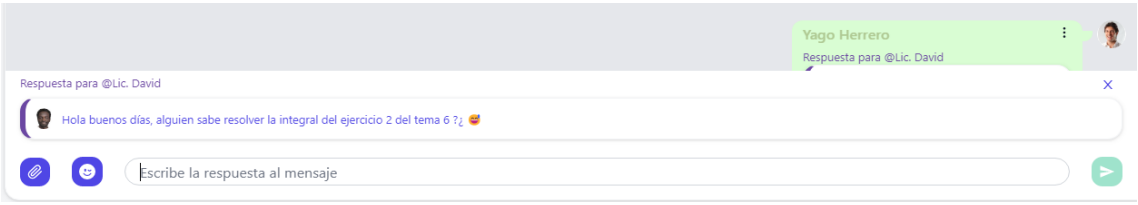


Ilustración 112. Entrada de chat - Mensaje seleccionado

Si elegimos la opción de ver hilo, está nos mostrará todas las respuestas del mensaje de una forma organizada y sencilla como podemos ver a continuación.

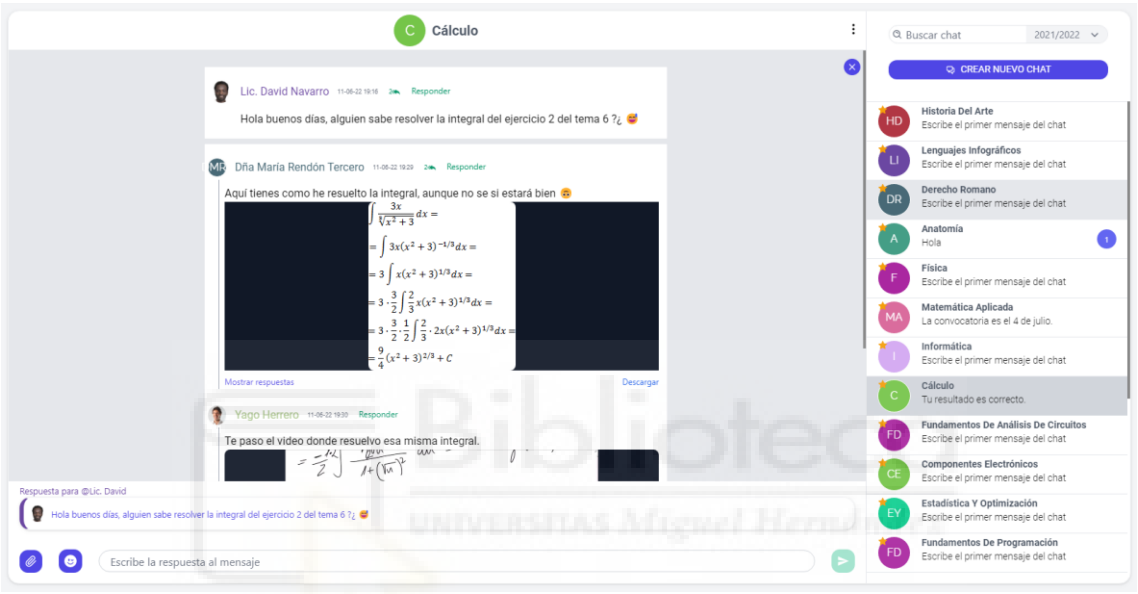


Ilustración 113. Página de chat - Hilo de mensaje

Por otro lado si se selecciona la opción de añadir a preguntas destacadas de un mensaje se nos abrirá un modal para colocarle un título descriptivo y este será publicado en la sección de “preguntas destacadas” de la asignatura con todas las respuestas.

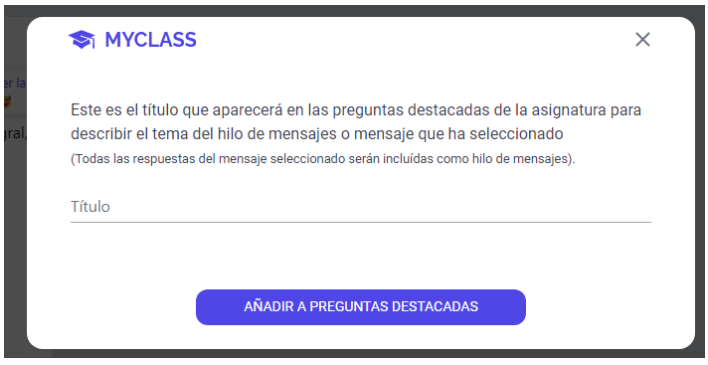


Ilustración 114. Chat - Modal añadir a preguntas destacadas

6.2.8. PÁGINAS DE MIS ASIGNATURAS

En esta sección mostraremos la interfaz para visualizar los datos de las asignaturas que estemos cursando o impartiendo. Está vista está disponible para los usuarios de tipo estudiante y profesor.

A continuación se muestra la página de mis asignaturas en la que aparecen la lista de asignaturas que se están cursando o impartiendo, asimismo dispone de la posibilidad de ver las asignaturas de años anteriores mediante el selector de curso disponible en la parte superior.

