UNIVERSIDAD SANTO TOMAS
PRIMER CLAUSTRO UNIVERSITARIO DE COLOMBIA
T U N J A

TÍTULO PROYECTO

MÓDULO DE ADMINISTRACIÓN DE TRABAJOS DE GRADO MAESTRÍA EN CALIDAD Y GESTIÓN INTEGRAL

PROPONENTE(S)

Juan Diego Gomez Gomez
1049656137
2251175

Jose Alejandro Garcia Preciado
1052414125
2251172

DIRECTOR

Carlos Andres Guerrero Alarcon

Tunja
23 de agosto de 2022

ÍNDICE DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1. FICHA TÉCNICA DEL PROYECTO

Título	Módulo de administración de trabajos de grado Maestría en Calidad y Gestión Integral
Nombre Estudiante	Juan Diego Gomez Gomez Jose Alejandro Garcia Preciado
Documento estudiante	1049656137 1052414125
Correo electrónico	juan.gomezg@usantoto.edu.co jose.garciap@usantoto.edu.co
Director	Carlos Andres Guerrero Alarcon
Lugar de ejecución del proyecto	Universidad Santo Tomás
Duración	7 meses
Costo	69.050.000$
Palabras claves	Arquitectura de Software, Vistas, Desarrollo Ágil, SDLC.

Los abajo firmantes confirman que todos los datos incluidos en la presente propuesta son correctos y verídicos, que no incumplen ninguna ley o norma vigente (incluir nombres y firmas de estudiantes y director).

Firma del autor
Jose Alejandro Garcia Preciado

Firma del autor
Juan Diego Gomez Gomez

Firma del director
Carlos Andres Guerrero Alarcon

2. PLANTEAMIENTO DEL PROBLEMA

El desarrollo de trabajos de grado a nivel de maestría tiene un alto nivel de complejidad, el cual requiere seguimiento y control para lograr el éxito del proyecto. Para la Universidad Santo Tomás es de vital importancia que los directores y futuros magister tengan una interacción en tiempo real, que permita formular el proyecto, hacer seguimiento y tener un control permanente de los avances de cada investigación.

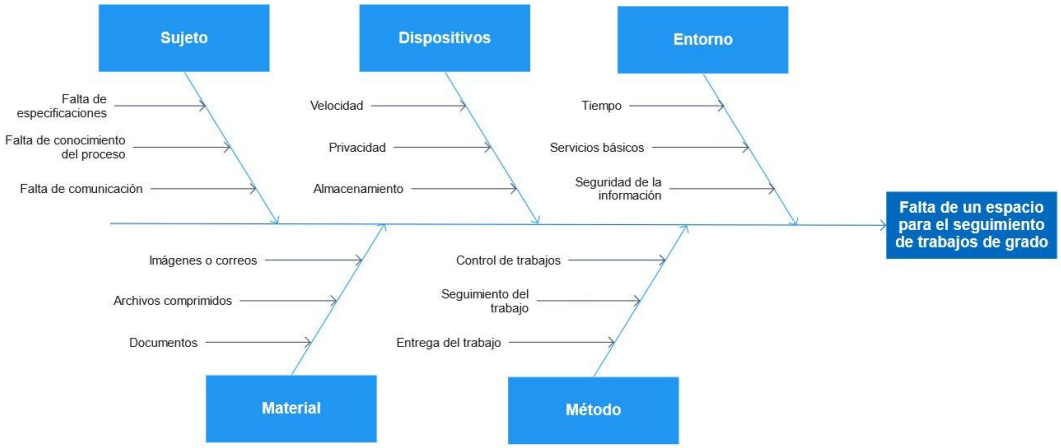
Independientemente del tipo de investigación que se esté realizando y de la cantidad de personas que interactúen en el proyecto, es necesario brindar los espacios y las herramientas necesarias para garantizar el desarrollo del proyecto. Actualmente se realiza el seguimiento y control de los trabajos de maestría de manera manual, con el incremento de la cantidad de trabajos tanto en ejecución como los trabajos entregados, se ha generado un alto nivel de dificultad a la hora de realizar seguimiento y consultar los proyectos finalizados.

La Universidad Santo Tomás cuenta con el recurso humano necesario para garantizar la dirección de los trabajos de grado. Los directores utilizan sus propias herramientas para hacer el seguimiento de los proyectos, sin embargo, para la universidad es un inconveniente la ausencia de una herramienta para tener un control detallado del proceso de avance de sus trabajos de maestría. Se han presentado casos en donde tanto director como estudiante han entregado excelentes trabajos en los tiempos estipulados, sin embargo, gran parte de trabajos de grado a nivel de maestría no cuentan con un seguimiento y control adecuados.

Lo anteriormente expuesto plantea una oportunidad para la construcción de una plataforma que permita apoyar el proceso de seguimiento de los trabajos de grado de la maestría en Calidad y Gestión Integral.

A continuación se presenta la Figura 1, en donde se encuentra el diagrama de problemas del presente proyecto.

Figura 1. Diagrama de Problemas



Fuente: Autor

En este diagrama se expone resumidamente lo problemática que tiene el proyecto a realizar.

3. JUSTIFICACIÓN

Dada la complejidad de los trabajos de maestría, es de vital importancia llevar un control de los avances de proyecto detallado, de tal manera que no haya pérdida de información entre cada avance presentado por el estudiante. Tener un control sobre los proyectos de investigación permitirá a la universidad, cumplir su promesa de valor en lo relacionado a los tiempos de desarrollo de las tesis de maestría.

Como se conoce los trabajos de grado se están entregando de manera manual, por lo que es conveniente realizar una plataforma para evitar labores repetitivos y consolidar toda la información de estos trabajos en un solo sitio, es decir, brindar una plataforma que tenga toda la información y herramientas necesarias para que tanto directores como estudiantes se vean beneficiados de esta.

Es importante mencionar que la idea de construir esta plataforma es evitar las tareas engorrosas que hasta el momento están haciendo los directores al recibir los trabajos, esto se debe a que tienen que utilizar herramientas propias, lo cual hace que el tema de realizar un seguimiento adecuado a un proyecto sea difícil. Por eso es mejor que tanto estudiantes como directores tengan un espacio donde se consolide todo lo que comprende el trabajo de grado de la maestría y sea de fácil acceso para todos los involucrados.

Cabe destacar que el proyecto a realizar tendrá un enfoque educativo donde la plataforma se construirá, diseñara e implementara con conceptos y técnicas aprendidas a lo largo de la carrera con el propósito de mejorar los métodos y espacios para las entregas,control y seguimiento de los trabajos de grado en la maestría en Calidad y Gestión Integral.

4. OBJETIVOS

4.1. Objetivo General

Construir una plataforma informática para la gestión y control de los avances de los proyectos de la maestría en Calidad y Gestión Integral de la Universidad Santo Tomás, utilizando herramientas de código abierto para el desarrollo de la plataforma y una arquitectura orientada a servicios para la definición de las capas del proyecto.

4.2. Objetivos específicos

A continuación se presentan los objetivos específicos del proyecto.

Tabla 1. Objetivos específicos

Nro.	Objetivo específico
1	Construir la especificación de requerimientos funcional del proyecto, utilizando la herramienta REM(Requirements management) propuesta por Amador Duran en su tesis doctoral.
2	Diseñar la arquitectura de la plataforma utilizando el modelo de vistas 4+1 propuesto por P. Kruchten, para definir los diseños requeridos en la fase de desarrollo.
3	Implementar una plataforma Web para el control y seguimiento de los trabajos de la maestría en Calidad y Gestión Integral de la Universidad Santo Tomás, incluyendo los siguientes módulos: • Autorización y autenticación. • Registro de trabajos de maestría. • Control de versiones de los trabajos entregados. • Asignación de recursos: Humano y otros. • Seguimiento de los avances del proyecto. • Consolidado de los trabajos de grado
4	Asegurar la funcionalidad de la plataforma Web por medio de pruebas controladas donde se enfocarán en técnicas para pruebas unitarias y de integración de aplicaciones.

Fuente: Autor

5. MARCO TEÓRICO

En este capítulo se presentan los fundamentos teóricos, en los cuales se basó el desarrollo de este proyecto.

5.1. Aplicaciones Web

Una aplicación web se comprende como una herramienta informática, la cual se enfoca en la comunicación con un usuario por medio de peticiones y respuestas; todo esto se lleva a cabo con el uso de un navegador. De acuerdo con Ollivier, S., & Gury, P. A. (2016) “Un sitio web clásico está compuesto por un conjunto de páginas por las que el usuario puede navegar. El servidor contiene la lógica de la aplicación, su rol es proporcionar las páginas que debe mostrar y reaccionar a las acciones del usuario, por ejemplo el envío de un formulario o el clic sobre un enlace de navegación. El rol del navegador es, simplemente, mostrar las páginas enviadas por el servidor y transmitirle las acciones del usuario”[15].
Todas las aplicaciones web tienen algo en común, y es su operatividad en cuanto a sus tecnologías, ya que todas necesitan definir una estructura para el contenido de sus componentes por medio del “Lenguaje de marcado de hipertexto” más conocido como HTML y su estilización por medio de las “Hojas de estilo en cascada” para hacer agradable el uso de estas aplicaciones; cabe aclarar que a la parte visual se le conoce como frontend. Otra parte de las aplicaciones web son las funcionalidades de fondo que tienen como guardar datos o archivos, entre otras cosas. Esto corresponde a el backend que dispone de bases de datos y lenguajes de programación.

5.1.1. HTML

El Lenguaje de Marcas de Hipertexto(HyperText Markup Language) es un componente de la Web, el cual define el significado y la estructura del contenido web. MDN contributors(2022) hace una aclaración donde mencionan que “Hipertexto hace referencia a los enlaces que conectan páginas web entre sí, ya sea dentro de un único sitio web o entre sitios web"[9].

5.1.2. CSS

Hojas de Estilo en Cascada (Cascading Style Sheets) es el lenguaje de estilos utilizado para describir la apariencia de los elementos HTML en la pantalla. Este posee estándares frente a la comunidad internacional del Consorcio World Wide Web(W3C).

5.1.3. SASS y SCSS

Se define a SASS como CSS con superpoderes, ya que Sass es un lenguaje de hojas de estilo compilado en CSS, a causa del manejo de variables, funciones y una forma única de organizar los CSS, para que su uso fomente la eficacia y facilidad de comprensión en los diseños[19].

5.1.4. JavaScript

Es un lenguaje de programación ligero, interpretado, o compilado justo-a-tiempo con funciones que pueden ser tratadas como cualquier otra variable. Si bien, es más conocido como un lenguaje de scripting (secuencias de comandos) para páginas web, y es usado en muchos entornos fuera del navegador, tal como Node.js.

JavaScript es un lenguaje de programación basada en prototipos que es “un estilo de programación orientada a objetos en el que las clases no se definen explícitamente, sino que se derivan de agregar propiedades y métodos a una instancia de otra clase o, con menos frecuencia, agregarlos a un objeto vacío” (MDN contributors, 2022), multiparadigma, de un solo hilo, dinámico, con soporte para programación orientada a objetos, imperativa y declarativa. Hoy en día es uno de los lenguajes de programación más importantes en el mundo de la informática, pues se utiliza en la gran mayoría de los sitios web modernos (Flanagan, 2011, p.1). Una ventaja de JavaScript es la incrustación rápida dentro de HTML. Además, como afirma Subramanian (2018, p. vi) , es uno de los lenguajes de programación más poderosos, extendidos y flexibles que existen actualmente.

5.1.5. TypeScript

TypeScript es un lenguaje de programación fuertemente tipado que se basa en JavaScript creado por Microsoft, lo que le brinda mejores herramientas a cualquier escala. A diferencia de JavaScript, TypeScript agrega sintaxis adicional a JavaScript para admitir

una integración más estrecha con su editor. Detecta errores tempranamente en su editor. Cabe aclarar que el código TypeScript se convierte a JavaScript, y se aplica en cualquier lugar donde se ejecute JavaScript: en un navegador, en Node.js o Deno y en sus aplicaciones. Además TypeScript entiende JavaScript y utiliza la inferencia de tipos para brindarle excelentes herramientas sin código adicional[12].

5.1.6. NPM

Según (Npm Inc., 2009) npm es “el administrador de paquetes de Node.js. Fue creado en 2009 como un proyecto de código abierto para ayudar a los desarrolladores de JavaScript a compartir fácilmente módulos de código empaquetados”.

5.1.7. Node.js

Es un entorno de ejecución multiplataforma para JavaScript como lenguaje script, el cual permite construir aplicaciones del lado del servidor y/o redes escalables con JavaScript. Su principal ventaja es que a la hora de gestionar peticiones puede atender multitud de estas, sin tener que recurrir a los hilos del sistema operativo. Por este motivo, es una gran opción a tener en cuenta a la hora de desarrollar sistemas escalables[16].

5.2. Frameworks

Se define como un diseño a gran escala donde describe cómo se descompone un programa en un conjunto de objetos y cómo estos interactúan. Usualmente se representa como un conjunto de clases abstractas y formas en las que interactúan sus instancias[7]. Dado que incluye las clases que interactúan, es el modelo colaborativo o patrón de interacción de objetos, tanto como lo son, los tipos de clases en el diseño.

La parte más importante de un framework es la forma en que un sistema se divide en diferentes componentes. Los frameworks también reutilizan la implementación, pero esto es menos importante, que la reutilización de las interfaces internas de un sistema y la forma en que sus funciones se dividen entre sus componentes.

Este diseño de alto nivel es el principal contenido intelectual del software, y los frameworks son una forma de volver a utilizarlo.

5.2.1. Angular

Angular es uno de los frameworks de JavaScript de código abierto más usados actualmente en el mercado del mundo web, debido a que fue creado por Google en el año 2008 con el nombre de AngularJS[1].

AngularJS fue uno de los primeros frameworks enfocado Single-Page Applications(SPAs), que como su palabra lo indica son aplicaciones de una sola página, lo cual fue una mejora para el rendimiento de respuestas al servidor y tiempo de respuesta al usuario al mostrar nuevas pestañas o secciones de la página[15].

En 2014 AngularJS fue relanzado con el nombre de Angular 2 que fue un framework completamente reescrito, en 2016 se le cambió el nombre a Angular[11]. Con el paso del tiempo una de las mayores fortalezas de Angular fue la importante introducción de TypeScript como transpirador de código en JavaScript sin formato en el momento de compilación. Además como muchos otros frameworks, está basado en componentes; esto significa que los componentes son los principales bloques de construcción. Pueden mostrar información, general plantillas y realizar acciones en los datos.

5.2.1.1 Angular Command Line Interface

Angular CLI se podría definir como la “columna vertebral” del framework, cabe aclarar que es una herramienta para gestionar los proyectos de Angular a lo largo de todo el ciclo de vida de una aplicación. Sirve como un generador de código que simplifica en gran medida el proceso de creación de nuevos proyectos, así como el proceso de generación de nuevos componentes, servicios y rutas en una aplicación existente. También puede utilizar Angular CLI para crear paquetes de código para el desarrollo y la implementación de la producción. Angular CLI no solo generará un proyecto base para el desarrollador, sino que también instalará el framework Angular y todas sus dependencias[13].

5.2.2. Express

Es una infraestructura web rápida, minimalista y flexible, que proporciona un conjunto sólido de funciones para aplicaciones web y móviles en Node.js. En general, se prefiere “Express” a “Express.js”, aunque esta también se acepta[17].

Una de las principales características de Express es que ”proporciona una capa delgada de características fundamentales de la aplicación web, sin oscurecer las características de Node.js que conoce y ama” (OpenJS, 2010).

5.3. Librerías

Se define librerías, según Cutiño & EdTeam (2020) “es un conjunto de funcionalidades, que resuelven necesidades específicas del proyecto, empaquetadas y reutilizables”.

5.3.1. PrimeNG

Es una librería desarrollada por PrimeTek y es usada para la construcción de aplicaciones web enfocadas en la parte visual usando como base Angular, cabe mencionar que es una librería open source. El beneficio de usar PrimeNG es construir aplicaciones con el menor esfuerzo posible, ya que se enfoca en productividad y velocidad para los desarrolladores. Las aplicaciones construidas perfectamente pueden ser distribuidas en dispositivos móviles y entorno de escritorio como sitios web[18].
PrimeNG cuenta con unos componentes que enriquecen la experiencia del usuario al interactuar con una aplicación web.

5.3.2. JSON Web Token (JWT)

JSON Web Token [2] es un estándar definido en la RFC-7519 [8] para intercambiar información en formato JSON entre dos interlocutores de forma segura. Esta información va almacenada en el payload de una estructura JSON Web Signature, o en el plaintext de una estructura JSON Web Encryption, permitiendo que la información vaya firmada con un código de autenticación de mensaje (MAC por sus siglas en inglés) o bien encriptada.

5.4. Sequelize(PG)

Sequelize es un *Object-Relational Mapping(*ORM) moderno de TypeScript y Node.js para Postgres, MySQL, MariaDB, SQLite y SQL Server, y más. Con un sólido soporte de transacciones, relaciones, carga ansiosa y diferida, replicación de lectura y más[21].

5.5. Herramientas y recursos

5.5.1. Visual Studio Code

Es un editor de código redefinido y optimizado para crear y depurar aplicaciones web y en la nube[23].

5.5.2. PowerDesigner

Es una herramienta de modelización de datos, donde mezcla excepcionalmente diferentes técnicas de modelización (modelo conceptual tradicional, físico y lógico con una modelización única de inteligencia de negocios y de traslado de datos) para juntar análisis de negocios con soluciones formales de diseño de base de datos[24].

5.5.3. PostgreSQL

Es un sistema de base de datos relacional de objetos de código abierto que utiliza y amplía el lenguaje SQL combinado con muchas características que almacenan y escalan de forma segura las cargas de trabajo de datos más complicadas[25].

5.5.4. Figma

Es una aplicación de edición de gráficos y diseño de interfaz de usuario basada en la web. Puede usarlo para realizar todo tipo de creación de wireframes, diseño de interfaces de aplicaciones móviles y web, y creación de prototipos de diseños[26].

5.5.5. Postman

Es una plataforma API para construir y usar API. Postman simplifica cada paso del ciclo de vida de la API y agiliza la colaboración para que pueda crear mejores API, más rápido[27].

5.5.6. Git - GitHub

Git es un sistema de control de versiones distribuido de código abierto diseñado para gestionar proyectos. Permite organizar las distintas versiones del código del proyecto y documentar todos los cambios realizados a lo largo del desarrollo. A diferencia de GitHub que es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git[28].