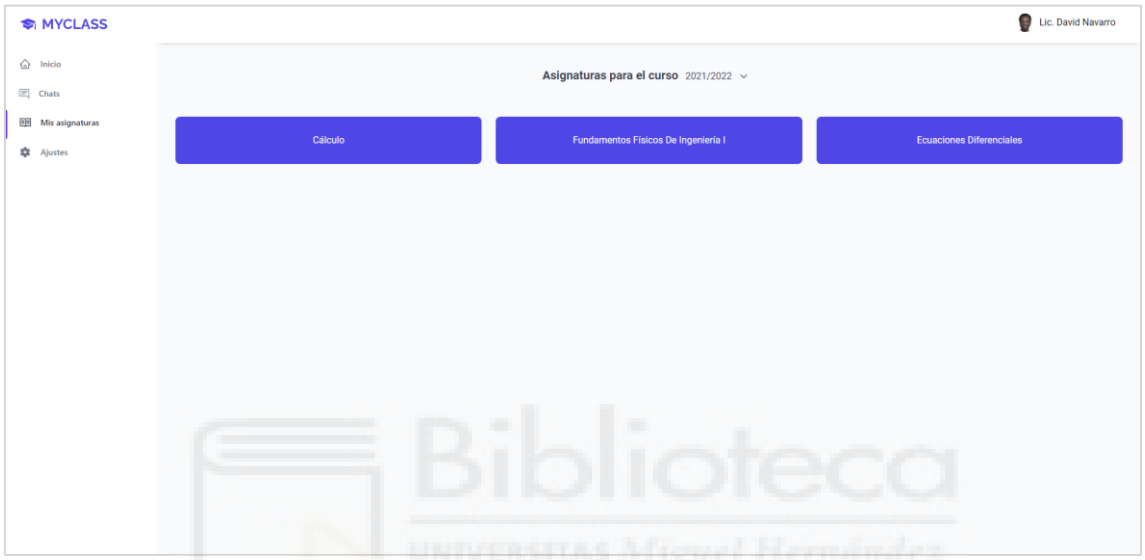


Ilustración 115. Página de mis asignaturas

Por último, si seleccionamos una asignatura de la lista, esta nos llevará a la página de la asignatura como se muestra en la ilustración siguiente.

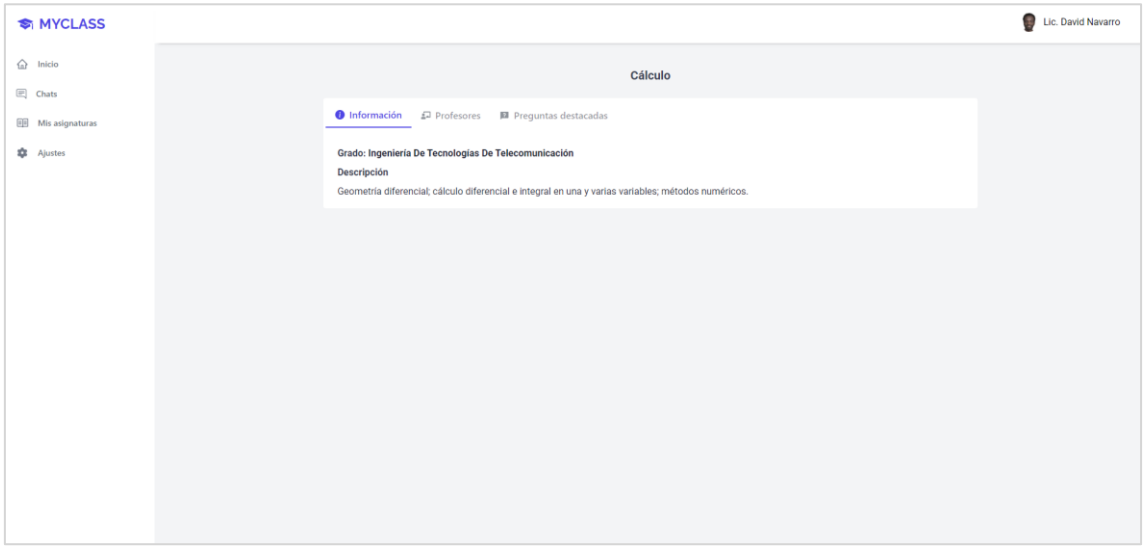


Ilustración 116. Página de una asignatura

En esta página tenemos acceso a la información de la asignatura, los profesores que la están impartiendo y la sección de preguntas por medio de un sistema de tabs.

A continuación se mostrará como ejemplo la sección de preguntas destacadas de la asignatura, que contiene los mensajes seleccionados por el profesor o administrador del grupo de chat. Esta sección servirá para tener una colección de las dudas resueltas más comunes en la asignatura, permitiendo a los alumnos de cursos venideros resolverlas con facilidad.