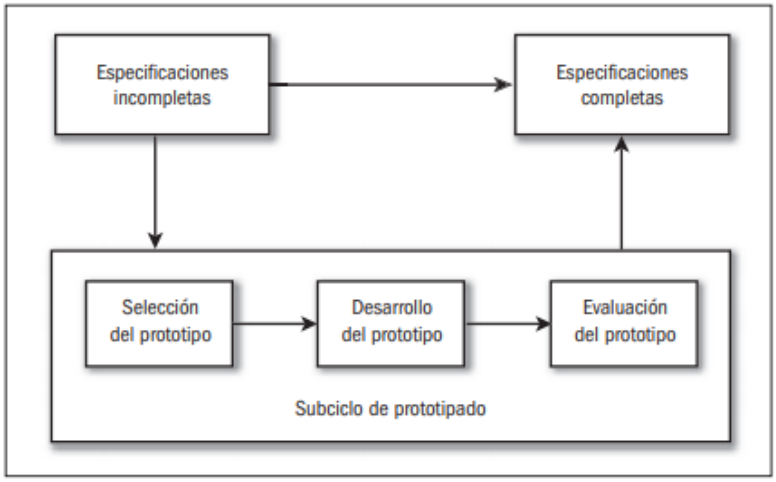
6. MODELO DE DESARROLLO

En cuanto a la metodología del proyecto “Módulo de administración de trabajos de grado Maestría en Calidad y Gestión Integral”, es importante definir el tipo de investigación y el ciclo de vida del software para poder comenzar. Con respecto al tipo de investigación del proyecto podemos definirla como la investigación cuantitativa según Roberto Hernandez Sampieri(2014) en su libro “Metodología de la investigación” en su sexta edición, donde establece que los alcances de una investigación depende totalmente a los objetivos expuestos por el proponente, en donde este tendrá que seleccionar el tipo de alcances que puede tener un estudio cuantitativo; en este proyecto podemos concluir que el tipo de investigación es exploratorio, es decir, que es un estudio donde no sea realizado ningún otro tipo de investigación y parte desde una base mínima de conocimiento del tema[29].

Para el desarrollo de este proyecto es necesario definir un modelo o metodología para el ciclo de vida de un proyecto, por eso al hacer una búsqueda investigativa, tenemos que encontrar la mejor opción que se tiene para el desarrollo ágil con altas posibilidades de éxito.

Debido a esto se eligió el modelo de ciclo de vida por prototipos, más conocido como ciclo de vida prototipado; en este modelo el objetivo se basa en lograr un producto intermedio, antes de llegar a una versión final, donde por medio de prototipos se desarrollan todas las funcionalidades previstas para el producto final. A continuación en la Figura 2, se presenta la simplicidad del modelo en la forma de transicionar entre los requerimientos iniciales a los finales.

Figura 2. Modelo de Prototipos



Fuente: https://ingsw.pbworks.com/f/Ciclo+de+Vida+del+Software.pdf

Al determinar el modelo de ciclo de vida, es necesario definir las fases de Software Development life cycle (SDLC) que serán las siguientes:

1. Recopilación y análisis de requisitos: En esta fase es la que brinda una imagen de claridad frente a los alcances de todo el proyecto, metas, problemas y oportunidades que desencadenaron en el proyecto. Se usará Requirements Management(REM) herramienta conocida como un entorno metodológico de Ingeniería de Requisitos para sistemas de información, presentada por Amador Durán.
2. Estudio de viabilidad: Este proceso se llevó a cabo con la ayuda del documento 'Especificación de requisitos de software' también conocido como documento 'SRS'. Incluye todo lo que debe diseñarse y desarrollarse durante el ciclo de vida del proyecto, donde se evidencia en cinco puntos importantes que son: Económico, legal, viabilidad de la operación, técnico y tiempo y calendarios.
3. Diseño: Se hará una fase de construcción de wireframes y mockups(prototipos hechos en la marcha del proyecto), para definir el diseño, colores y la interacción del diseño, con el fin de llegar a la fase de codificación y plasmar las ideas aprobadas por el encargado del proyecto, respetando los colores de la universidad.

4. Codificación: En esta fase se definió el uso de la herramienta del desarrollo Angular para el proyecto, dado que es un framework ágil que ahorra tiempo en el desarrollo, este trae varias ventajas a la hora de hacer un desarrollo web. Como entorno de ejecución se utilizará Nodejs, debido a que permite la creación de aplicaciones web rápidas con múltiples conexiones simultáneas; esto se traduce en una alta escalabilidad.
5. Prueba: Realizar pruebas sobre el aplicativo web con datos reales de trabajos de grado que figuren en la base de datos de la universidad, testear el funcionamiento óptimo del módulo administrativo con tiempos de respuesta y evaluar el cumplimiento de los requerimientos.
6. Instalación / Implementación: Desplegar el módulo administrativo en un dominio web y así poder hacer uso del mismo.

7. DESARROLLO DEL PROYECTO

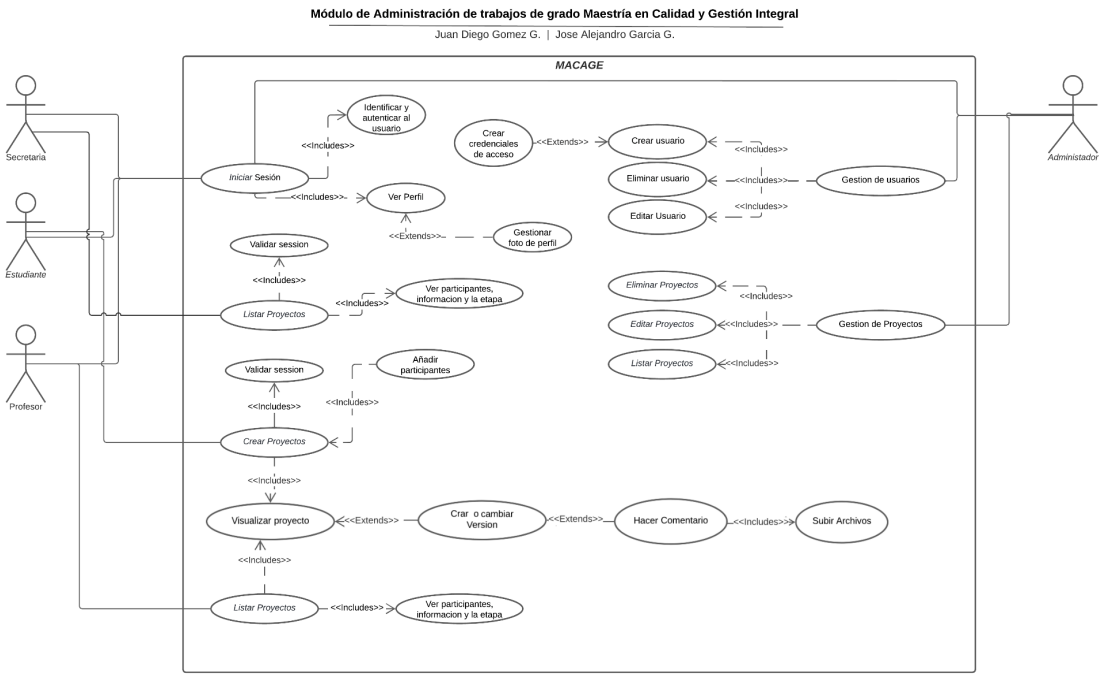
En esta sección, se detalla cada una de las actividades de las fases del proyecto.

7.1. Fase 1. Recopilación y análisis de requisitos

En la fase de Recopilación y análisis de requisitos se realizaron una serie de actividades, que se pueden evidenciar en el Anexo 1, donde se realizó la gestión de requerimientos según la tesis doctoral de Amador Duran [30].

Una vez se elaboró la construcción de Especificación de Requisitos de Software(ERS), dio como resultado una fase preliminar de arquitectura de un sistema de software, donde se realizó un diagrama de casos de uso para darle relación, según lo escrito en la especificación de los requisitos [Anexo 1]. A continuación en la Figura 3 se dará un vistazo al diagrama de casos de uso.

Figura 3. Diagrama casos de uso



Fuente: Autores

Como se puede observar se definieron cuatro actores fundamentales en el proyecto, los cuales se definirán más adelante según avance el documento.

Tabla 2. Fase 1. Recopilación y análisis de requisitos

Tipo	Elementos

Entradas	• Narrativa del proyecto. • Descripción del proyecto.
Instrumentos y/o herramientas	• Computador. • Conexión a internet. • Navegador Web. • Google Docs.
Salidas	Anexo 1. Documento de Gestión de Requerimientos
Tiempo	4 semanas

Fuente: Autor

7.2. Fase 2. Estudio de viabilidad

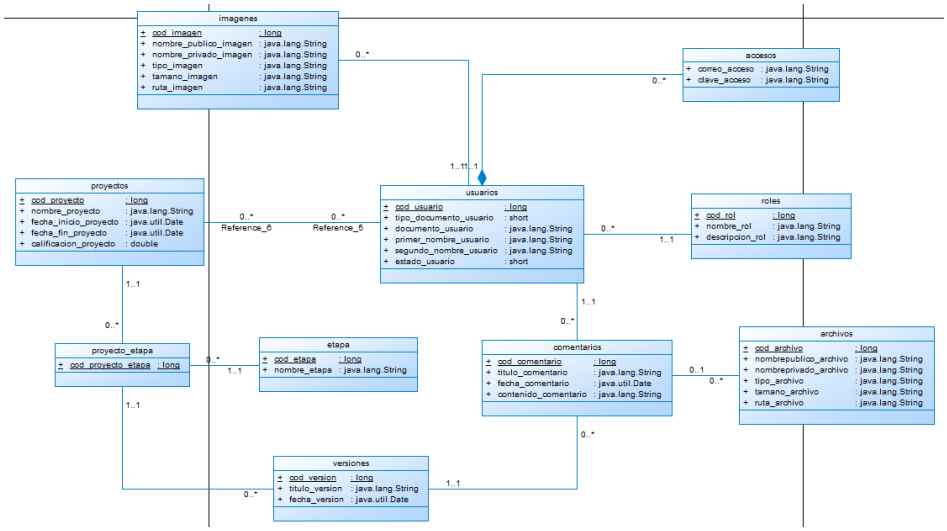
Para darle una viabilidad a un producto fue necesario definir una arquitectura para poder abarcar el proyecto desde diferentes aspectos, por lo que se hizo la selección de la arquitectura de "modelos de vistas 4+1" propuesto por P. Kruchten en 1995 [31], con esta se definió que un sistema de software se debe documentar por medio de cuatro vistas bien diferenciadas y que a su vez estas cuatro vistas se han de relacionar entre sí con una vista más, que es la denominada vista "+1". Estas cuatro vistas las denominó Kruchten como: vista lógica, vista de procesos, vista de despliegue, vista física y la vista "+1", que tiene la tarea de relacionar las cuatro vistas citadas, y a esta última la denominó vista de escenario.

A continuación se muestran las vistas del diseño arquitectónico del proyecto.

7.2.1. Vista Lógica:

Representa la funcionalidad que el sistema proporcionará a los usuarios. Es decir, se ha de representar lo que el sistema debe hacer, y las funciones y servicios que ofrece. A continuación la siguiente figura ilustra el diagrama que corresponde a la vista lógica.

Figura 4. Diagrama de clases

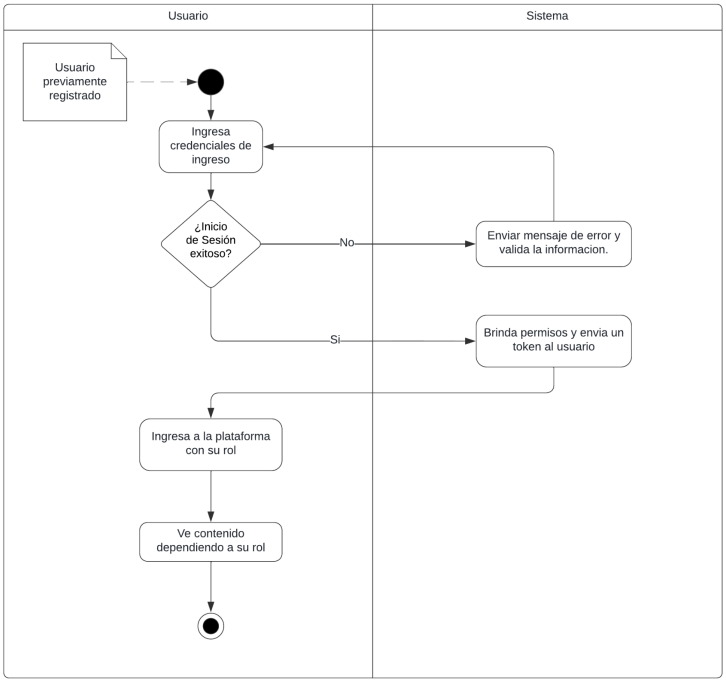


Fuente: Autores

7.2.2. Vista de Procesos

Permite ver los procesos que existen en el sistema y la forma en la que se comunican estos; describe el flujo de trabajo paso a paso de negocio y operacionales que conforman el sistema.

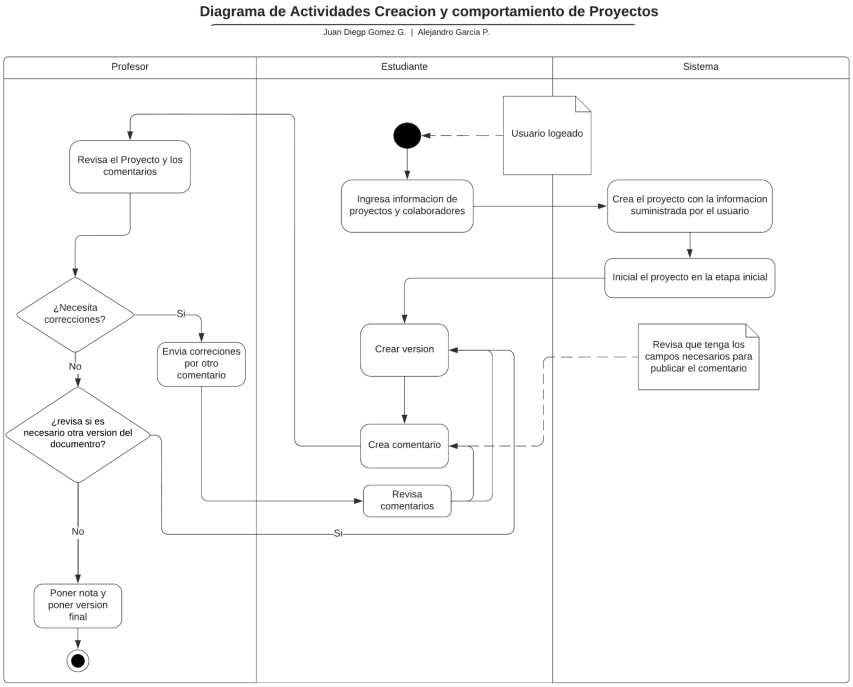
Figura 5. Diagrama de Actividad Inicio de Sesión



Fuente: Autores

En la Figura 5 podemos observar cómo será la funcionalidad de inicio de sesión, los pasos a seguir por el usuario y cómo operará la plataforma.

Figura 6. Diagrama de Actividad Creación y comportamiento de proyectos



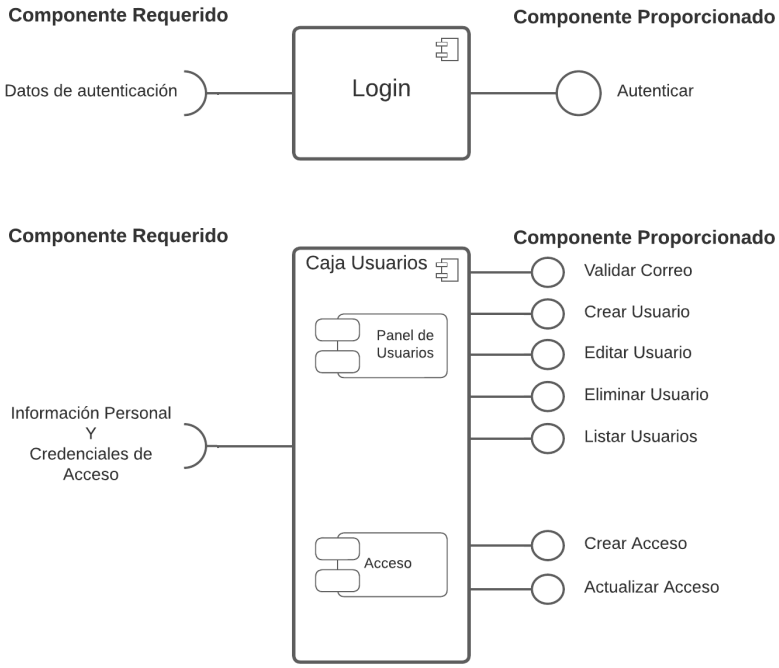
Fuente: Autores

En la Figura 6, se detalla la operatividad que tendrá nuestra aplicación con los diferentes roles involucrados en la creación de proyectos.

7.2.3. Vista de Despliegue

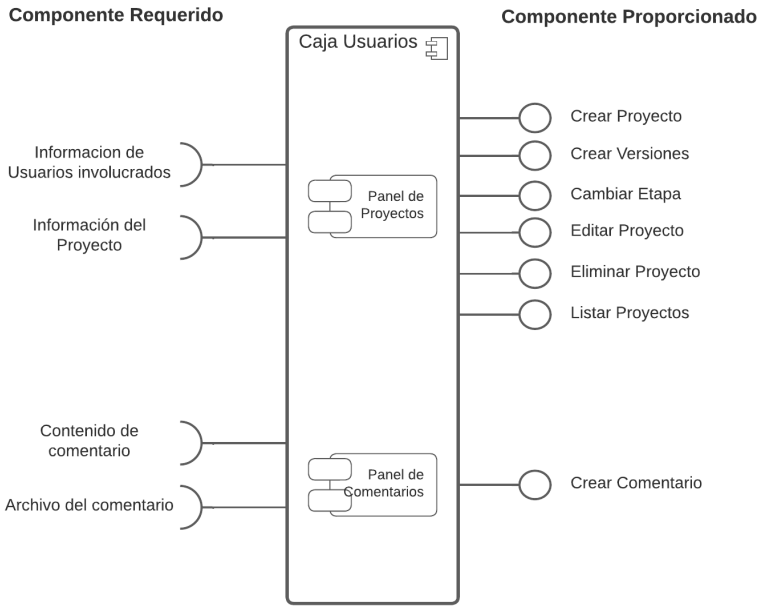
Muestra el sistema desde la perspectiva de *un desarrollador*; o en otras palabras, se va a mostrar cómo está dividido el sistema software en componentes y las dependencias que hay entre estos [Figura 7, Figura 8]

Figura 7. Diagrama de componentes de login y usuarios



Fuente: Autores

Figura 8. Diagrama de componentes de proyecto

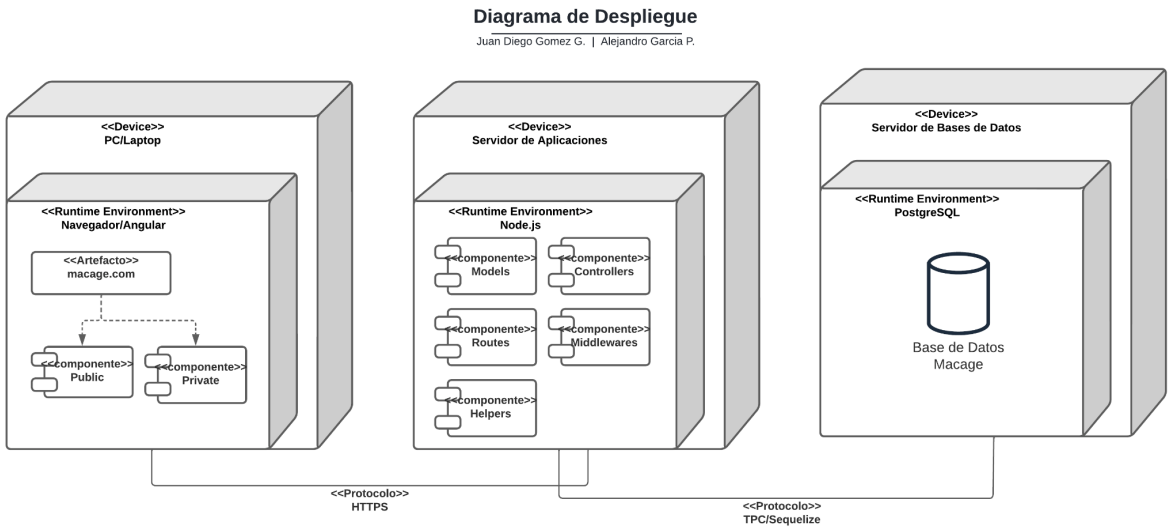


Fuente: Autores

7.2.4. Vista Física

Se toma desde la perspectiva de nodos y cómo se conforman todos los componentes del sistema, así como las conexiones entre estos componentes que conforman la solución.

Figura 9. Diagrama de despliegue



Fuente: Autores

En la Figura 9 se muestra el funcionamiento de la aplicación de sus nodos y la comunicación entre estos, teniendo en cuenta lo anterior podemos hacer la definición de la vista “+1”.

7.2.5. “+1” Vista de Escenarios

Esta vista es la representación del diagrama de Casos de uso[Figura 3] y tiene la función de unir y relacionar las otras cuatro vistas.

Luego de definir estas vistas tenemos claridad en la arquitectura del software; el paso a seguir es establecer las tecnologías a usar, el tiempo de desarrollo y sobre todo, la viabilidad de la implementación del proyecto “Módulo de administración de trabajos de grado Maestría en Calidad y Gestión Integral”.

Tabla 3. Fase 2. Estudio de viabilidad

Tipo	Elementos
Entradas	• Anexo 1. Documento de Gestión de Requerimientos

Instrumentos y/o herramientas	• Computador. • Conexión a internet. • Navegador Web. • Google Docs. • The 4+1 View Model of architecture
Salidas	• Diagrama de Clases • Diagrama de Actividades • Diagrama de componentes • Diagrama de Despliegue • Diagrama de Casos de Uso
Tiempo	4 semanas

Fuente: Autor

7.3. Fase 3. Diseño del sistema

El desarrollo de la fase 3 de diseño del sistema está compuesto por dos partes: el diseño de la base de datos y el diseño preliminar.

7.3.1. Diseño de la base de datos

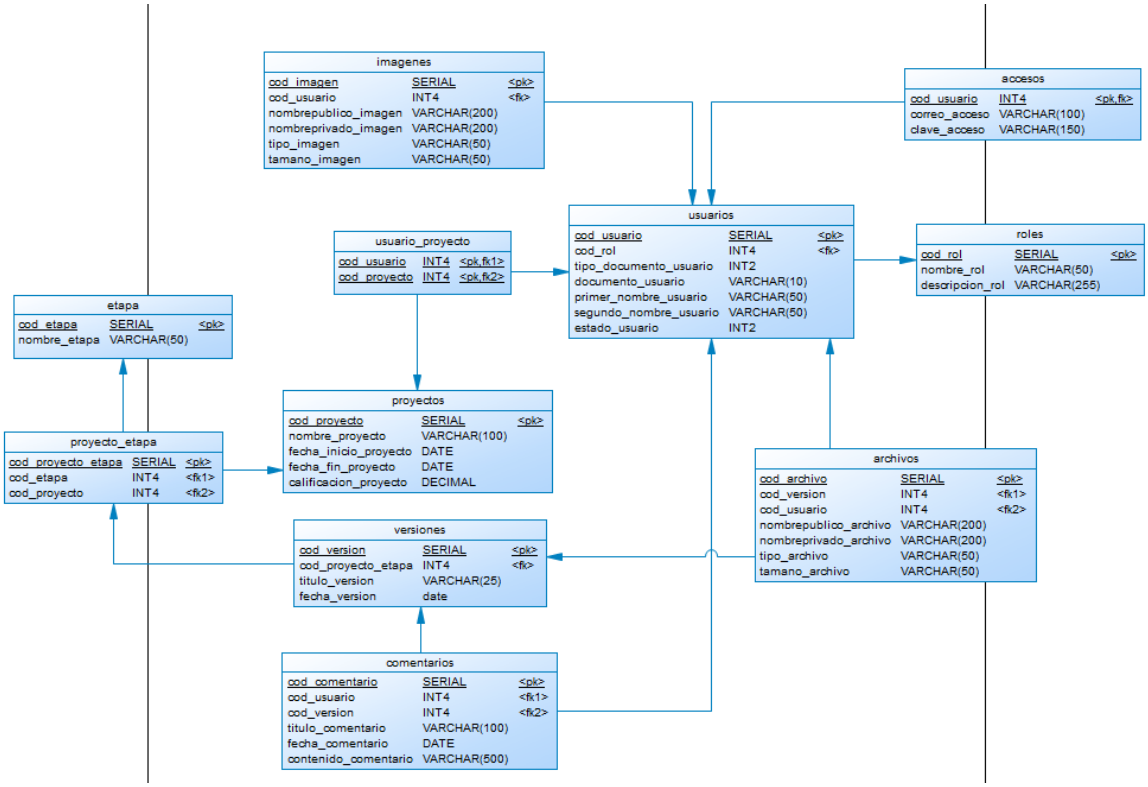
Para el diseño de la base de datos se usó la herramienta Power Designer [24] para realizar el modelo de datos físicos, según la información adquirida en la primera fase (Fase 1. Recopilación y análisis de requisitos) y en la segunda fase, se definió una arquitectura con una perspectiva real de lo que se requería para desarrollar este modelo de datos. Este modelo de entidad–relación de la base de datos tuvo como versión final once tablas que son:

- roles: Roles que existen dentro de la aplicación definidos como Administrador, Estudiante, Profesor y Secretaria.
- usuarios: Información de usuarios.
- accesos: Credenciales de acceso para cada usuario registrado.
- imagenes: Imágenes de perfil de cada usuario y modificación de las mismas.
- usuario_proyecto: Tabla de rompimiento para unir usuarios con proyectos.
- proyectos: Información de los proyectos.
- etapas: Estado de los proyectos.
- proyecto_etapa: Tabla de rompimiento para definir estados a los proyectos.
- Versiones: Versión del proyecto para el manejo de comentarios.
- comentarios: Observaciones a definir según cada proyecto.

- archivos: Archivos vinculados a usuarios y comentarios dentro de las versiones de los proyectos.

En la versión anterior al modelo final se observa en la Figura 10, que no cumplió las funcionalidades exigidas en la Fase 1 del proyecto. Debido a tener una falla en la relación de las tablas “archivos” y “versiones”, ya que resultaba poco práctico el uso de los comentarios dentro de un proyecto.

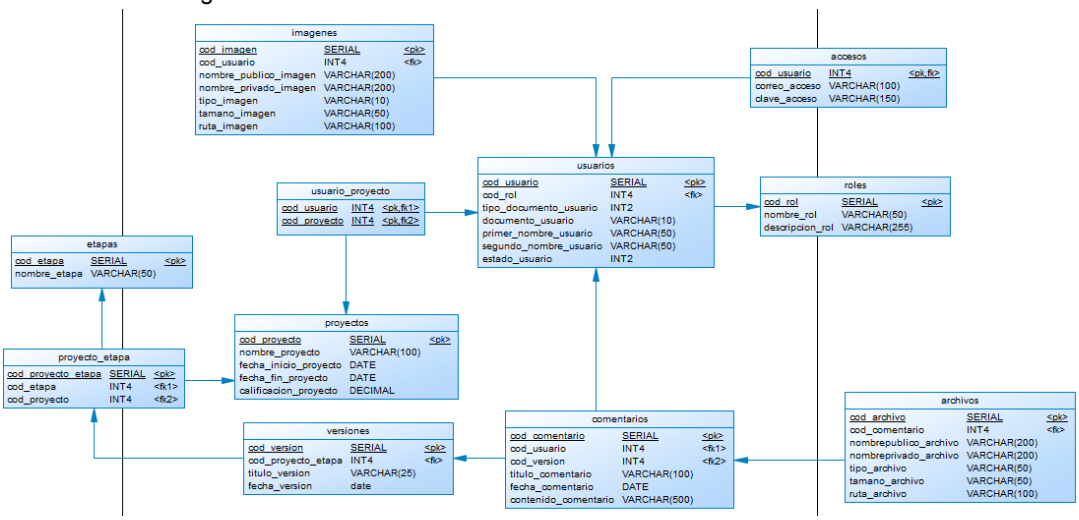
Figura 10. Diseño de base de datos Versión 1



Fuente: Autores

Por lo tanto se decidió cambiar el modelo a una versión 2(final) que se representa en la Figura 11.

Figura 11. Diseño de base de datos Versión 2



Fuente: Autores

Luego de los cambios realizados en el modelo, tuvo como resultado una reducción de tiempo al desarrollo, ya que simplifican la forma de revisión de proyectos tanto para un profesor como para un estudiante. Además de cambiar relaciones se hicieron unos ajustes en algunas tablas como es el caso de la tabla “Comentario” que se añadió el campo de “título_comentario” para tener mejor control de la usabilidad del proyecto.

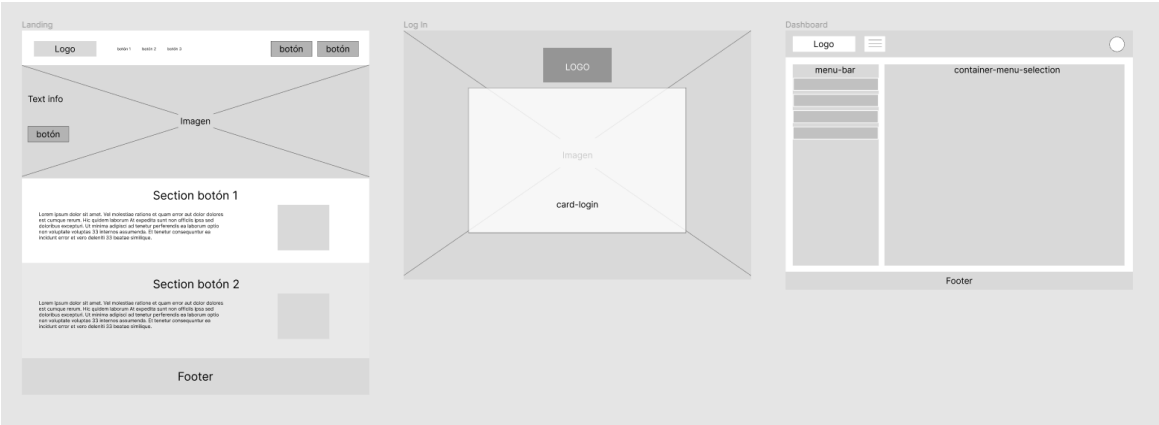
Adicionalmente se realizaron configuraciones y mejoras generales como las siguientes:

- Nombres de tablas y campos en español con una estructura de uso de minúsculas y guión bajo donde se use más de una palabra, por ejemplo, proyecto_etapa.
- Constraints:
    - Llaves primarias: Todas las tablas tienen llave primaria “cod_nombre_de_la_tabla” de tipo entero autoincremental.
    - Campos únicos: Se validó que determinados campos no permitan registros repetidos por ejemplo el correo y el documento de un usuario.
    - Llaves foráneas: Uso de DELETE CASCADE para eliminar padres e hijos de manera indiscriminada para evitar errores en acciones del usuario o administrador del sitio.
- Campos nulos y obligatorios: Según las narrativas dadas en el Anexo 1 se dejaron campos obligatorios y no obligatorios, por ejemplo cod_comentario en la tabla “archivos”.

7.3.2. Diseño preliminar

Para la parte del diseño preliminar se hizo uso de la herramienta Figma, ya que se basó en los principios de usabilidad, por ello se decidió mantener el estilo de Single Page Application(SPA), que aprendimos a lo largo de los últimos semestres con el uso de Angular[11]. Debido a esa razón tenemos que el diseño del wireframe fue el siguiente[Figura 12]:

Figura 12. Wireframe Macage.fig

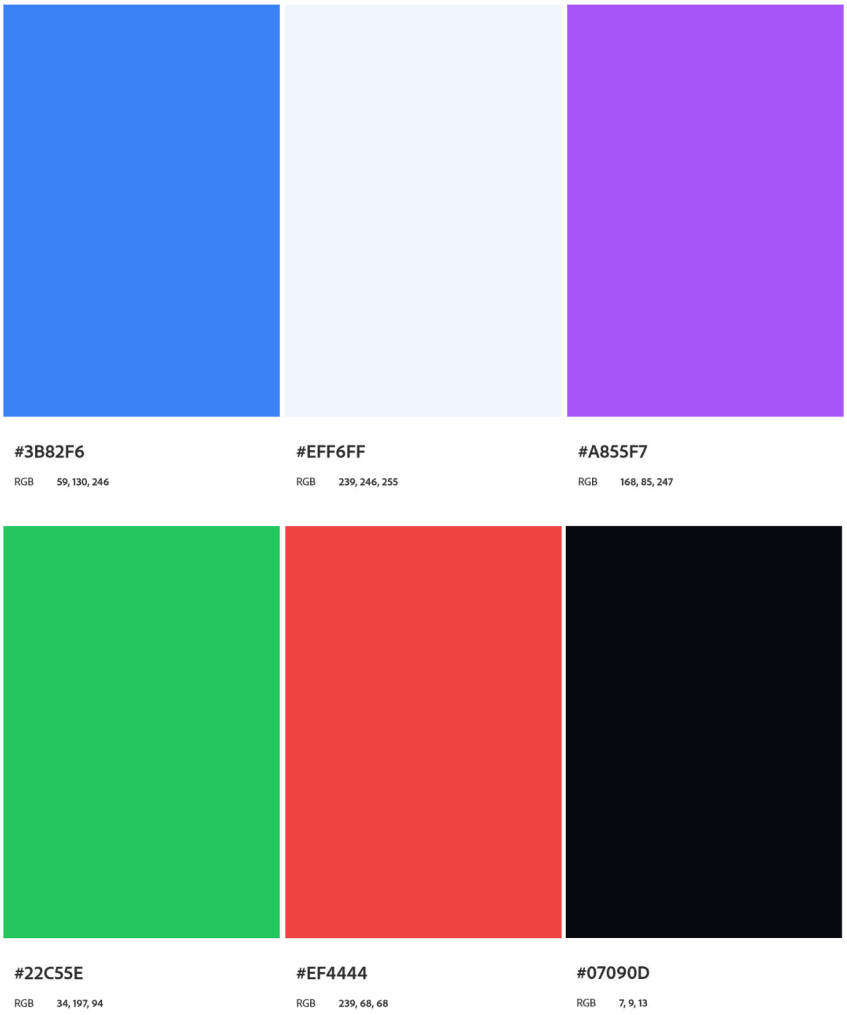


Fuente: Autores

Cabe mencionar que se escogieron tres partes esenciales del wireframe que fueron una landing page más conocida como *Home*, el *Login* y el *Dashboard.* Tenemos que el Dashboard es la vista que tiene más trabajo debido a que no se quería hacer muchos cambios en la interfaz, para que el usuario no tuviera complejidad a usar la aplicación; por esa misma razón decidimos que era muy bueno el concepto de SPA, ya que no habían muchos cambios en la plantilla y que no era necesario recargar la página para cambiar elementos o hacer diferentes tipos de peticiones.

En la selección de colores, se escogieron conforme a los colores de la universidad y las sensaciones que evoca el uso del color.

Figura 13. Paleta de colores



Fuente: Autores

Tabla 4. Definición de colores

Color	Función
Morado#A855F7	Evoca creatividad
Blanco#EFF6FF	Evoca pureza, inocencia y positividad y ayuda a depurar el diseño
Azul#3B82F6	Evoca calma, confianza, honestidad
Verde#22C55E	Evoca seguridad y tranquilidad
Rojo#EF4444	Evoca energía, como también puede significar peligro y destrucción.
Negro#07090D	Evoca elegancia, oscuridad y la noche.

Fuente: Autores

Estos colores representados en la Tabla 4 se utilizaron para distintas partes del proyecto. El azul se empleó para botones, representación de selecciones y títulos principales, el

blanco se empleo para el fondo de la aplicación y para el texto de los botones, el negro empleo para el texto de la aplicacion y algunos campos en los formularios, el morado se empleo para algunos botones, el verde se empleo para botones de confirmacion y el rojo se utilizó para representar errores y botones de eliminación.