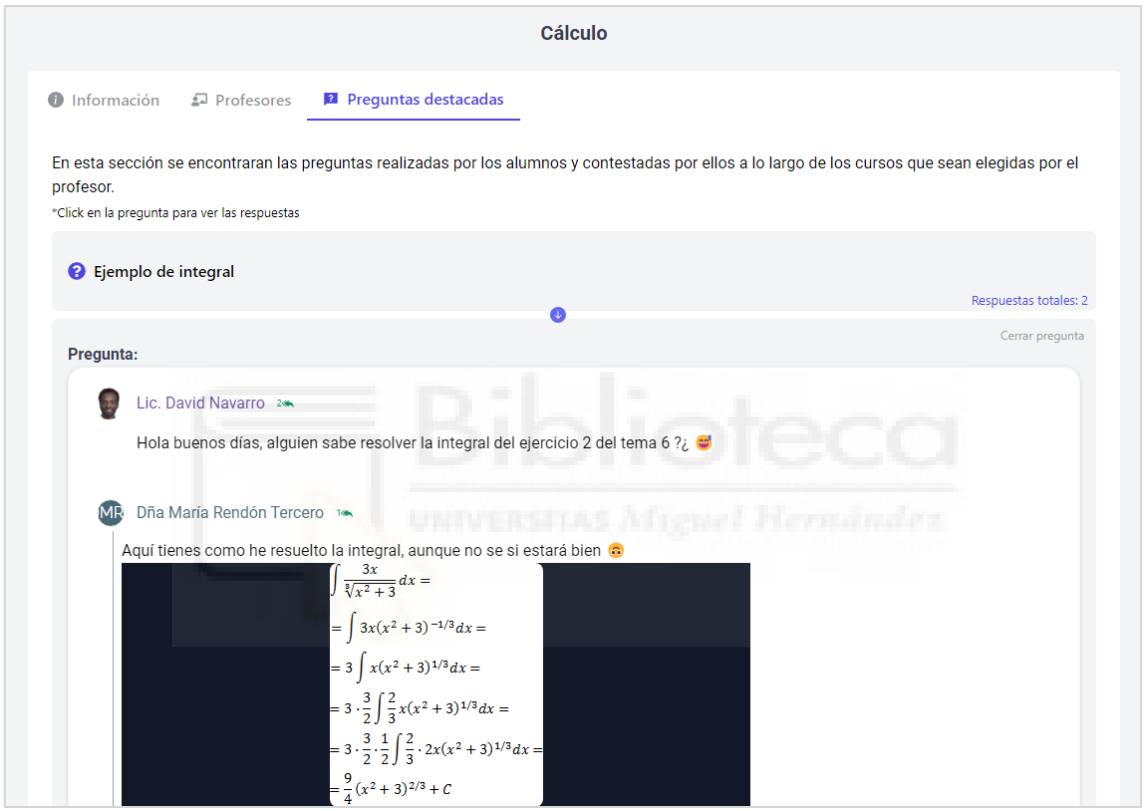


Ilustración 117. Página de una asignatura – Ejemplo preguntas destacadas

Antes de finalizar este capítulo me gustaría comentar que toda la interfaz mostrada es responsive para cualquier tipo de pantalla, permitiendo su uso tanto en ordenador, tablets y dispositivos móviles.

7. CONCLUSIONES

En este capítulo se recogen las conclusiones del proyecto, así como posibles desarrollos futuros para la aplicación realizada.

7.1. CONCLUSIONES

Tras el desarrollo del presente trabajo de fin de grado, la primera conclusión que podemos obtener es que se ha conseguido llevar a cabo el objetivo principal marcado al inicio de esta memoría, que consistía en lograr el desarrollo e implementación de una aplicación web que tiene como finalidad crear un prototipo de herramienta educativa de comunicación a tiempo real mediante un chat-foro.

Por otro lado, a modo personal podemos destacar distintas conclusiones. En primer lugar se ha comprendido la complejidad y delicadeza que conlleva el desarrollo de una aplicación web. Además he conseguido ampliar mis conocimientos y habilidades para desarrollar aplicaciones web con las tecnologías más populares como React y Laravel que serán de gran ayuda para mi futuro profesional. Asimismo se han adquirido nociones básicas para la administración de servidores y manejos de plataformas cloud como amazon AWS.

Por último, he de comentar que este proyecto ha sido un gran reto que me ha aportado personalmente muchos conocimientos en un ámbito que me apasiona como es el desarrollo web.

7.2. DESARROLLOS FUTUROS

Pese a que se han cumplido la totalidad de objetivos y requerimientos expuestos en la memoria, a continuación se exponen una serie de mejoras que se me han ocurrido durante el diseño e implementación de la aplicación para posibles desarrollos futuros.

- Utilizar un sistema de inteligencia artificial para la detección automática de preguntas ya respondidas sobre un tema, referenciando a las respuesta existente en la aplicación.
- Mejorar la interfaz de usuario y el diseño responsive para todo tipo de pantallas y navegadores.
- Incluir un sistema de notificación global para la aplicación.
- Incluir una implementación propia de la librería websockets que permita la conexión bidireccional con la aplicación de Laravel, esto permitiría reducir al máximo las peticiones a la API.

8. BIBLIOGRAFÍA

[1] Información sobre Microsoft Teams.
https://www.microsoft.com/es-xl/momentumms/13-razones-para-emplear-microsoft-teams/

[2] Características Microsoft Teams.
https://consejeriainformatica.com/13-razones-para-emplear-microsoft-teams/

[3] Información Google Classroom.
https://www.xataka.com/basics/google-classroom-que-como-funciona

[4] Información sobre aplicaciones de mensajería para la educación.
https://blogthinkbig.com/apps-de-mensajeria-instantanea-para-la-educacion-profesores-padres-y-alumnos

[5] Información sobre Slack.
http://elearningmasters.galileo.edu/2018/04/13/slack-en-la-educacion-virtual/

[6] Información sobre Moodle.
https://docs.moodle.org/all/es/Acerca_de_Moodle

[7] Información sobre servidores web.
https://www.stackscale.com/es/blog/top-servidores-web/

[8] ¿ Qué es una base de datos relacional ?.
https://www.oracle.com/es/database/what-is-a-relational-database/

[9] Información sobre los gestores de base de datos más importantes.
https://www.inesem.es/revistadigital/informatica-y-tics/los-gestores-de-bases-de-datos-mas-usados/

[10] ¿ Qué es una base de datos no relacional ?.
https://aws.amazon.com/es/nosql/

[11] Top frameworks para el lado del servidor.
https://dev.to/kaperskyguru/top-5-backend-frameworks-2021-20hd

[12] Tecnologías usadas por el cliente en una aplicación web.
https://docs.microsoft.com/es-es/dotnet/architecture/modern-web-apps-azure/common-client-side-web-technologies

[13] ¿ Qué es TailwindCSS ?.
https://es.wikipedia.org/wiki/Tailwind_CSS

[14] ¿ Qué es Bootstrap ?.
https://www.hostinger.es/tutoriales/que-es-bootstrap

[15] ¿ Qué es Foundation ?.
https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/zurb-foundation-compacto-framework-ui

[16] Información sobre el framework de Angular.
https://angular.io/guide/what-is-angular

[17] Información sobre el framework de React.
https://es.reactjs.org/tutorial/tutorial.html#what-is-react

[18] Información sobre el framework de Vue.
https://es.vuejs.org/v2/guide/

[19] ¿ Qué es PHP ?.
https://www.php.net/manual/es/intro-whatis.php

[20] Información sobre el framework de Laravel.
https://www.tutorialspoint.com/laravel/laravel_overview.htm

[21] Información sobre Amazon AWS.
https://aws.amazon.com/es/

[22] Información sobre el editor de Visual Studio Code.
https://es.wikipedia.org/wiki/Visual_Studio_Code

[23] Web oficial de MySQL Workbench.
https://www.mysql.com/products/workbench/

[24] Web oficial de TablePlus.
https://tableplus.com/

[25] Web oficial de Postman.
https://www.postman.com/

[26] Web oficial de MobaXTerm.
https://mobaxterm.mobatek.net/

[27] Información sobre las DevTools de Chrome.
https://developer.chrome.com/docs/devtools/

[28] Web oficial de GitHub.
https://github.com/

[29] ¿ Qué es NodeJS ?.
https://openwebinars.net/blog/que-es-nodejs/

[30] Información sobre arquitectura de una aplicación web - 1
https://programacionwebisc.wordpress.com/2-1-arquitectura-de-las-aplicaciones-web/

[31] Información sobre arquitectura de una aplicación web - 2
https://www.clickittech.com/devops/web-application-architecture/

[32] ¿ Qué es una REST API ?.
https://www.ibm.com/cl-es/cloud/learn/rest-apis

[33] Documentación oficial de Laravel.
https://laravel.com/docs/9.x/releases

[34] Información sobre la arquitectura Modelo-Vista-Controlador.
https://fernando-gaitan.com.ar/laravel-parte-3-modelo-vista-controlador/

[35] ¿ Qué es el Websockets ?.
https://developer.mozilla.org/es/docs/Web/API/WebSockets_API

[36] Introducción a ReactJS.
https://medium.com/@alex_lobera/introduction-to-react-3000e9cbcd26

[37] ¿ Qué es el DOM ?.
https://developer.mozilla.org/es/docs/Web/API/Document_Object_Model/Introduction

[38] ¿ Qué es el Virtual DOM en React ?.
https://styde.net/que-es-el-virtual-dom-en-react/

[39] Documentación oficial de Surge.
https://surge.sh/

[40] Documentación oficial de Composer.
https://getcomposer.org/doc/

[41] Documentación de Laravel Sanctum.
https://laravel.com/docs/9.x/sanctum

[42] Documentación de Pusher.
https://pusher.com/

[43] Documentación Laravel Websockets.
https://beyondco.de/docs/laravel-websockets/getting-started/installation

[44] Documentación oficial de ReactJS.
https://es.reactjs.org/docs/getting-started.html

[45] Página de descarga de Node.
https://nodejs.org/es/download/

[46] Instalación de TailwindCSS en apliación de React.
https://tailwindcss.com/docs/guides/create-react-app

[47] Información sobre la librería Yup.
https://github.com/jquense/yup

[48] Documentación oficial Formik.
https://formik.org/docs/api/formik

[49] Documentación de React Router.
https://reactrouter.com/docs/en/v6

[50] Documentación de la librería de Laravel Echo.
https://github.com/laravel/echo

[51] Información sobre la librería de React Infinite Scroll.
https://github.com/ankeetmaini/react-infinite-scroll-component

ANEXO A: Instalación de tecnologías en el servidor

En este anexo se detallan los pasos a seguir para crear y poner en marcha una instancia EC2 (VPS) de Amazon AWS y la instalación de las diferentes tecnologías necesarias para el correcto funcionamiento de la parte del Backend.

Creación de una instancia en Amazon AWS

Para crear una instancia en Amazon AWS nos dirigimos a su web: https://aws.amazon.com/es/ donde crearemos una cuenta totalmente gratuita.

Ilustración 118. Página de AWS

Una vez hemos creado la cuenta, iniciamos sesión y desplegamos el menú de servicios del navbar o buscamos en la barra de navegación el servicio **EC2** y accedemos a él. A continuación, nos aparecerá el panel de EC2 desde donde podemos gestionar nuestras instancias o crear nuevas.