Tabla 5. Fase 3. Diseño del sistema

Tipo	Elementos
Entradas	• Anexo 1. Documento de Gestión de Requerimientos • Diagramas de Arquitectura 4+1
Instrumentos y/o herramientas	• Computador. • Conexión a internet. • Navegador Web. • Google Docs. • Figma. • PowerDesigner
Salidas	• Diseño entidad-relación de la base de datos • Diseño visual preliminar del Software
Tiempo	8 semanas

Fuente: Autores

7.4. Fase 4. Desarrollo de la aplicación

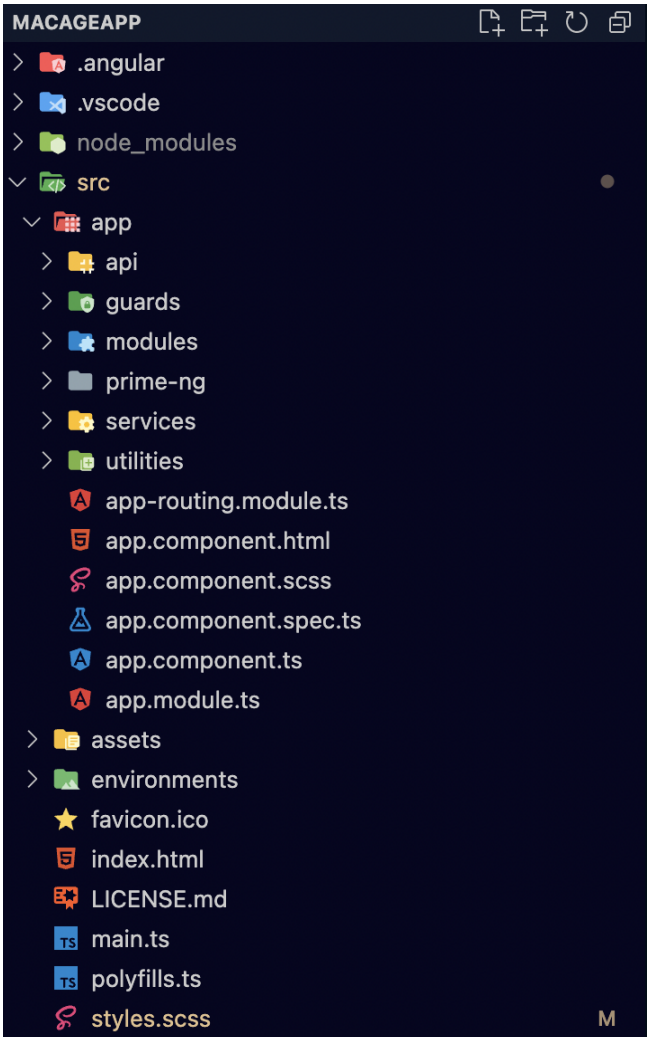
El desarrollo de la Fase 4 se dividió en dos partes importantes que fueron: Frontend y Backend.

7.4.1. Desarrollo del Frontend

Teniendo en cuenta la Fase 2 y la Fase 3 se determinó el uso de Angular con su Command Line Interface [13] como framework del frontend y para hacer más ágil la creación de interfaces se optó por usar la librería de PrimeNg [18], donde se determinó usar una plantilla proporcionada por PrimeNg, en esta plantilla existen variedad de componentes y estilos, lo que ahorrará tiempo y trabajo al momento de codificar las interfaces.

Se definió la estructura de la aplicación gracias a PrimeNG y Angular CLI, Se crearon *módulos, guardianes, servicios y utilidades* [Figura 14].

Figura 14. Estructura de MacageApp



Fuente: Autores

7.4.1.1 Desarrollo de Interfaces

Como primera vista del proyecto tenemos un landing page, que se define como ***home***, donde da inicio a todo el proyecto. Cabe mencionar que a cada usuario del sistema se le presentará una serie de interfaces acorde al rol que posea. A continuación en la Figura 15 se mostrará la vista definida como ***Home***.

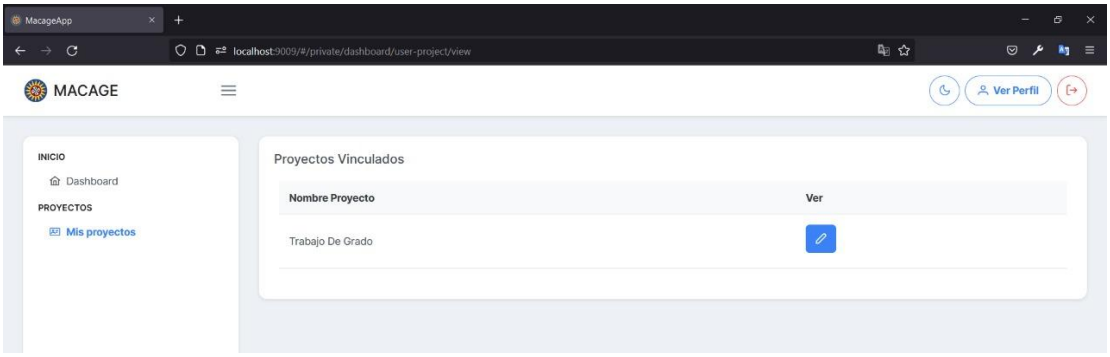
Figura 15. Vista Home

Fuente: Autores

Una vez en el home podemos ver las herramientas usadas en el proyecto, antes de ingresar a la aplicación se debe aclarar las interfaces compartidas para todos los usuarios que son: *Inicio de Sesión* y *Perfil.* Luego de estas interfaces compartidas se especifican las interfaces por roles.

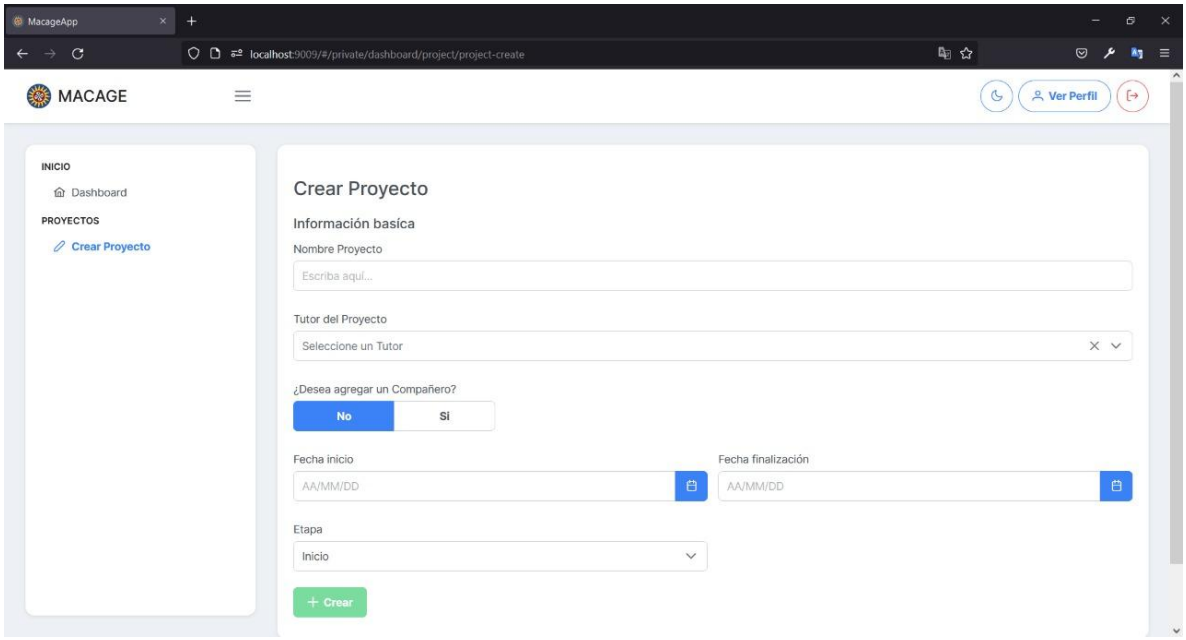
1. Interfaces rol *Estudiante*:
    - **Proyectos:** Esta interfaz le permite al estudiante ver los proyectos que ha creado, que por lo general, no supera la cantidad de un proyecto, en caso de que el estudiante no tenga proyectos creados la aplicación le permitirá crear un proyecto, donde el podrá agregar a los participantes y un profesor tutor; esto se podrá observar en las Figuras 16 y Figura 17.

Figura 16. Interfaz Estudiante con proyecto creado



Fuente: Autores

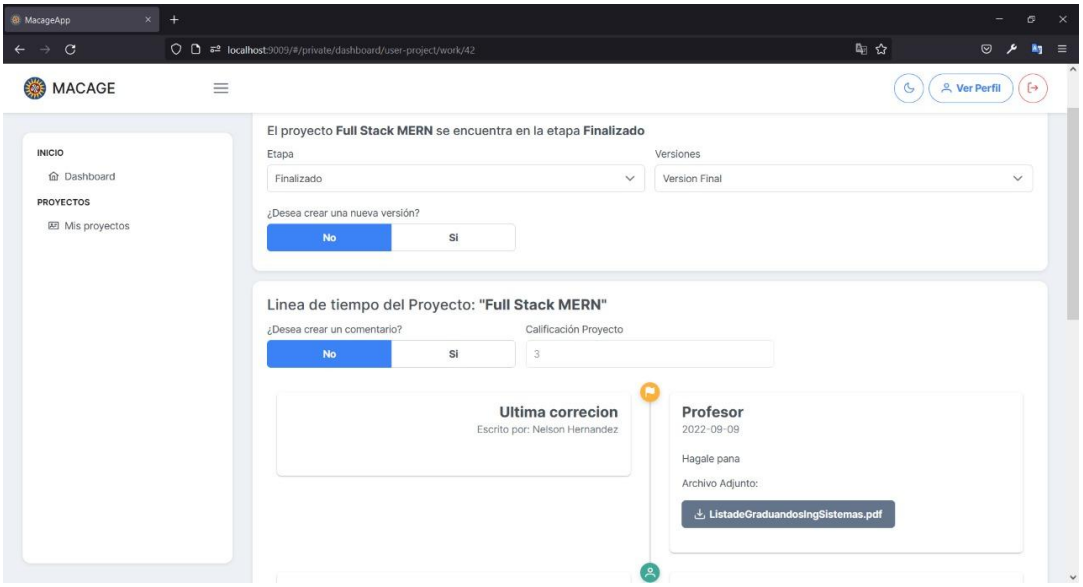
Figura 17. Interfaz Estudiante con proyecto sin proyectos creados



Fuente: Autores

Además de eso una vez se cree el proyecto, el usuario podrá hacer comentarios y subir archivos a estos para que el profesor pueda revisar sus avances y realizar correcciones [Figura 18].

Figura 18. Interfaz Estudiante dentro del proyecto

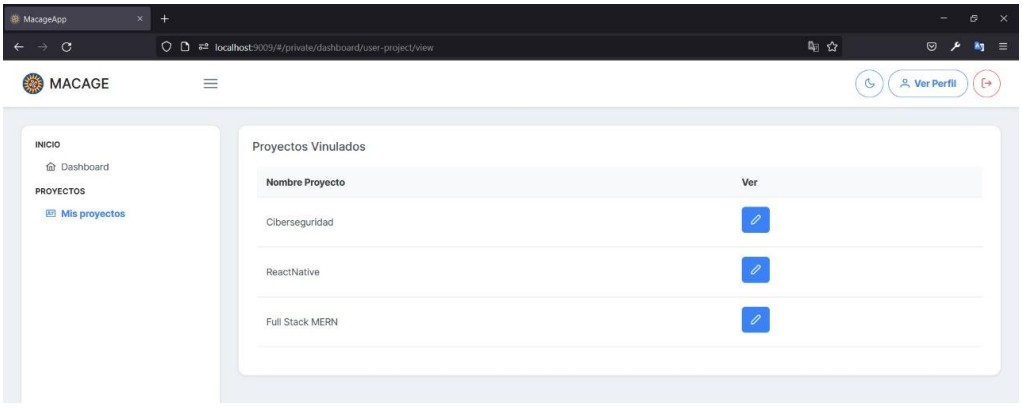


Fuente: Autores

2. Interfaz rol *Profesor*:

- **Mis proyectos:** Esta interfaz le permite al docente ver todos proyectos en los que ha sido vinculado, de tal manera que puede realizar comentarios y cambiar de versiones.

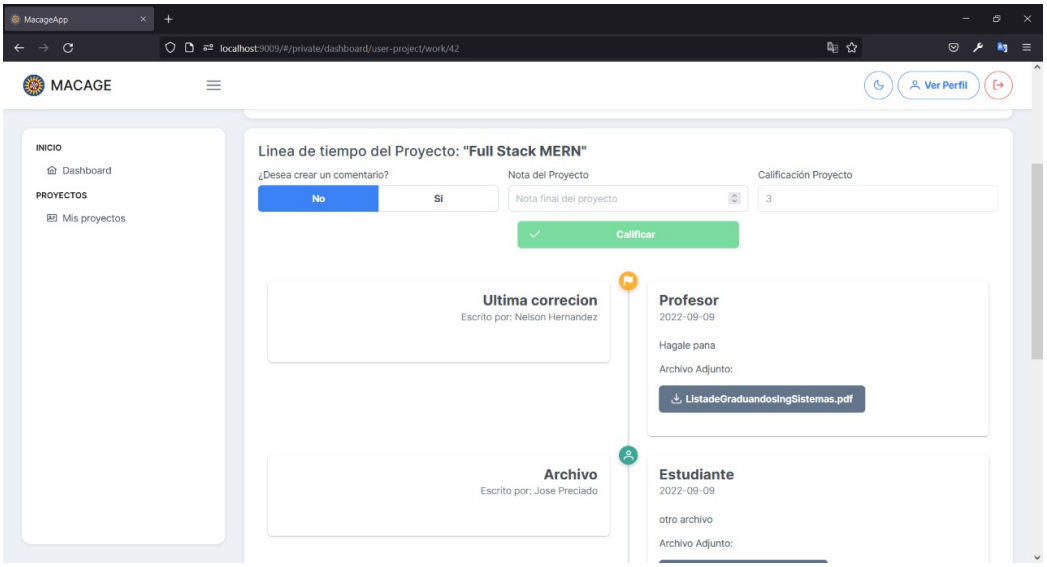
Figura 19. Interfaz Profesor



Fuente: Autores

Al profesor se le agregó una funcionalidad para calificar los proyectos desde la misma vista dentro del proyecto, esto hace que la interfaz sea más intuitiva para su uso[Figura 20].

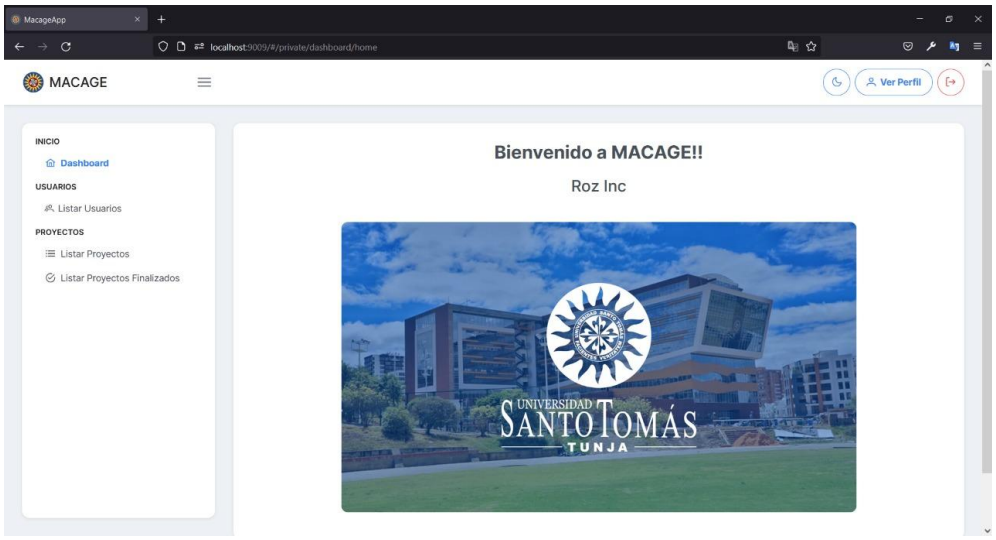
Figura 20. Vista Profesor dentro del proyecto



Fuente: Autores

3. Interfaces rol *Secretaria*:
    - **Proyectos:** Esta interfaz le permite a la secretaría consultar información de los proyectos que están en desarrollo.
    - **Proyectos culminados:** Esta interfaz le permite a la secretaría consultar información de los proyectos que están en la etapa de culminados.
    - **Usuarios:** Esta interfaz le permitirá ver los usuarios en el sistema.

Figura 21. Vista Secretaria

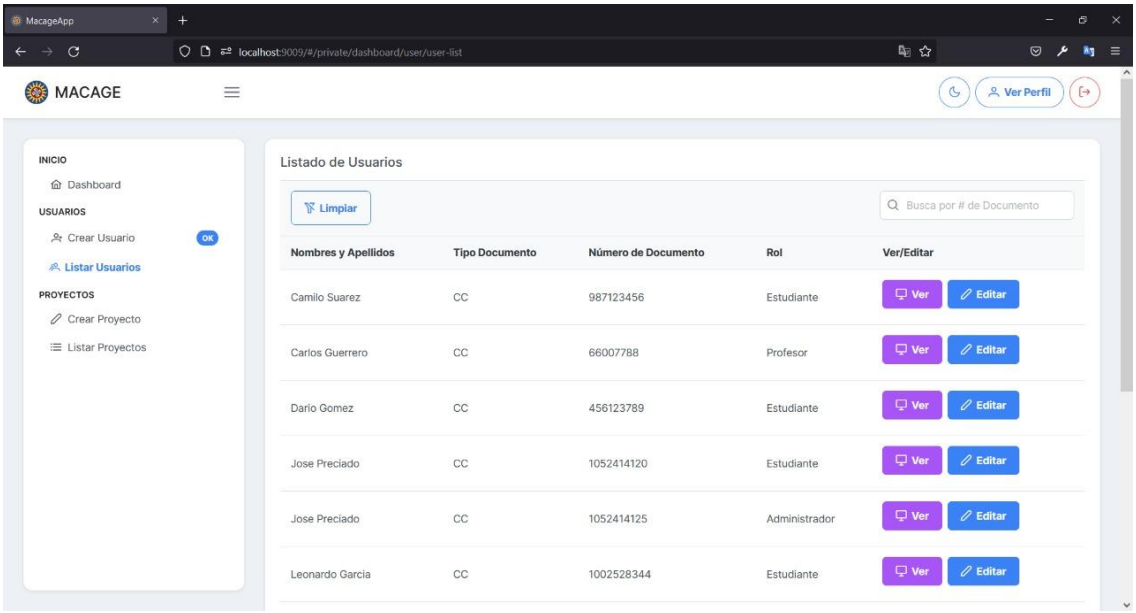


Fuente: Autores

4. Interfaces rol *Administrador*:

- **Módulo de usuarios:** En este panel de administración se podrá listar, editar y eliminar usuarios según la situación lo necesite. Cabe destacar que el administrador es el único que puede crear usuarios y brindarles acceso a la aplicación.