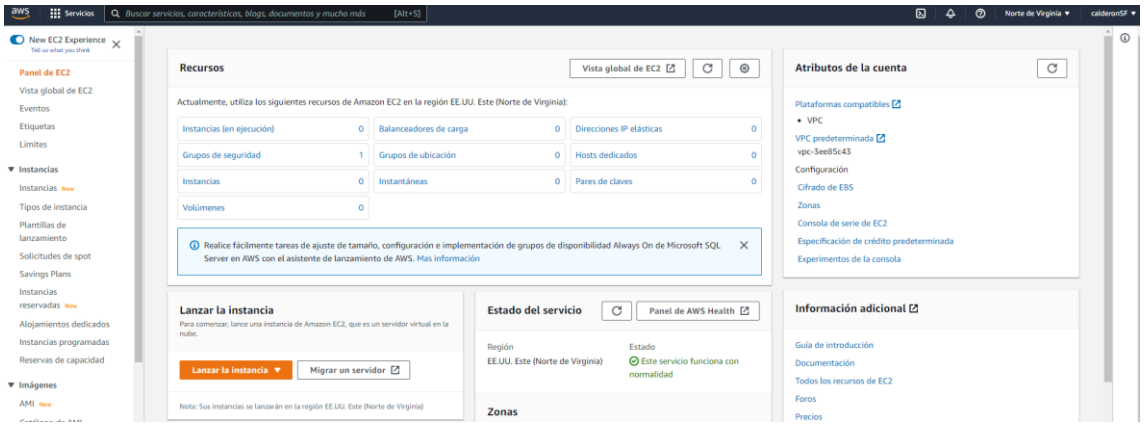


Ilustración 119. Dashboard AWS

Para crear una nueva instancia, hacemos clic en el botón de lanzar instancia y nos aparecerá una nueva pantalla con las opciones para configurarla, donde le tendremos que dar el nombre a la instancia, elegir el sistema operativo, la cantidad de memoria de almacenamiento y el tipo de máquina que queremos que ejecute nuestra instancia (para el caso de este proyecto se ha seleccionada la máquina que entra en la capa gratuita).