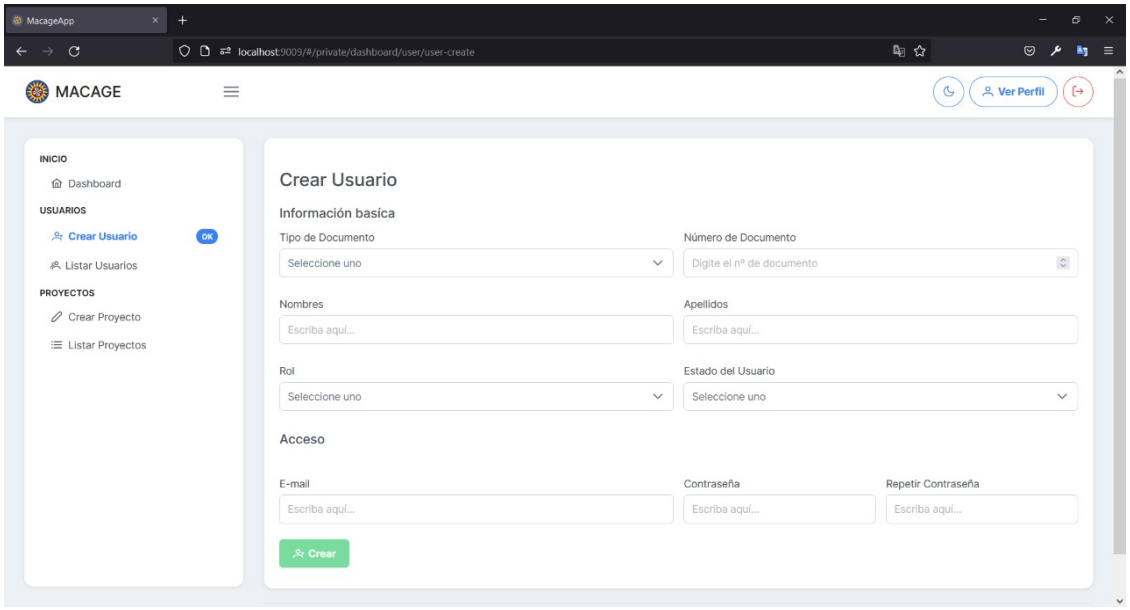
Figura 22. Panel de administración de usuarios



Fuente: Autores

En otra parte del menú tenemos la forma de crear usuarios como se muestra en la siguiente figura.

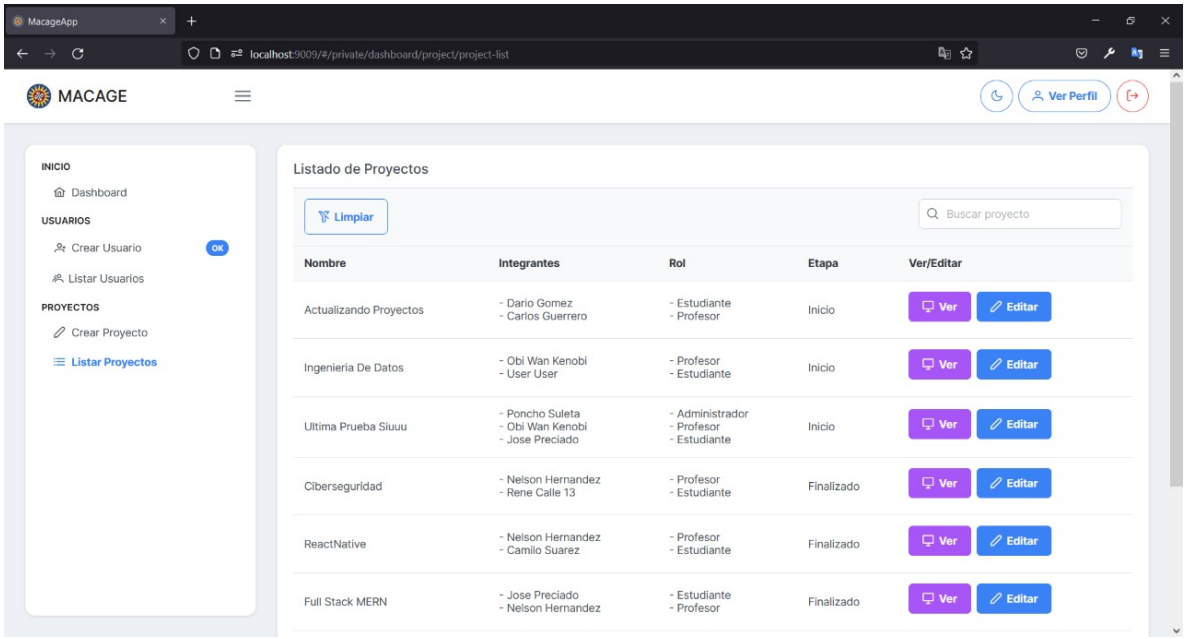
Figura 23. Interfaz de crear usuario



Fuente: Autores

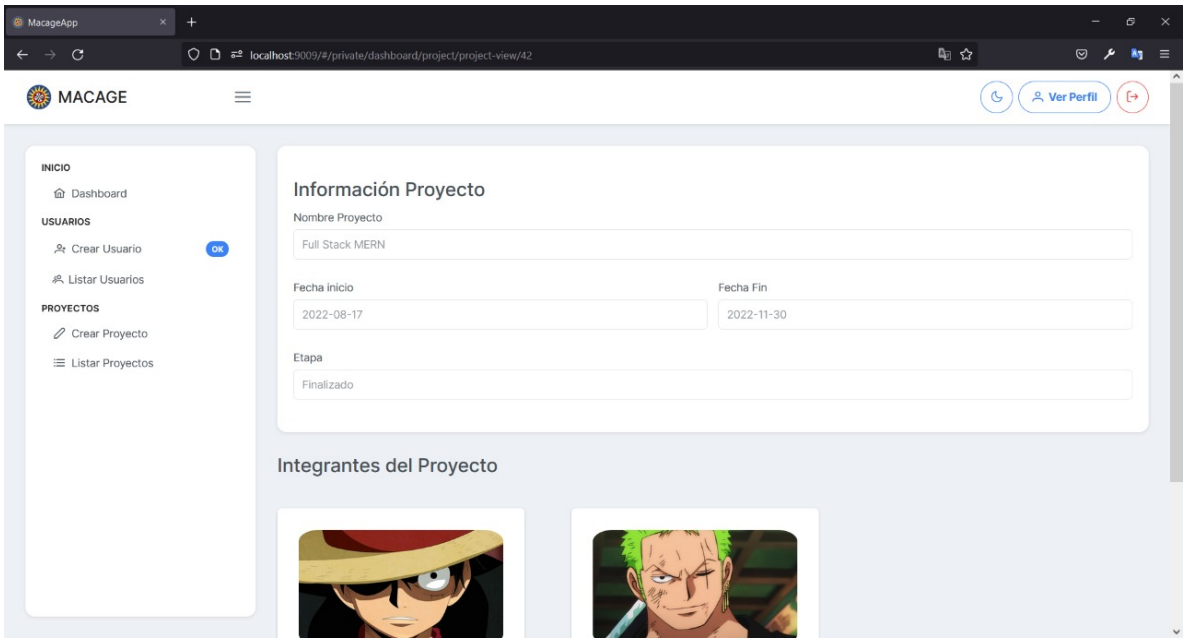
- **Módulo de proyectos:** En esta interfaz el administrador podrá listar, editar y eliminar proyectos del sistema[Figura 24, Figura 25, Figura 26].

Figura 24. Panel de administración de proyectos



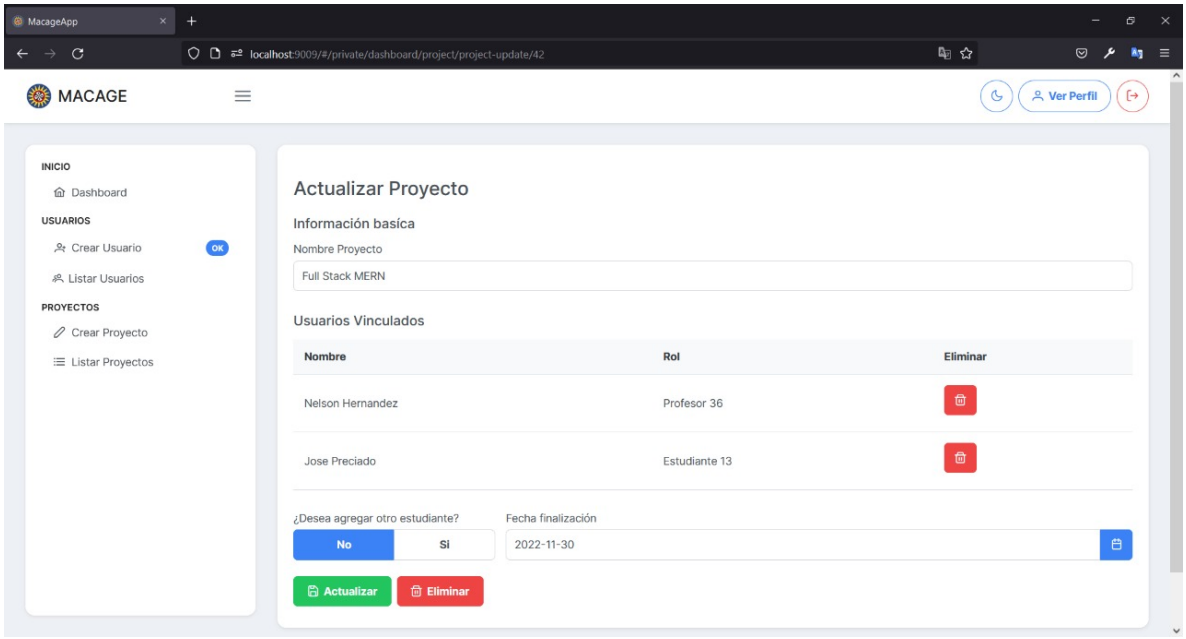
Fuente: Autores

Figura 25. Interfaz Ver proyecto



Fuente: Autores

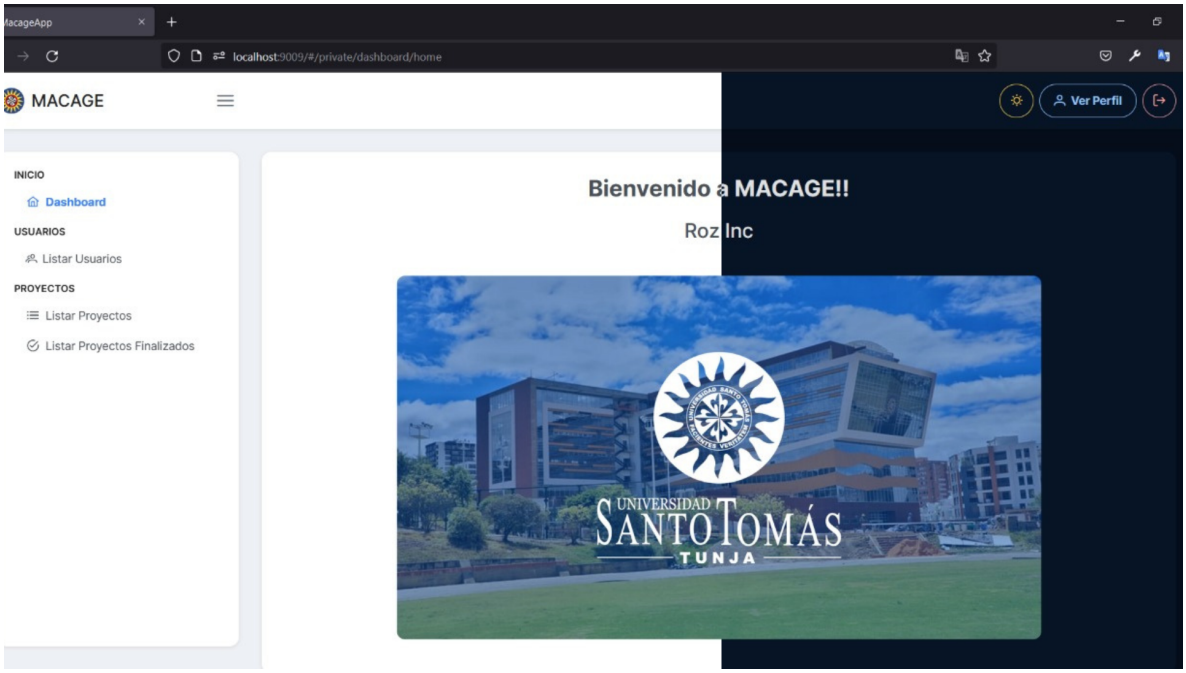
Figura 26. Interfaz Editar o eliminar proyecto



Fuente: Autores

Aparte de las interfaces existentes decidimos hacer algo para la experiencia de usuario y fue la implementación del modo oscuro de la aplicación de tal manera que podemos observar las diferencias en la Figura 27

Figura 27. Interfaz light vs Dark



Fuente: Autores

7.4.1.2 Consumo de API's

Para esta parte en específico se mostrarán algunos ejemplos de cómo fue el consumo de las API REST que nos proporciona el backend de una forma flexible.

Figura 28. Servicio de autenticación de usuario

```
import { HttpClient } from '@angular/common/http';      alejandroG13, 2 month
import { Injectable } from '@angular/core';
import { catchError, map, of, tap, Observable } from 'rxjs';
import { AuthResponse, Usuario } from 'src/app/api/interfaces/interfaces';
import { URL_API } from '../url';

alejandroG13, 2 months ago | 1 author (alejandroG13)
@Injectable({
  providedIn: 'root',
})
export class AuthService {
  private baseUrl: string = URL_API;

  constructor(private http: HttpClient) {}

  login(correo_acceso: string, clave_acceso: string) {
    const url = `${this.baseUrl}/auth/login`;
    const body = { correo_acceso, clave_acceso };

    return this.http.post<AuthResponse>(url, body).pipe(
      tap((resp) => {
        if (resp.ok) {
          localStorage.setItem('token', resp.tokenSession!);

          localStorage.setItem('rol', JSON.stringify(resp.rol));

          localStorage.setItem('usuario', JSON.stringify(resp.usuario));
        }
      }),
      map((resp) => resp.ok),
      catchError((err) => of(err.error.msg))
    );
  }
}
```

Fuente: Autores

En este caso es “auth.service” que es el servicio implementado en el frontend consumiendo un endpoint del backend, en donde se crea la sesión del usuario con su respectivo Json Web Token(JWT)[2], este JWT junto la información del usuario se almacena en el localstorage del navegador. Por otra parte tenemos un guardián llamado “validar-token.guard”, el cual valida la seguridad del sitio, evitando saltos en la seguridad o poder ingresar al dashboard sin el token generado para cada usuario.

Figura 29. Guardian validar token

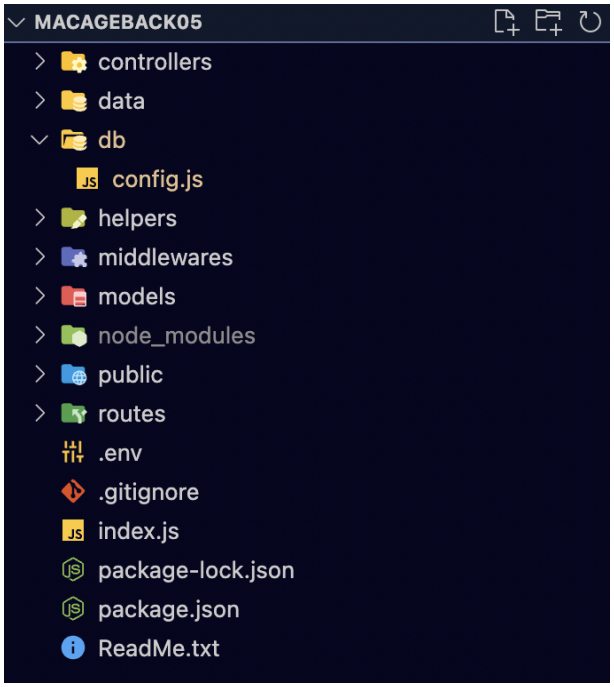
```
import { Injectable } from '@angular/core';      alejandroG13, 2 months a
import { CanActivate, CanLoad, UrlTree, Router } from '@angular/router';
import { Observable } from 'rxjs';

alejandroG13, 2 months ago | 1 author (alejandroG13)
@Injectable({
  providedIn: 'root',
})
export class ValidarTokenGuard implements CanActivate, CanLoad {
  constructor(private router: Router) {}
  canActivate(): Observable<boolean | UrlTree> | boolean {
    const token = localStorage.getItem('token');
    let validate: boolean =  true;
    if (!token) {
      this.router.navigate(['welcome/home']);
      validate = false
    }
    return validate;
  }
  canLoad(): Observable<boolean | UrlTree> | boolean {
    const token = localStorage.getItem('token');
    let validate: boolean =  true;
    if (!token) {
      this.router.navigate(['welcome/home']);
      validate = false
    }
    return validate;
  }
}
```

Fuente: Autores

7.4.2. Desarrollo del Backend

Se determinó emplear un lenguaje de programación y framework que facilita la implementación y desarrollo ágil de este tipo de APIs, por lo que se usó el lenguaje de programación JavaScript y el framework Express con Node.js. La definición de la estructura del Backend se logró a través del uso de Express [17] por medio de *archivos de configuración, modelos, controladores, middlewares y rutas*[Figura 30].

Figura 30. Estructuración del Backend



Fuente: Autores

A continuación se especifican las partes del backend representadas en la figura anterior, las cuales son esenciales para su funcionamiento.

7.4.2.1 Creación del servidor y conexión con la base de datos

Para esta sección se hizo uso de express para establecer la inicialización del servidor y su conexión de la base de datos. Además se puede observar en la Figura 31 todas las rutas usadas en los servicios que se le provee al frontend.

Figura 31. Index.js

```
const express = require("express");
const cors = require("cors");
const { sequelize } = require("./db/config");
require("dotenv").config();

//Crear el servidr/aplicación de express
const app = express();

//Directorio Publico
app.use(express.static("public"));

//CORS
app.use(cors());

//Lectura y parseo del body
app.use(express.json());

//Rutas
app.use("/api/usuario", require("./routes/usuarios.routes"));
app.use("/api/acceso", require("./routes/acceso.routes"));
app.use("/api/auth", require("./routes/auth.routes"));
app.use("/api/proyecto", require("./routes/proyecto.routes"));
app.use("/api/rol", require("./routes/roles.routes"));
app.use("/api/etapa", require("./routes/etapas.routes"));
app.use("/api/valid", require("./routes/validate.routes"));
app.use("/api/up", require("./routes/up.routes"));
app.use("/api/comentario", require("./routes/comentario.routes"));
app.use("/api/storage", require("./routes/archivos.routes"));

async function main() {
  try {
    await sequelize.sync({ force: false });
    app.listen(process.env.PORT);
    console.log("Server listening on port", process.env.PORT);
  } catch (error) {
    console.error("Unable to connect to the database:", error);
  }
}

main();
```

Fuente: Autores

Para hacer exitoso la conexión a la base de datos fue necesario crear un archivo de configuración, donde se hace el uso del ORM Sequelize [21] con PostgreSQL, este archivo tiene información clave como: el nombre de la base de datos, usuario de la base de datos y su contraseña [Figura 32]. También se define el host y el dialecto que en este caso es PostgreSQL [25].