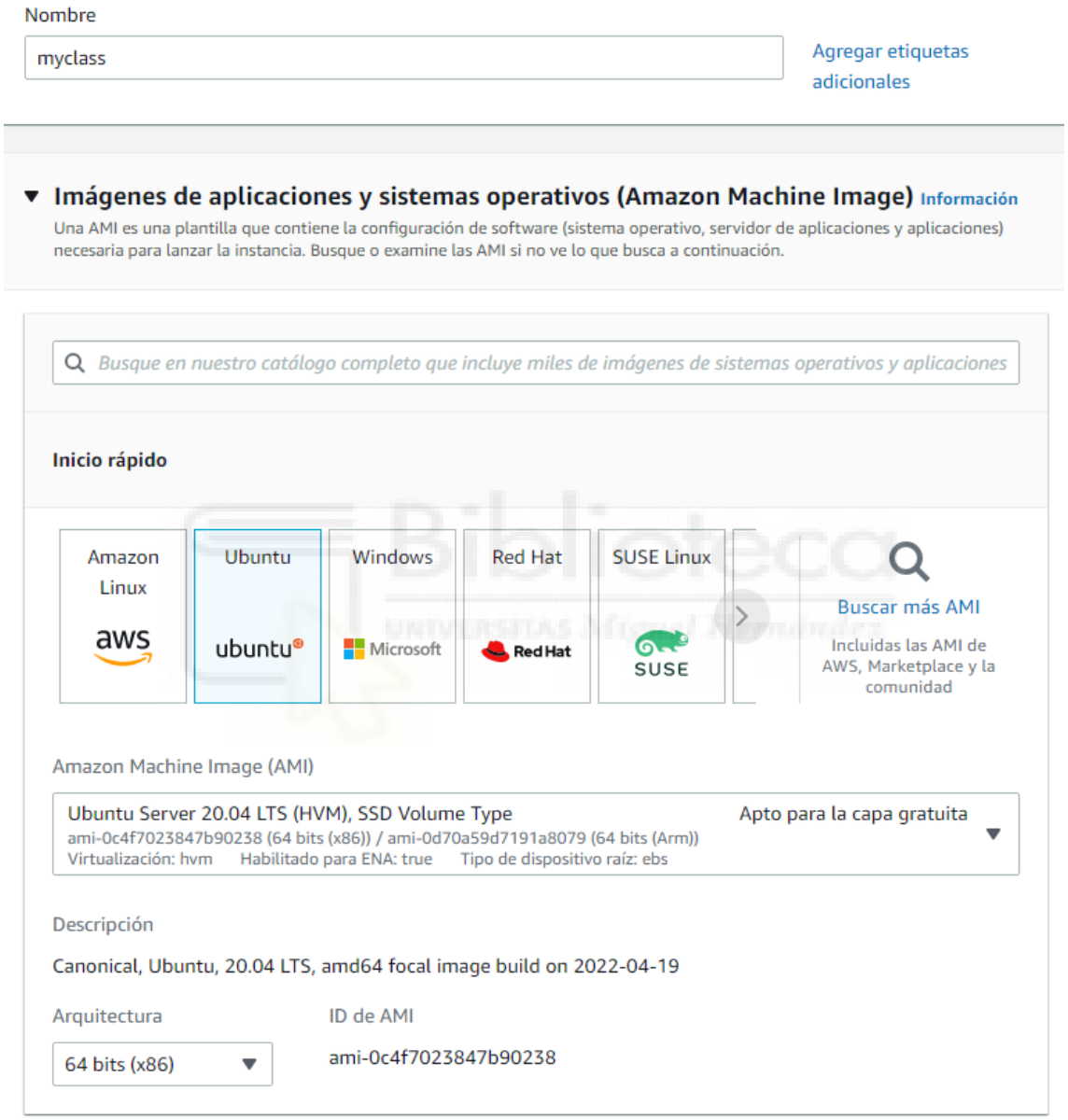


Ilustración 120. Pantalla de configuración Instancia AWS - 1

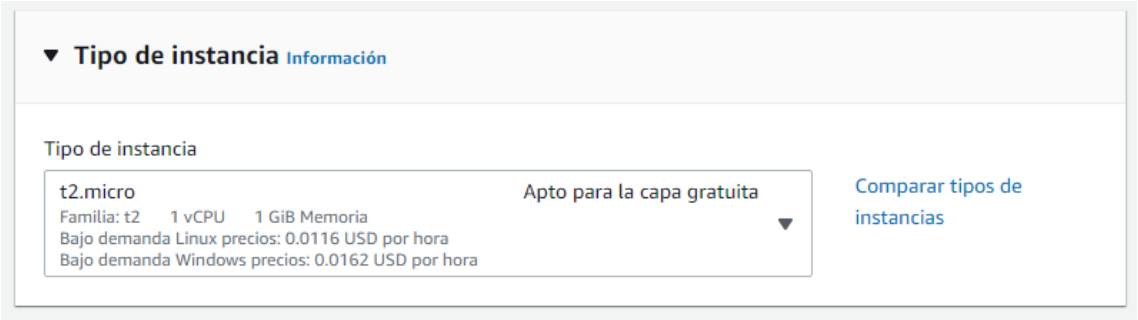


Ilustración 121. Pantalla de configuración Instancia AWS - 2

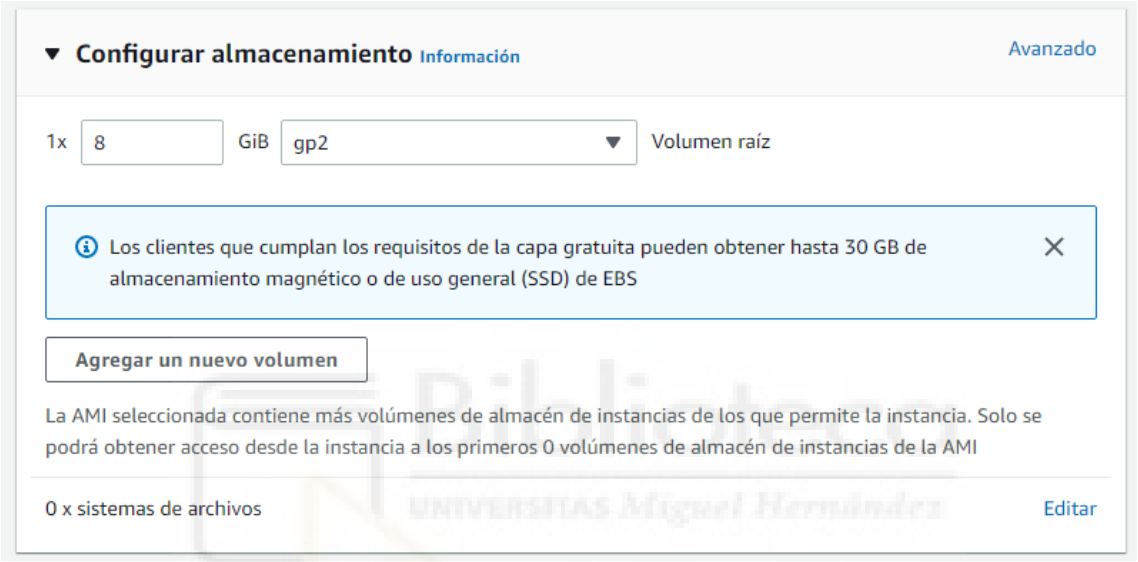


Ilustración 122. Pantalla de configuración Instancia AWS - 3

Además, nos permite hacer configuraciones de red directamente desde este menú, donde podemos habilitar tráfico SSH a nuestra máquina o abrir puertos de la máquina para permitir el tráfico en nuestra página web (puerto 80) o la conexión al servidor websockets (puerto 6001).

Una vez configurada nuestra instancia, hacemos clic en el botón de lanzar instancia, esta se empezará a crear y nos proveerá en unos segundos de una VPS totalmente configurada y funcional, la cual podremos acceder desde nuestro panel de instancias para poder detenerla o modificar su configuración.

Ilustración 123. Panel de instancias AWS

Instalación de las tecnologías del servidor

En esta sección se instalarán todas las tecnologías requeridas en el servidor para poder ejecutar nuestra aplicación del Backend.

Como ya se comentó en la memoria, se hará uso de Linux como sistema operativo, Apache como servidor web, PHP como lenguaje en el lado del servidor y MySQL como gestor de base de datos.

Para realizar la instalación de estas tecnologías nos conectamos por medio de SSH a nuestra máquina virtual de amazon AWS y seguimos los pasos siguientes.

Paso 1. Instalación del servidor web Apache

Para la instalación del servidor web de apache ejecutamos los siguientes comandos:

```
> sudo apt-get update

> sudo apt-get install apache2
```

Para verificar la instalación de nuestro servidor web, nos dirigimos a el navegador y navegamos a la dirección IP de nuestro servidor, en ella nos debería salir la página por defecto de Apache como se muestra a continuación.