Figura 32. Config.js

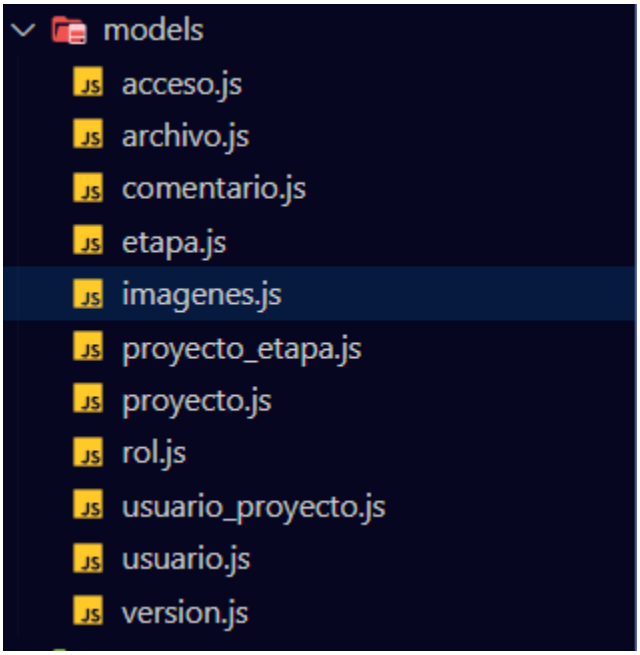
```
const sequelize = new Sequelize("railway", "postgres", "mswIMsMQQHS3z9DJ6Knq", {
  host: "containers-us-west-41.railway.app",
  dialect: "postgres",
  port: 6593,
  define: {
    freezeTableName: true,
  },
});

module.exports = { sequelize };
```

Fuente: Autores

7.4.2.2 Definición del modelo Entidad-Relación

Como se planteó anteriormente en la Fase 3(Diseño del sistema) con el diseño de la base de datos [Figura 11], se mapean las tablas existentes en la base de datos, para poder manipular los datos. Estas clases serán las siguientes:

Figura 33. Clases del Backend



Fuente: Autores

A continuación se muestra una de las clases creadas en relación de la base de datos, para esto se utilizó Sequelize [21] para realizar el mapeo de campos.

Figura 34. Clase archivo

```
const { DataTypes } = require("sequelize");
const { sequelize } = require("../db/config");
const { Comentario } = require("./comentario");

const Archivo = sequelize.define(
  "archivos",
  {
    cod_archivo: {
      type: DataTypes.INTEGER,
      primaryKey: true,
      autoIncrement: true,
    },
    nombre_publico_archivo: {
      type: DataTypes.STRING(200),
    },
    nombre_privado_archivo: {
      type: DataTypes.STRING(200),
    },
    tipo_archivo: {
      type: DataTypes.STRING(10),
      allowNull: false,
    },
    tamano_archivo: {
      type: DataTypes.STRING(50),
      allowNull: false,
    },
    ruta_archivo: {
      type: DataTypes.STRING(100),
      allowNull: false,
    },
  },
  {
    timestamps: false,
  }
);

//Relacion archivo_usuario
Comentario.hasOne(Archivo, {
  foreignKey: "cod_comentario",
  sourceKey: "cod_comentario",
  allowNull: true,
});

Archivo.belongsTo(Comentario, {
  foreignKey: "cod_comentario",
});

module.exports = { Archivo };
```

Fuente: Autores

Para la integración de estas clases en otras partes del proyecto se importa como una constante en JavaScript de la siguiente manera `const { Comentario } = require("./comentario");`, los demás modelos o clases se podrán ver a detalle en el Anexo 5.

7.4.2.3 Creación de APIs

Para hacer la creación de las API del proyecto, estas constan de tres partes esenciales que son: controladores, rutas y middleware si esta lo requiere.

7.4.2.3.1 Controladores

Los controladores es la parte esencial de la API, ya que son los que manejan las peticiones hechas por el usuario con el fin de enviar respuestas al frontend; En la siguiente figura se puede apreciar uno de los controladores del proyecto.

Figura 35. Controlador de Usuarios

```
const { Acceso } = require("../models/acceso");      alejandroG13, 2 months ago • macageback
const { Archivo } = require("../models/archivo");
const { Comentario } = require("../models/comentario");
const { Imagen } = require("../models/imagenes");
const { Rol } = require("../models/rol");
const { Usuario } = require("../models/Usuario");
const { Up } = require("../models/usuario_proyecto");
const { encrypt } = require("../helpers/handleBcrypt");
const { createAcceso } = require("./acceso-controller");
const email_institucional = require("../helpers/emailPattern");

const getUsuarios = async (req, res = response) => {
  try {
    const usuarios = await Usuario.findAll({
      include: [
        {
          model: Rol,
          attributes: ["nombre_rol"],
        },
      ],
    });
    return res.json({ usuarios });
  } catch (error) {
    return res.status(500).json({ message: error.message });
  }
};

const getProfesores = async (req, res = response) => {
  try {
    const profesores = await Usuario.findAll({
      where: {
        cod_rol: 3,
      },
      include: [
        {
          model: Rol,
          attributes: ["nombre_rol"],
```

Fuente: Autores

En estos controladores vemos que las funciones flechas en su gran mayoría de veces tienen los mismos parámetros “req” y “res”, estos representan request y response que en

español significa “petición” y “respuesta”. En la Figura 35 podemos ver el controlador “usuario-controller”, del mismo modo, el método “getUsuarios” que se encarga cómo su nombre lo indica traer todos los usuarios registrados en la base de datos.

7.4.2.3.2 Middleware

Es la parte del software que brinda servicios y funciones comunes a las aplicación, de tal modo que se usó para hacer chequeos y validaciones con el fin de no sobrecargar la lógica expuesta en los controladores, a continuación se verá el claro ejemplo de uno de los middlewares del backend[Figura 36].

Figura 36. Middleware de autenticación

```
const { verifyToken } = require("../helpers/generateJwt");

const checkAuth = async (req, res, next) => {
  try {
    //TODO: authorization: Bearer 1010101010101001010100
    const token = req.headers.authorization.split(" ").pop(); //TODO:123123213
    const tokenData = await verifyToken(token);
    if (tokenData && tokenData._id) {
      next();
    } else {
      res.status(409);
      res.send({ error: "Token invalido" });
    }
  } catch (error) {
    console.log(error);
    res.status(409);
    res.send({ error: error.message });
  }
};

module.exports = { checkAuth };
```

Fuente: Autores

En este middleware se verifica que el usuario logueado en la plataforma posee un token de autenticación(JWT) para poder hacer peticiones al Backend, con esto se genera un capa de seguridad a la aplicación. Cabe aclarar que este middleware se usa en acciones que pueden afectar en los datos que existen en el sistema, como lo es borrar un registro de cualquier tipo.

7.4.2.3.3 Rutas

Las rutas se conocen como endpoints que permiten la comunicación entre el frontend y el backend. Estas se usan por medio de *peticiones HTTP* como *GET, POST, PUT/PATCH, y DELETE*.

Figura 37. Ruta de Acceso

```
const { Router } = require("express");      alejandroG13, 2 months ago • macageba
const {
  createAcceso,
  updateAcceso,
  deleteAcceso,
  getAccesoById,
} = require("../controllers/acceso-controller");
const { checkAuth } = require("../middlewares/auth");
const { checkRoleAuth } = require("../middlewares/authRole");

const router = Router();

router.post("/", createAcceso);
router.put("/:cod_usuario", updateAcceso);
router.delete("/:cod_usuario", checkAuth, checkRoleAuth([1]), deleteAcceso);
router.get("/:documento_usuario", getAccesoById);

module.exports = router;
```

Fuente: Autores

En la Figura 37 se observa la construcción de de las rutas, se componen de una sintaxis que es "`router.delete("/:cod_usuario", checkAuth, checkRoleAuth([1]), deleteAcceso);`", donde *delete* es el método de petición HTTP, "/:cod_usuario" es la dirección que tomará en una url establecida en el index.js de la aplicación; en el caso de que la petición requiere un parámetro se debe añadir a la ruta el nombre del parámetro que se necesite, "checkAuth" es el middleware de la ruta y al final tenemos un método del controlador que es "deleteAcceso". Para hacer uso de las rutas es necesario exportar el módulo "router".

Para hacer uso de las APIs es necesario declarar las rutas en el index.js, las cuales unen todo el desarrollo de controladores, middlewares y modelos de la aplicación, con el fin de ser consumidas.

Tabla 6. Fase 4. Desarrollo de la aplicación

Tipo	Elementos
Entradas	• Anexo 1. Documento de Gestión de Requerimientos. • Diagramas de Arquitectura 4+1.

	• Modelo entidad-relación de la base de datos
Instrumentos y/o herramientas	• Computador. • Conexión a internet. • Navegador Web. • Visual Studio Code. • PowerDesigner. • pgAdmin 4. • Postman
Salidas	• Construcción de Frontend. • Construcción de Backend. • Integración del Backend en el Frontend.
Tiempo	16 semanas

Fuente: Autores

7.5. Fase 5. Pruebas

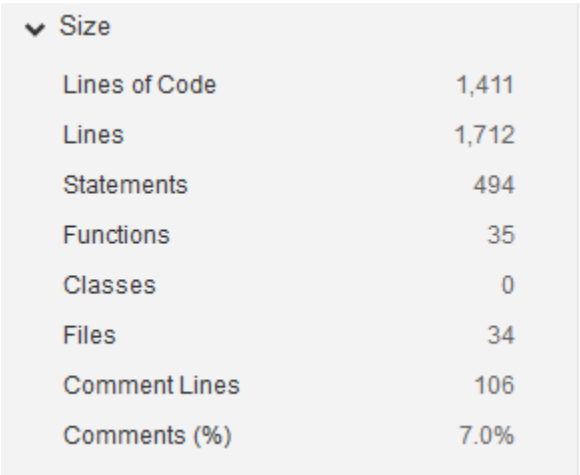
Para el cumplimiento de las pruebas se usó la herramienta de SonarQube , la cual es una herramienta de revisión de código automática lo que ayuda sistemáticamente a entregar Clean Code[32]. A continuación se presentan los resultados al escanear el backend y el frontend evaluando en los siguientes aspectos: *Tamaño del sistema y código fuente, mantenibilidad, Code Smells, Confiabilidad* y *Seguridad*.

7.5.1. Backend

- **Tamaño del sistema y código fuente**

  La aplicación SonarQube calcula líneas de código, funciones, clases, archivos, líneas comentadas, declaraciones, entre otros [Figura 38], indicando el tamaño del software.

Figura 38. SonarQube tamaño Backend



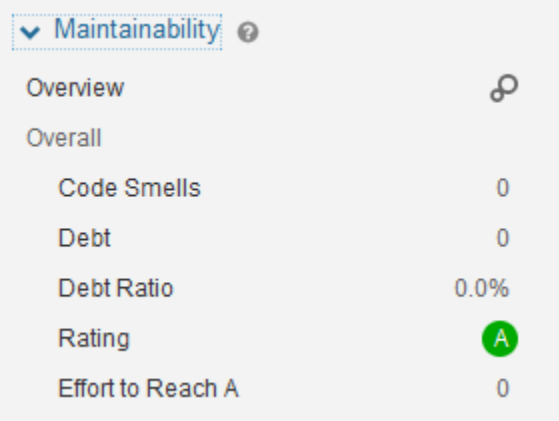
Fuente: Autores

El backend tiene una densidad total de comentarios de 7.0% debido a que solo las funcionalidades de los controladores se documentaron mientras las otras partes del proyecto no ya que tenían código legible.

- **Mantenibilidad**

  Se obtuvo una calificación A indicando que no es demasiado complejo aplicar procesos correctivos.

Figura 39. SonarQube mantenibilidad Backend

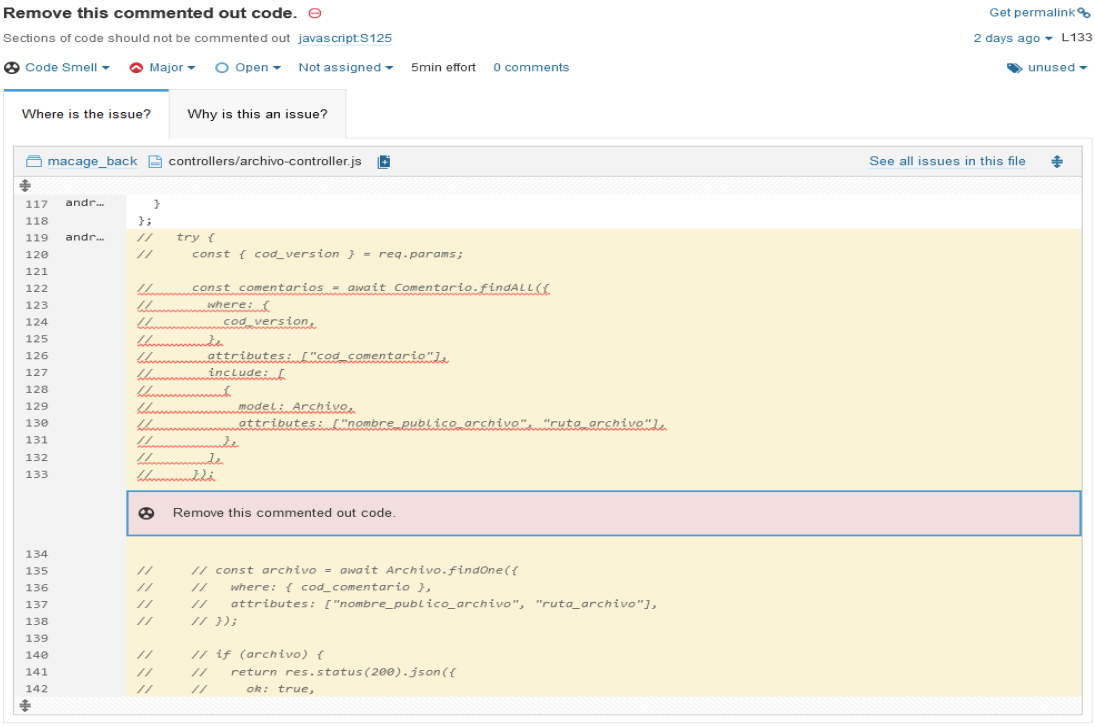


Fuente: Autores

- **Code Smells**

  La cantidad de code smells(código apestoso) encontrados fueron 13, entre ellos comentarios, corrección de sintaxis y definición de constantes. En la siguiente figura se muestra parte del código donde hay un code smell debido a creación de comentarios a código innecesario.

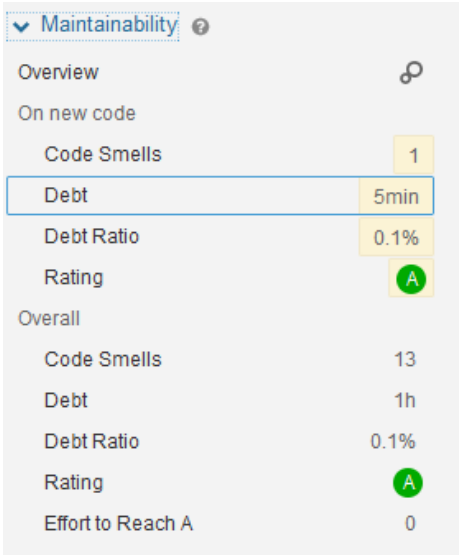
Figura 40. SonarQube Code Smell Backend



Fuente: Autores

Luego de realizar las correcciones pertinentes SonarQube arrojó nuevos resultados en cuanto a los code smells.

Figura 41. SonarQube Code Smell Backend luego de hacer cambios

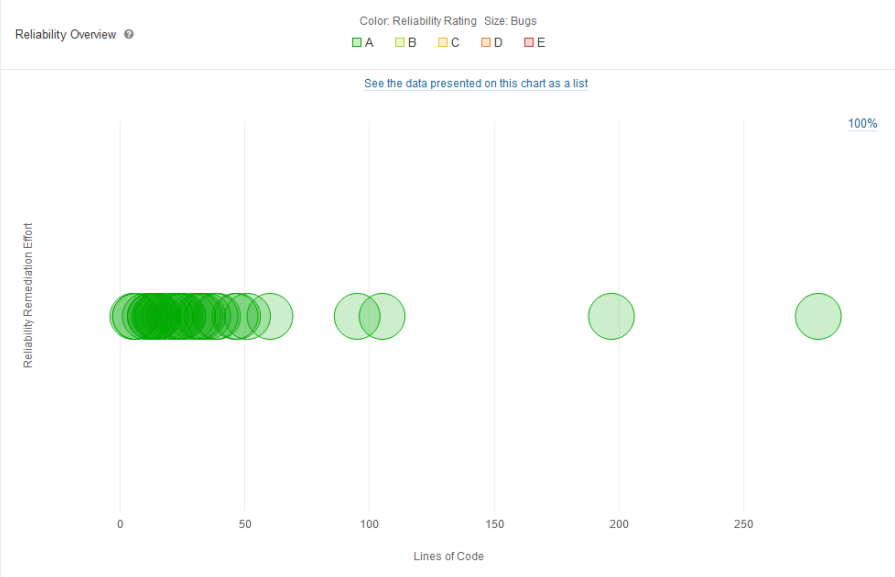


Fuente: Autores

- **Confiabilidad**

  Se obtuvo una calificación A indicando que no se encuentra ningún bug en el software por lo que no es necesario aplicar procesos correctivos.

Figura 42. SonarQube Fiabilidad Backend

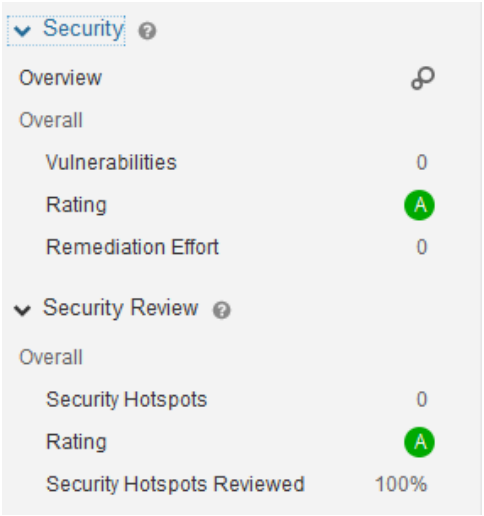


Fuente: Autores

- **Seguridad**

  Se obtuvo una calificación A indicando que no se encuentra ninguna vulnerabilidad en el software por lo que no es necesario aplicar procesos correctivos.

Figura 43. SonarQube Seguridad del Backend



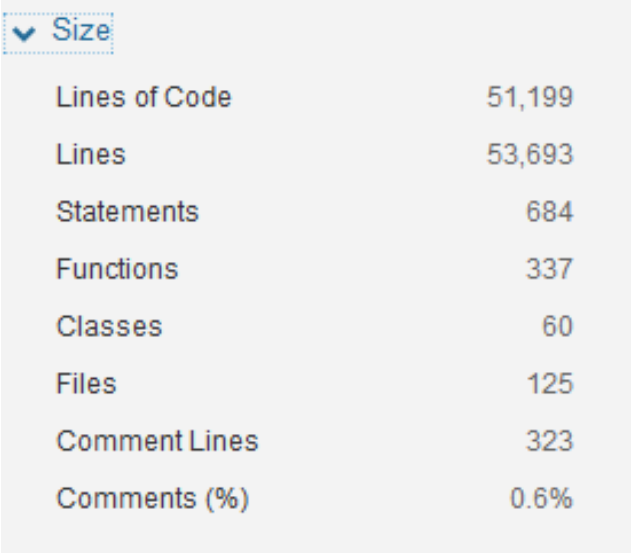
Fuente: Autores

7.5.2. Frontend

- **Tamaño del sistema y código fuente**

  El Frontend contiene más de 50.000 líneas de código, esto se debe al uso de la plantilla de PrimeNg que contiene diferentes estilos para sus temas; la densidad de comentarios es baja debido a que se usan los componentes de PrimeNg.

Figura 44. SonarQube tamaño Frontend

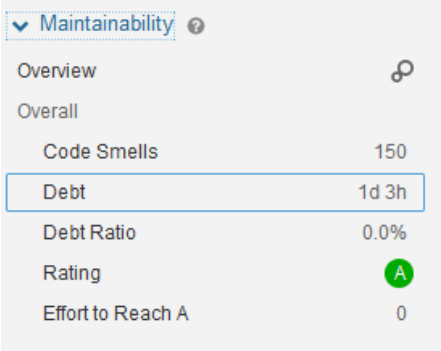


Fuente: Autores

- **Mantenibilidad**

  Se obtuvo una calificación A indicando que no es demasiado complejo aplicar procesos correctivos.

Figura 45. SonarQube mantenibilidad Frontend



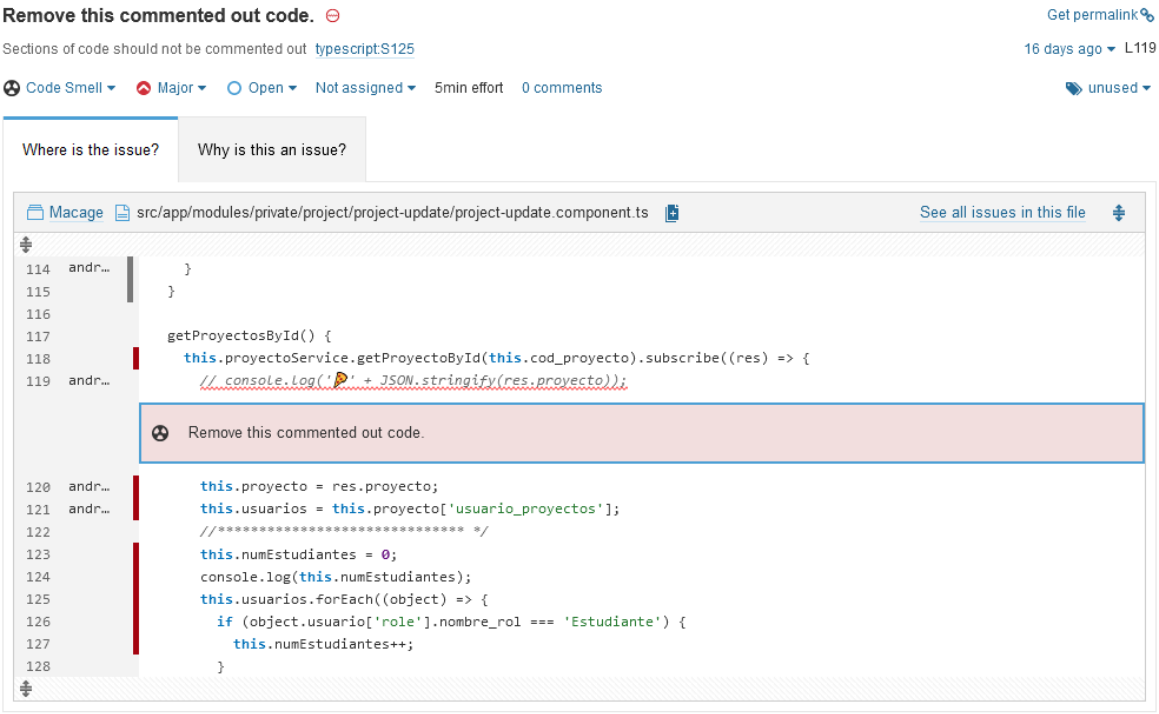
Fuente: Autores

Al ser tan grande el proyecto se tiene que hacer un trabajo intensivo en la limpieza del código para la corrección de code smells. Según SonarQube esta tarea duraría aproximadamente un día y tres horas.

- **Code Smells**

  La cantidad de code smells encontrados fueron 150, entre ellos uso de patrones para los datos, código comentado, bloques de código repetidos, importaciones que no se usan y definición de variables mal escritas. En la siguiente figura se muestra parte del código donde hay un code smell debido a creación de comentarios a código innecesario.

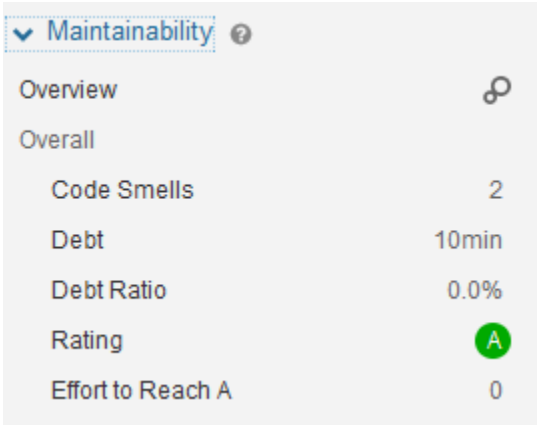
Figura 46. SonarQube Code Smell Frontend



Fuente: Autores

Luego de realizar las correcciones pertinentes SonarQube arrojó nuevos resultados en cuanto a los code smells.

Figura 47. SonarQube Code Smell Frontend luego de hacer cambios

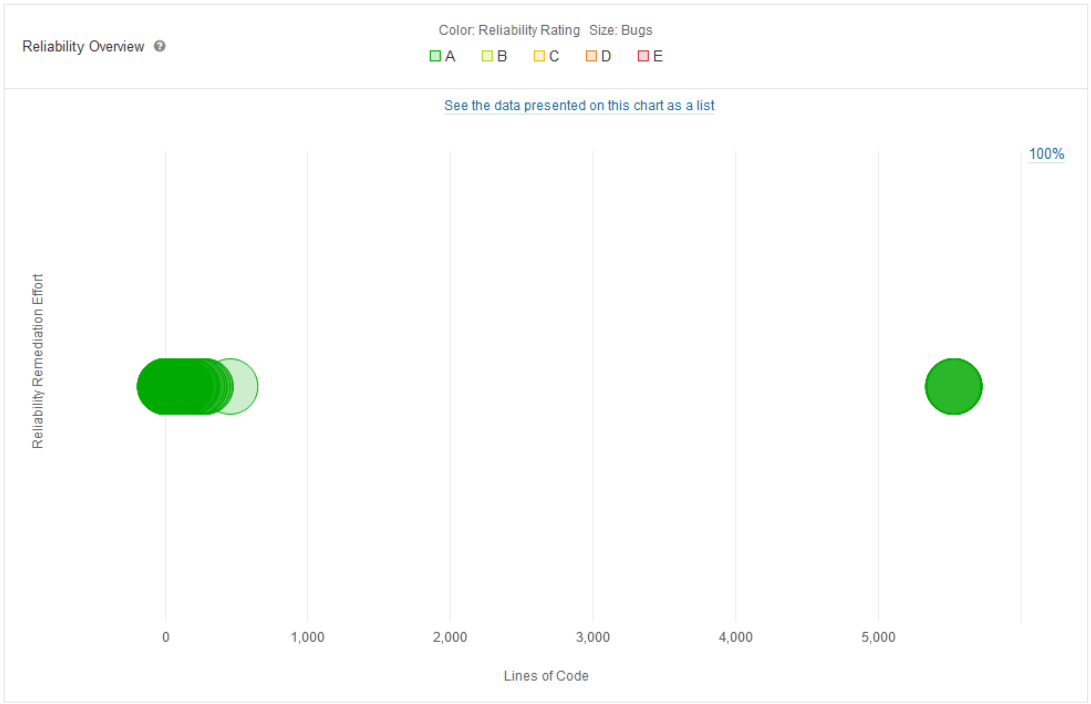


Fuente: Autores

- **Confiabilidad**

  Se obtuvo una calificación A indicando que no se encuentra ningún bug en el software por lo que no es necesario aplicar procesos correctivos.

Figura 48. SonarQube mantenibilidad Frontend

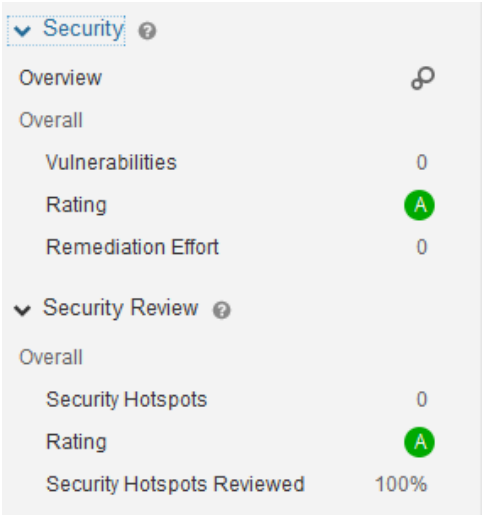


Fuente: Autores

- **Seguridad**

  Se obtuvo una calificación A indicando que no se encuentra ninguna vulnerabilidad en el software por lo que no es necesario aplicar procesos correctivos.

Figura 49. SonarQube Seguridad del Frontend



Fuente: Autores

A continuación, se muestran las entradas, instrumentos/herramientas, entregables y tiempo de la fase de pruebas y validación.

Tabla 7. Fase 5. Pruebas

Tipo	Elementos
Entradas	• Aplicación del Módulo de administración de trabajos de grado Maestría en Calidad y Gestión Integral
Instrumentos y/o herramientas	• Computador. • Conexión a internet. • Navegador Web. • Visual Studio Code. • SonarQube
Salidas	• Resultados análisis SonarQube
Tiempo	1 semana

Fuente: Autores

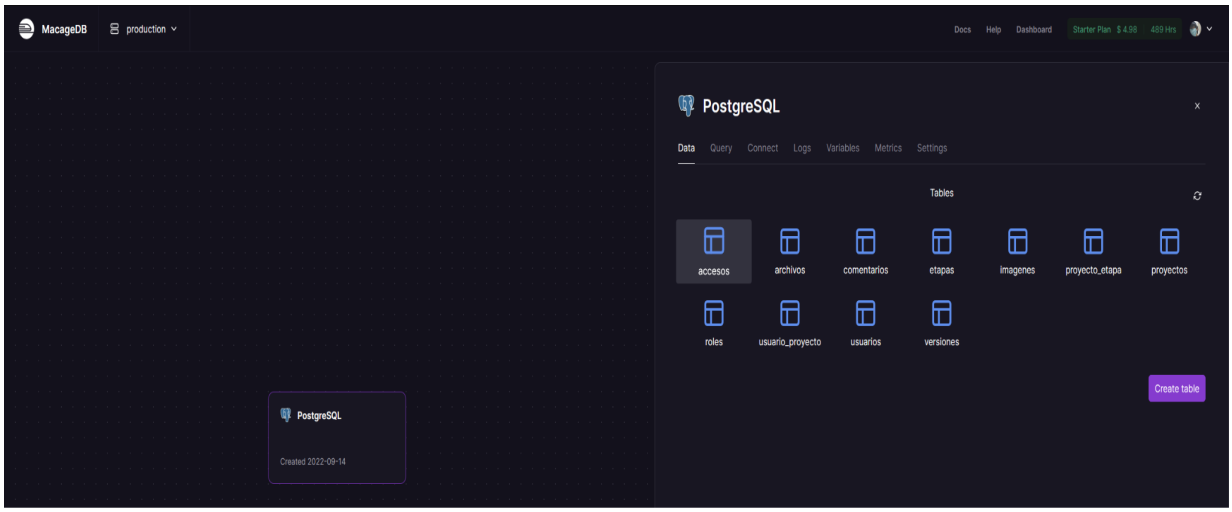
7.6. Fase 6. Puesta en producción

Para la fase de producción se dividió en 3 partes fundamentales que fueron el hosteo de la base de datos, el backend y el frontend.

7.6.1. Base de datos

Para la implementación de la base de datos en la nube se hizo uso de Railway la plataforma de infraestructura donde se puede alojar infraestructura, desarrollar con esa infraestructura localmente y luego implementarla en la nube[34]. En la plataforma de Railway simplemente se hizo uso del Anexo 3, se usó de los scripts de la base de datos sacados de las anteriores fases del proyecto anteriormente definidas en este documento. A continuación se observa en la figura 50 como es la interfaz de Railway y la importancia que tiene para la implementación.

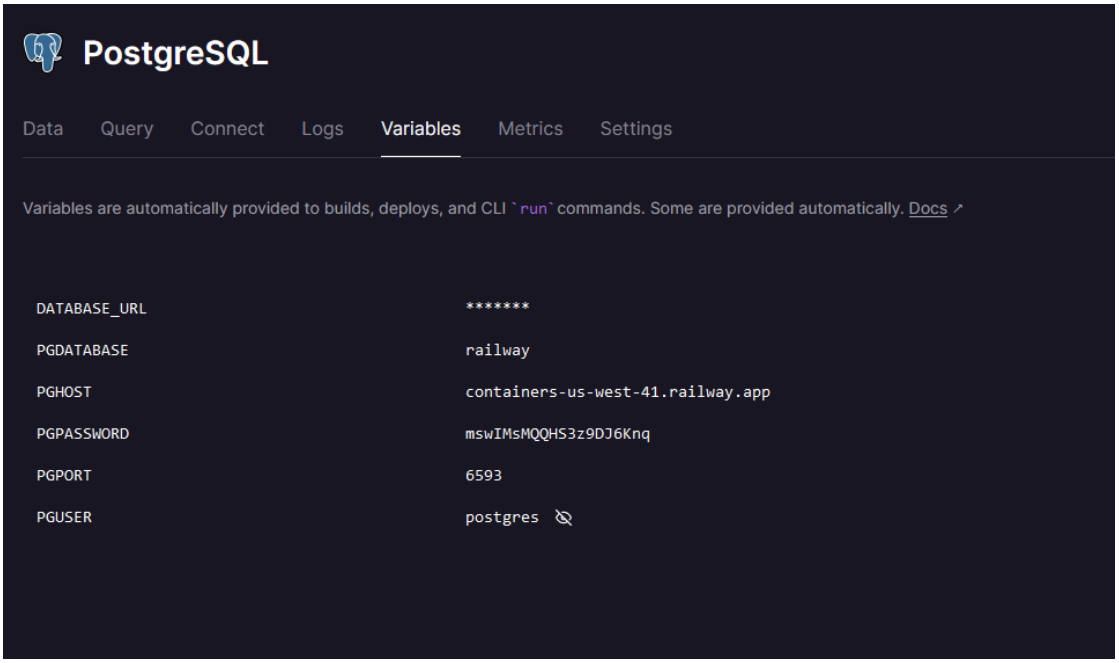
Figura 50. Interfaz Base de Datos



Fuente: Autores

Luego de crear la base de datos en Railway pudimos extraer las variables de configuración acorde a lo necesario para que funcionase el backend [Figura 51].

Figura 51. Interfaz Base de Datos



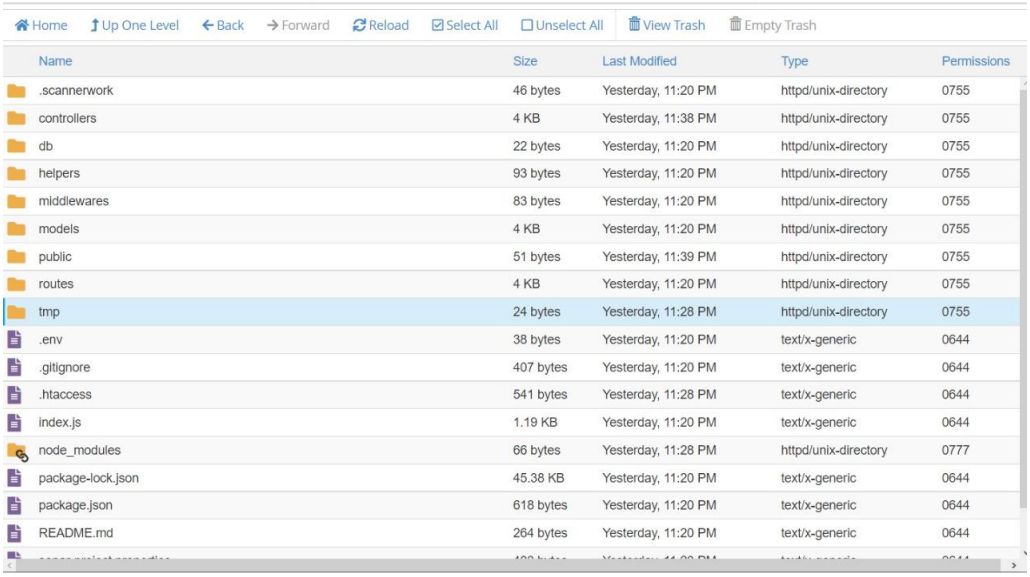
Fuente: Autores

Al extraer la información se hace la conexión el backend tal como se muestra en la Figura 32 del documento que es el archivo “config.js”, el cual se encarga de conectar el backend con la base de datos.

7.6.2. Backend

Para el backend se hizo uso de en Cpanel una herramienta de Dongee hosting donde básicamente se subía el proyecto compilado del backend. El despliegue dentro de Dongee fue sencillo, ya que dentro del host por defecto tiene las dependencias de node.js. Se subió el proyecto a una carpeta dentro de la raíz del host como se muestra a continuación.