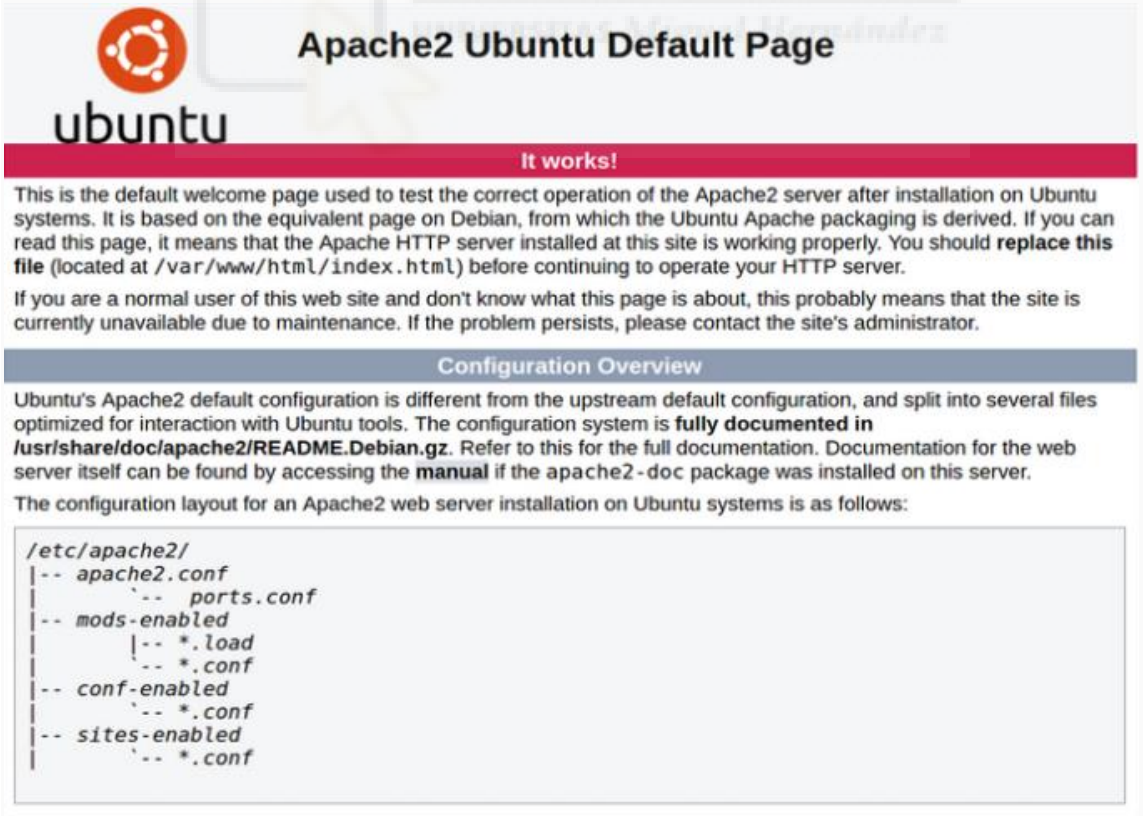


Ilustración 124. Página de Apache

Tener en cuenta que una vez creemos el proyecto de Laravel en la carpeta de **/var/www/html** tenemos que cambiar la configuración de Apache de la siguiente manera:

```
> nano /etc/apache2/sites-available/laravel.conf
```

Dentro de este archivo colocamos la siguiente configuración:

```
<VirtualHost *:80>
    ServerAdmin admin@example.com
    ServerName laravel.example.com
    DocumentRoot /var/www/html/backend-myclass/public

    <Directory /var/www/html/backend-myclass>
      Options Indexes MultiViews
      AllowOverride None
      Require all granted
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Guarda y cierra el archivo y luego habilita el módulo de reescritura de Apache y activa el host virtual Laravel con los comandos siguientes:

```
> a2enmod rewrite
> a2ensite laravel.conf
```

Por último reinicia el servicio de apache con el comando siguiente:

```
> systemctl restart apache2
```

Paso 2. Instalación MySQL

Para la instalación del gestor de base de datos MySQL ejecutaremos el siguiente comando:

```
> sudo apt-get install mysql-server
```

Paso 3. Instalación PHP

Para la instalación de PHP ejecutaremos los siguientes comandos en la consola, estos comandos instalarán PHP y algunas de sus extensiones, además reiniciaremos el servidor web de Apache para que detecte los cambios realizados en el servidor:

```
> sudo apt-get install php libapache2-mod-php php-mcrypt php-
mysql
```

```
> sudo systemctl restart apache2
```

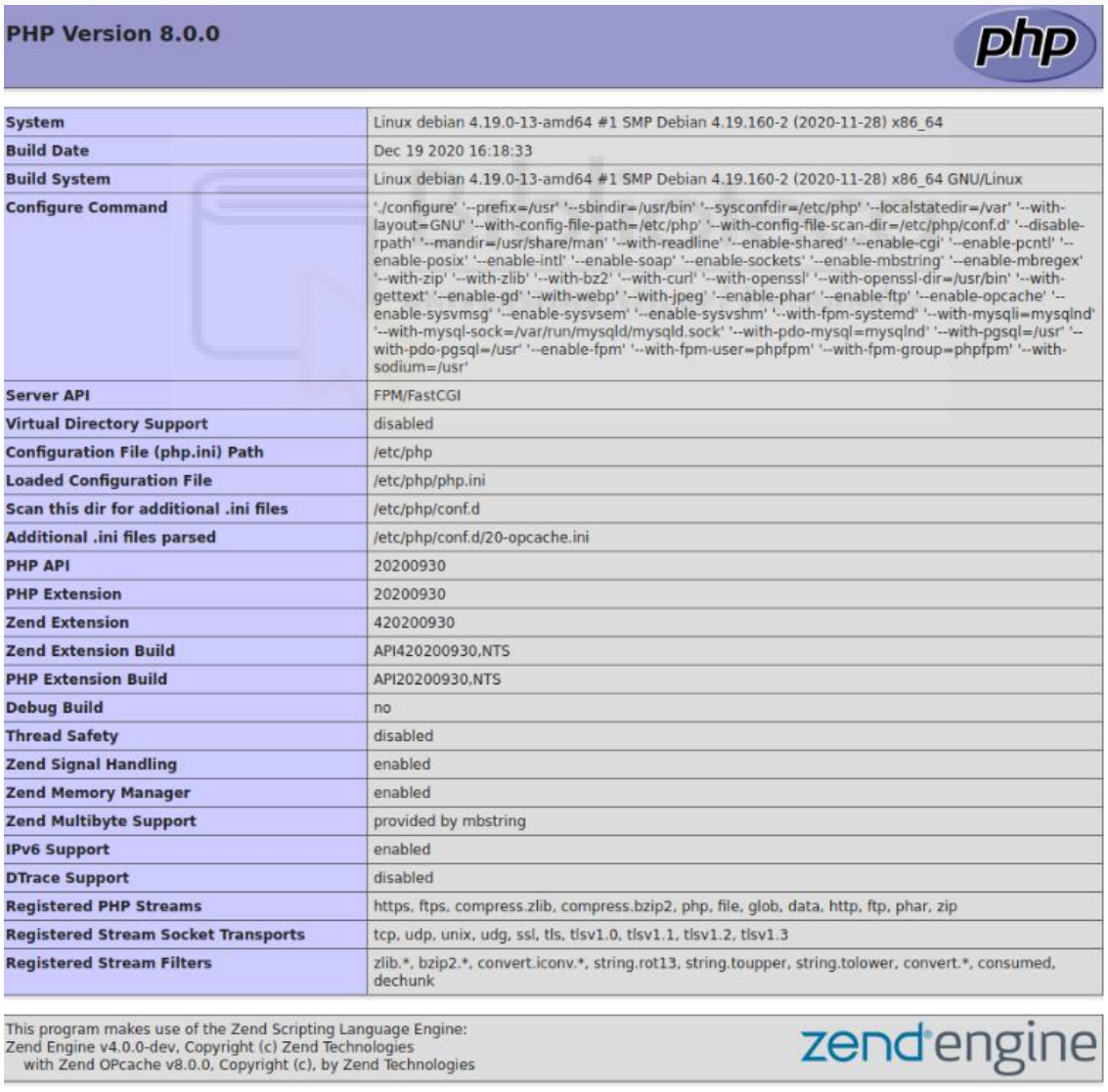
Para verificar que hemos instalado correctamente PHP editaremos el archivo **/var/www/html/info.php** mediante cualquiera de los editores disponibles en ubuntu.

```
> sudo nano /var/www/html/info.php
```

Dentro de este archivo generaremos el siguiente script en PHP.

```php
<?php
  phpinfo();
?>
```

Por último volvemos acceder desde el navegador a la dirección IP del servidor agregando como parámetro info.php, es decir, introducimos la ruta http://ip_servidor/info.php y nos debería aparecer la página de información de PHP que contiene la versión y las extensiones instaladas.



PHP Version 8.0.0

System	Linux debian 4.19.0-13-amd64 #1 SMP Debian 4.19.160-2 (2020-11-28) x86_64
Build Date	Dec 19 2020 16:18:33
Build System	Linux debian 4.19.0-13-amd64 #1 SMP Debian 4.19.160-2 (2020-11-28) x86_64 GNU/Linux
Configure Command	'./configure' '--prefix=/usr' '--sbindir=/usr/bin' '--sysconfdir=/etc/php' '--localstatedir=/var' '--with-layout=GNU' '--with-config-file-path=/etc/php' '--with-config-file-scan-dir=/etc/php/conf.d' '--disable-rpath' '--mandir=/usr/share/man' '--with-readline' '--enable-shared' '--enable-cgi' '--enable-pcntl' '--enable-posix' '--enable-intl' '--enable-soap' '--enable-sockets' '--enable-mbstring' '--enable-mbregex' '--with-zip' '--with-zlib' '--with-bz2' '--with-curl' '--with-openssl' '--with-openssl-dir=/usr/bin' '--with-gettext' '--enable-gd' '--with-webp' '--with-jpeg' '--enable-phar' '--enable-ftp' '--enable-opcache' '--enable-sysvmsg' '--enable-sysvsem' '--enable-sysvshm' '--with-fpm-systemd' '--with-mysqli=mysqlnd' '--with-mysql-sock=/var/run/mysqld/mysqld.sock' '--with-pdo-mysql=mysqlnd' '--with-pgsql=/usr' '--with-pdo-pgsql=/usr' '--enable-fpm' '--with-fpm-user=phpfpm' '--with-fpm-group=phpfpm' '--with-sodium=/usr'
Server API	FPM/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php
Loaded Configuration File	/etc/php/php.ini
Scan this dir for additional .ini files	/etc/php/conf.d
Additional .ini files parsed	/etc/php/conf.d/20-opcache.ini
PHP API	20200930
PHP Extension	20200930
Zend Extension	420200930
Zend Extension Build	API420200930,NTS
PHP Extension Build	API20200930,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	enabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring
IPv6 Support	enabled
DTrace Support	disabled
Registered PHP Streams	https, ftps, compress.zlib, compress.bzip2, php, file, glob, data, http, ftp, phar, zip
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, tls, tlsv1.0, tlsv1.1, tlsv1.2, tlsv1.3
Registered Stream Filters	zlib.*, bzip2.*, convert.iconv.*, string.rot13, string.toupper, string.tolower, convert.*, consumed, dechunk

This program makes use of the Zend Scripting Language Engine:
Zend Engine v4.0.0-dev, Copyright (c) Zend Technologies
with Zend OPcache v8.0.0, Copyright (c), by Zend Technologies

Ilustración 125. Página PHP INFO