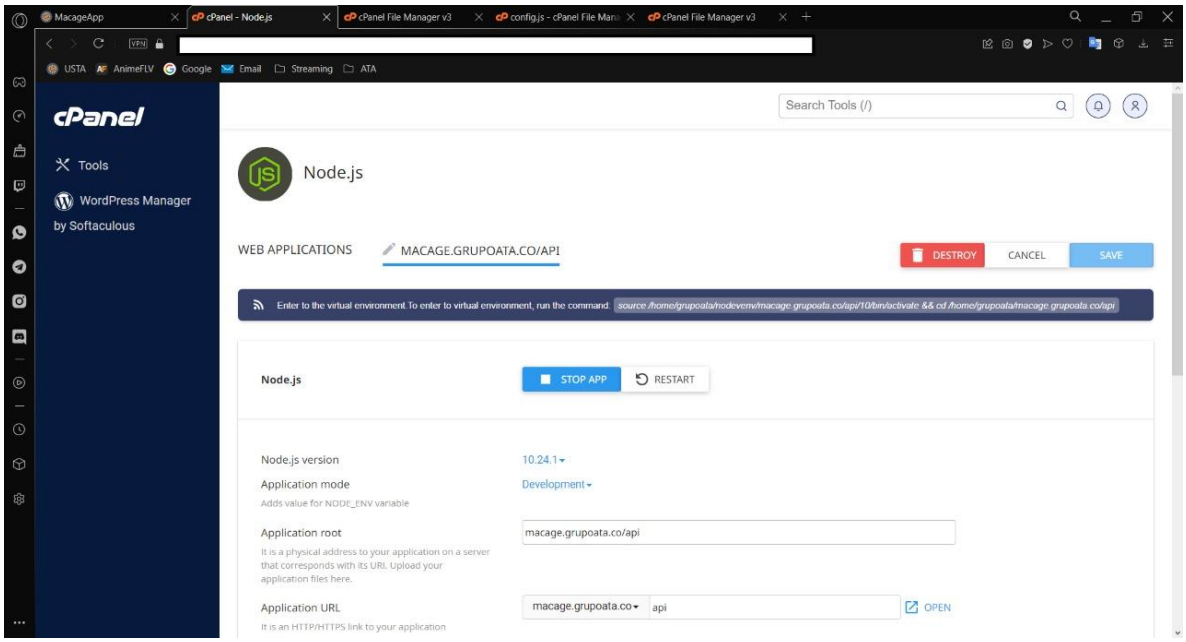
Figura 52. Explorador de archivos de Dongee



Fuente: Autores

Seguidamente hacemos creación de la aplicación en Cpanel de Node.js donde apuntamos a la ruta de la carpeta que contiene los archivos descomprimidos de nuestro backend, para que así el servidor reconozca el backend y nos permita realizar las instalación de paquetes y hacer desplegar la aplicación.

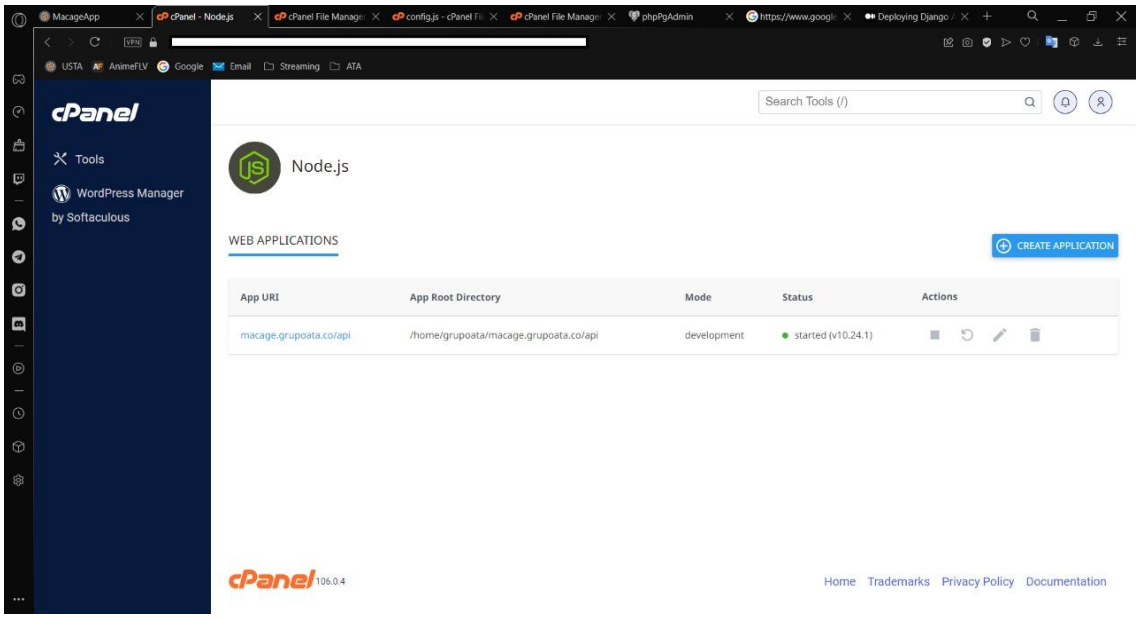
Figura 53. Creación de aplicación Node.js Dongee



Fuente: Autores

Comprobamos dentro del mismo panel su lanzamiento y su estado como se observa en la Figura 54.

Figura 54. Panel de aplicaciones Node.js



Fuente: Autores

7.6.3. Frontend

En cuanto al despliegue del frontend se hizo uso de Firebase [20] por medio de sus productos ofrecidos, que en este caso era el de hosting. Necesitas Firebase CLI (una herramienta de línea de comandos) para alojar tu sitio con Firebase Hosting.

Figura 55. Instalación FireBase Cli

```
$ npm install -g firebase-tools
```

Fuente: Autores

Luego de instalar la CLI de Firebase en la raíz del proyecto se hizo uso de *"firebase init"* comando usado para inicializar los archivos necesarios para el despliegue de la aplicación. Una vez se crearon las variables necesarias para la implementación de firebase se hizo la construcción del proyecto por medio del comando que se muestra en la siguiente figura.

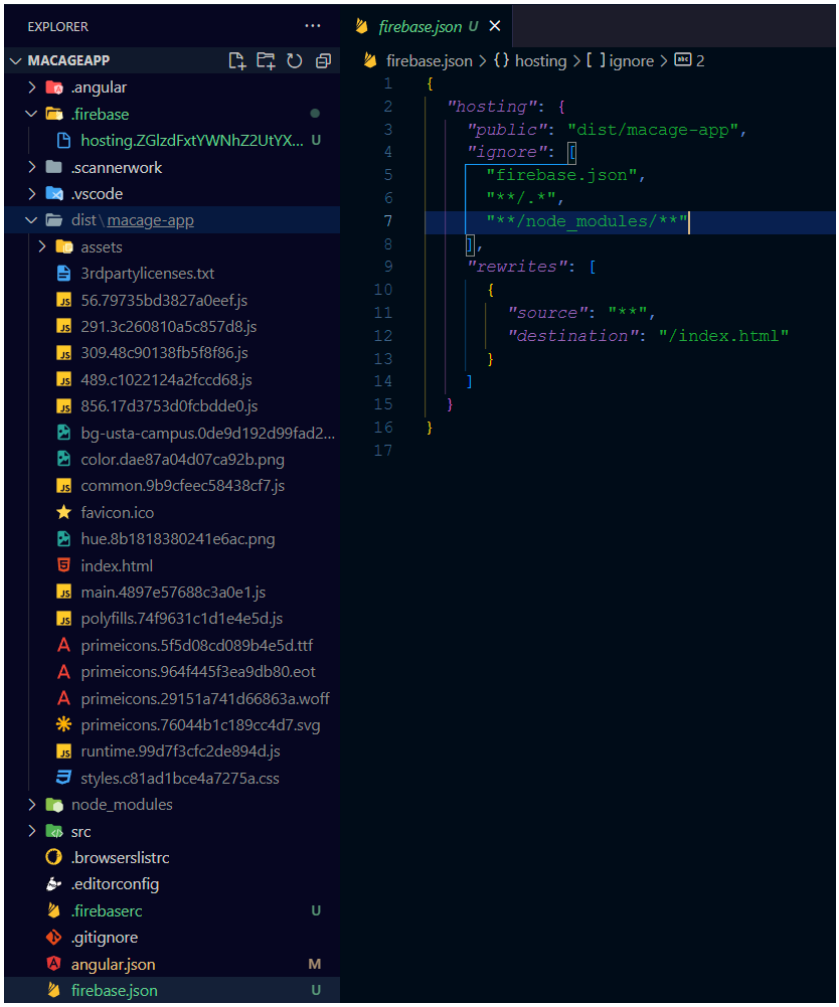
Figura 56. Comando de npm run build

```
PS C:\Users\juand\Documents\Proyecto de grado\Frontend\macageApp> npm run build

> macage-app@0.0.0 build
> ng build

✓ Browser application bundle generation complete.
✓ Copying assets complete.
✓ Index html generation complete.
```

Fuente: Autores

Por lo que la siguiente acción fue modificar el archivo de "*firebase.json*" donde se especifica que se quiere subir al host del firebase en el proyecto que se creó previamente a la instalación de su herramienta de línea de comandos.

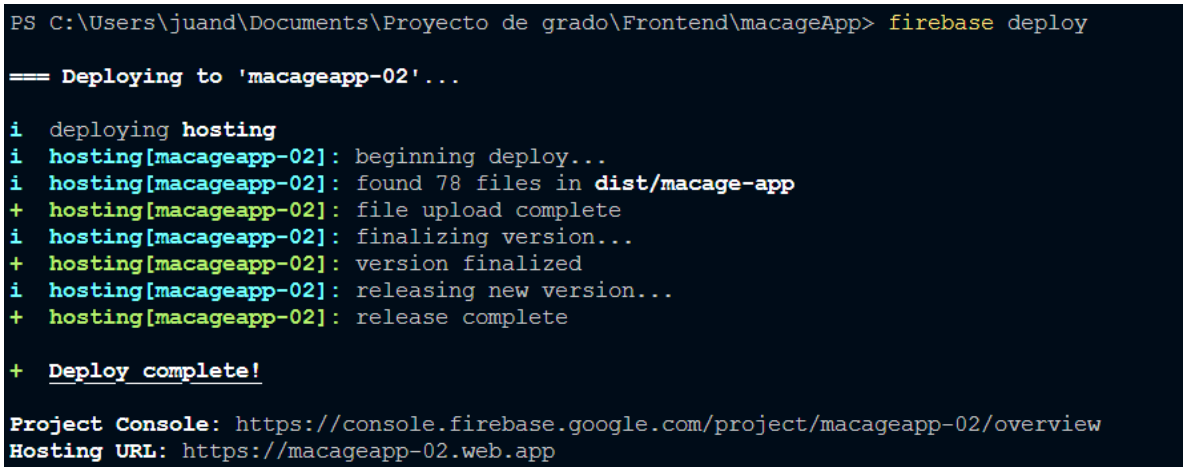
Figura 57. firebase.json y macage-app



Fuente: Autores

Al establecer la carpeta a subir al servidor solo faltaba hacer uso del comando de despliegue con el resultado que se esperaba.

Figura 58. firebase deploy



Fuente: Autores

Luego de finalizar el despliegue de esta aplicación vemos la URL de alojamiento de la aplicación, la cual corresponde a https://macageapp-02.web.app.

Tabla 8. Fase 6. Puesta en producción

Tipo	Elementos
Entradas	• Zips de frontend y backend compilados
Instrumentos y/o herramientas	• Computador. • Conexión a internet. • Navegador Web. • Visual Studio Code. • FireBase • Railway • Dongee hosting
Salidas	• MacageApp
Tiempo	1 semana

Fuente: Autores

8. CONCLUSIONES Y ANÁLISIS DE RESULTADOS

- Se desarrolló con éxito la plataforma para la gestión y control de proyectos de la maestría en Calidad y Gestión Integral con uso de herramientas de código abierto como lo fue Angular, Node.js, Express y PostgreSQL. Lo que ayudó al desarrollar el proyecto fue el uso de la metodología de desarrollo en prototipos lo que permitió mantener mejoría constante en el desarrollo de la arquitectura del proyecto que se enfocó en construir microservicios y consumirlos.
- Se especificó un documento con los requerimientos funcionales y no funcionales del proyecto necesarios para la construcción, haciendo uso de las técnicas de Amador Duran[30] para definir el alcance que tendría el software.
- Se implementó el uso del modelo de arquitectura de vistas 4+1 propuesto por P. Kruchten por medio de diagramas, necesarios para la definición de viabilidad del proyecto y sus fases de desarrollo.
- Tras el desarrollo y análisis de los módulos creados para el correcto funcionamiento de la plataforma web, se evidencia que cada uno de estos cumple a cabalidad su esperado funcionamiento.
- Se aseguró la funcionalidad de la plataforma con el escaneo al código fuente del proyecto proporcionado por SonarQube, el cual demostró la calidad del software. Con los resultados fue más sencillo hacer optimización de código, cambiar o agregar nuevas funcionalidades para que software continúe funcionando como se espera y en caso de que haya errores permite detectarlos rápidamente.

9. REFERENCIAS

[1] AppMaster Team. (2022, July 18). *Los mejores frameworks de JavaScript para usar en 2022 - Una guía completa*. AppMaster. Retrieved September 1, 2022, from https://appmaster.io/es/blog/los-mejores-frameworks-de-javascript-para-usar-en-2022-una-guia-completa

[2] Auth0. (2021). *JSON Web Tokens*. JSON Web Tokens - jwt.io. Retrieved September 5, 2022, from https://jwt.io/

[3] Cutiño, P., & EDteam. (2020, July 4). *Framework vs Librería*. Edteam. Retrieved September 5, 2022, from https://ed.team/blog/framework-vs-libreria

[4] Flanagan, D. (2011). *JavaScript: The Definitive Guide* (6th ed.). O'Reilly Media, Incorporated.

[5] Google LLC. (2020). *Angular - Introducción a la Documentación de Angular*. Angular. https://docs.angular.lat/docs

[6] Green, B., & Seshadri, S. (2013). *AngularJS*. O'Reilly Media, Incorporated.

[7] Johnson, R. E., & Foote, B. (1988). *Designing Reusable Classes* (Vol. 22-35). Journal of Object-Oriented Programming. https://www.researchgate.net/publication/215446177_Designing_Reusable_Classes

[8] Jones, M., Bradley, J., & Sakimura, N. (2015, 05 05). *RFC 7519: JSON Web Token (JWT)*. RFC Editor. Retrieved September 5, 2022, from https://datatracker.ietf.org/doc/html/rfc7519

[9] MDN contributors. (2022, August 4). *HTML: Lenguaje de etiquetas de hipertexto | MDN*. MDN Web Docs. https://developer.mozilla.org/es/docs/Web/HTML

[10] MDN contributors. (2022, August 15). *Prototype-based programming - Glosario | MDN*. MDN Web Docs. https://developer.mozilla.org/es/docs/Glossary/Prototype-based_programming

[11] Medium, V. M. (2017, April 25). *History of SPA frameworks: AngularJS 1.x and nostalgia.* Medium. Retrieved September 1, 2022, from https://medium.com/@metnew/history-of-spa-frameworks-angularjs-1-x-and-nostalgia-2e4a00df5ee2

[12] Microsoft. (2012, 10 01). *TypeScript Docs*. TypeScript: JavaScript With Syntax For Types. Retrieved September 1, 2022, from https://www.typescriptlang.org/es/

[13] Moiseev, A., & Fain, Y. (2018). *Angular Development with TypeScript*. Manning.

[14] Npm Inc. (2009, 09 02). *npm Docs*. npm Registry. Retrieved September 1, 2022, from https://www.npmjs.com/

[15] Ollivier, S., & Gury, P.-A. (2016). *AngularJS: Desarrolle hoy las aplicaciones web de mañana*. ENI.

[16] OpenJS. (2009, 04 27). *About*. Node.js. Retrieved September 3, 2022, from https://nodejs.org/en/about/

[17] OpenJS. (2010, 05 22). *Glosario de Express*. Express.js. Retrieved September 3, 2022, from https://expressjs.com/es/resources/glossary.html

[18] PrimeTek. (2009). *PrimeFaces*. PrimeFaces – Ultimate UI Framework. Retrieved September 5, 2022, from https://www.primefaces.org/#

[19] Sass Team. (2020, 06 01). *Documentation*. Sass. Retrieved August 31, 2022, from https://sass-lang.com/documentation/

[20] Google Developers. (2022). *Firebase*. Firebase by Google. Retrieved September 15, 2022, from https://firebase.google.com/docs

[21] Sequelize Contributors. (2022). *Sequelize*. Sequelize | Feature-rich ORM for modern TypeScript & JavaScript. Retrieved September 5, 2022, from https://sequelize.org/

[22] Subramaniam, V. (2018). *Rediscovering JavaScript: Master ES6, ES7, and ES8*. Pragmatic Bookshelf.

[23] Microsoft. (2015). *Visual Studio Code*. Visual Studio Code - Code Editing. Redefined. Retrieved September 6, 2022, from https://code.visualstudio.com/

[24] SAP. (2019). *SAP PowerDesigner | Data Modeling and Enterprise Architecture*. SAP. Retrieved September 6, 2022, from https://www.sap.com/latinamerica/products/technology-platform/powerdesigner-data-modeling-tools.html

[25] The PostgreSQL Global Development Group. (1996). *PostgreSQL*. PostgreSQL: The world's most advanced open source database. Retrieved September 6, 2022, from https://www.postgresql.org/

[26] Figma, Inc. (2016). Figma Website. Figma: the collaborative interface design tool. Retrieved September 6, 2022, from https://www.figma.com/

[27] Postman, Inc. (2022). *Postman Website*. Postman API Platform | Sign Up for Free. Retrieved September 6, 2022, from https://www.postman.com/

[28] Vega, F. (2019). ¿Qué es Git y para qué sirve? Platzi. Retrieved September 6, 2022, from https://platzi.com/clases/1557-git-github/20215-que-es-git/

[29] Hernández Sampieri, R., Fernández Collado, C., & Baptista Lucio, P. (2014). *Metodología de la investigación* (P. Baptista Lucio, Ed.). McGraw-Hill Education.

[30] Durán Toro, A. (2000). Un entorno metodológico de ingeniería de requisitos para sistemas de información. Universidad de Sevilla. Departamento de Lenguajes y Sistemas Informáticos. http://hdl.handle.net/11441/15365.

[31] Kruchten, P. (1995). *The 4+1 View Model of architecture* (Vol. 12). IEEE Software. 10.1109/52.469759

[32] SonarSource S.A. (2008). *SonarQube Documentation*. SonarQube Documentation. Retrieved September 11, 2022, from https://docs.sonarqube.org/latest/

[33] Amazon Web Services. (2022). *What is Service-Oriented Architecture? Service-Oriented Architecture Explained*. AWS. Retrieved September 11, 2022, from https://aws.amazon.com/what-is/service-oriented-architecture/?nc1=h_ls

[34] Railway Corp. (2022). *Railway*. Railway.app. Retrieved September 15, 2022, from https://railway.app/

10. ANEXOS

- Anexo 1. Gestión Requerimientos

Enlace: Anexo 1 GR Fase 1 Gestión Requerimientos

- Anexo 2. Wireframe MacageApp

Enlace: Wireframe Macage.fig

- Anexo 3. Scripts de Base de Datos de MacageApp

Enlace: Database_Version_2_DB

- Anexo 4. Código fuente de MacageApp Frontend

Enlace: Código_fuente_frontend

- Anexo 5. Código fuente de MacageApp Backend

Enlace: Código_fuente_backend

- Anexo 6. MacageApp Deploy

Enlace: https://macageapp-02.web.app

- Anexo 7. Diagramas Fase 2

Enlace: Diagramas del modelo de vistas 4+1