Sistema de gestión para trabajos de grado en posgrado

Juan Esteban Amaya Ramírez

Pontificia Universidad Javeriana Cali
Facultad de Ingeniería y Ciencias
Ingeniería de Sistemas y Computación
Cali
2025

Sistema de gestión para trabajos de grado en posgrado

Juan Esteban Amaya Ramírez

Trabajo de grado

*D*irector: Ing. Juan Carlos Martínez Arias

Pontificia Universidad Javeriana Cali
Facultad de Ingeniería y Ciencias
Ingeniería de Sistemas y Computación
Cali
2025

Santiago de Cali, Febrero 20 de 2025

Señores
Pontificia Universidad Javeriana Cali.
Dr. Gerardo M. Sarria
Director Carrera de Ingeniería de Sistemas y Computación.
Cali.

Cordial Saludo

Por medio de la presente me permito informarle que he revisado el proyecto de grado: **"Sistema de gestión para trabajos de grado en posgrado"** del estudiante: Juan Esteban Amaya Ramírez (código: 8949439) del cual soy Director, lo considero finalizado y listo para sustentación.

Atentamente,

Juan Carlos Martínez Arias
Firmado digitalmente por Juan Carlos Martínez Arias
Fecha: 2025.02.20 13:25:01 -05'00'

Juan Carlos Martínez Arias
Director de trabajo de grado
Pontificia Universidad Javeriana Cali

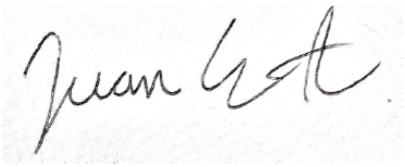
Santiago de Cali, Febrero 20 de 2025

Señores
Pontificia Universidad Javeriana Cali.
Dr. Gerardo M. Sarria
Director Carrera de Ingeniería de Sistemas y Computación.
Cali.

Cordial Saludo

Me permito presentar a su consideración el proyecto de grado denominado: **"Sistema de gestión para trabajos de grado en posgrado"**, con el fin de cumplir con los requisitos exigidos por la Universidad para llevar a cabo el término de este y posteriormente optar por el título de Ingeniero de Sistemas y Computación.

Atentamente,



Juan Esteban Amaya Ramírez
Código: 8949439

Índice

Índice de figuras

Resumen

La Universidad Javeriana Cali actualmente gestiona los trabajos de grado de manera manual, lo que implica una considerable carga de trabajo para el personal académico y aumenta el riesgo de errores y problemas en el proceso de evaluación. La comunicación, citación, registro de actas y verificaciones se realizan a través de correos electrónicos, generando repetición y posibles confusiones. La solución actual, basada en el uso de Excel y Teams, se muestra insuficiente y otras alternativas comerciales presentan limitaciones. Como respuesta a estos desafíos, se desarrolló un sistema de gestión integral que permitiera a los estudiantes tener un seguimiento claro del estado de sus trabajos, ingresar, registrar, editar, eliminar y mantener sus propios proyectos, así como facilitar la toma de decisiones. Para el personal de apoyo académico, el sistema ofrecería la capacidad de automatizar la comunicación, ver el proceso y tener toda la información en un mismo lugar, reduciendo los riesgos asociados con la gestión manual de los trabajos de grado en el posgrado de la Facultad de Ingeniería y Ciencias de la Universidad Javeriana Cali.

Palabras Clave: Gestión de proyectos, Trabajos de grado, Posgrado, Facultad de ingeniería y ciencias.

Summary

The Universidad Javeriana Cali currently manages degree projects manually, which implies a considerable workload for the academic staff and increases the risk of errors and problems in the evaluation process. Communication, citation, recording of minutes and verifications are done through e-mails, generating repetition and possible confusion. The current solution, based on the use of Excel and Teams, is insufficient and other commercial alternatives have limitations. In response to these challenges, a comprehensive management system was developed to allow students to clearly track the status of their work, enter, register, edit, delete and maintain their own projects, as well as facilitate decision making. For the academic support staff, the system would offer the ability to automate communication, view the process and have all the information in one place, reducing the risks associated with the manual management of graduate work in the graduate program of the Faculty of Engineering and Sciences of the Universidad Javeriana Cali.

Keywords: Project management, Degree projects, Graduate program, Faculty of Engineering and Sciences.

Este trabajo de grado va dedicado a Giovanny Amaya y a Liz Yuri Ramírez, dos padres maravillosos quienes hicieron posible la culminación de mis estudios universitarios de la mejor manera.

También quiero nombrar y agradecer a cada una de las personas que me ayudaron y acompañaron de forma académica y personal en todo este proceso:

- Nicolle Naranjo Astaiza
- Juan José Marín Ochoa
- Guido Ernesto Salazar Muñoz
- Dilan Andres Correa
- Nicole Molineros
- Santiago Torres Rincón
- María José Suarez
- Cristian Riascos
- Luis Alfredo Rodríguez
- Fabian Antoyne García

Para finalizar quiero agradecer a la persona más importante en mi vida y en todo este proceso Laura Catalina Lozano Tabares, por ser mi razón para seguir, mi apoyo incondicional para terminar este trabajo y mejorar mi persona cada día.

1. Introducción

Acceder a un nivel de posgrado en la Facultad de Ingeniería y Ciencias como estudiante implica no solo un nuevo nivel de exigencia académica, sino también una inmersión más profunda en la especialización de la disciplina y un compromiso más firme con la investigación avanzada. Los trabajos de grado en el posgrado representan una fase crucial en este proceso educativo, ofreciendo a los estudiantes la oportunidad única de abordar problemas más complejos y contribuir significativamente a la expansión del conocimiento en áreas específicas de su carrera.

En este contexto, la gestión efectiva de los trabajos de grado en posgrado se rige como un pilar central para asegurar el éxito en la formación académica. La planificación cuidadosa, la comunicación fluida y la evaluación meticulosa de estos trabajos son elementos clave que no solo garantizan un seguimiento claro y conciso del progreso académico, sino que también son fundamentales para la creación de un entorno propicio para la investigación avanzada. No obstante, esta tarea se ve obstaculizada si no se cuenta con una organización eficiente de los datos, si existen posibles problemas de comunicación entre estudiantes y tutores, o si se experimentan problemas en la evaluación de los trabajos de grado.

En respuesta a estas necesidades específicas, se desarrolló un sistema integral diseñado para apoyar la gestión de trabajos de grado en el posgrado, concentrándose de manera especializada en las dinámicas particulares de la Facultad de Ingeniería y Ciencias de la Universidad Javeriana Cali.

2. Descripción del Problema

2.1. Planteamiento del Problema

Actualmente, el proceso para gestionar trabajos de grado se hace de manera manual en la Universidad Javeriana Cali; todas las comunicaciones, citaciones, actas y comprobaciones se realizan por correo, con la misma estructura, y se envían uno por uno, lo que puede llegar a ser tedioso para las asistentes, ya que deben comunicarse con los estudiantes, directores de programa y posibles jurados muchas veces durante el proceso de evaluación de los proyectos.

La repetición de las comunicaciones y el error humano pueden causar problemas y fallos en la gestión de los trabajos de grado, lo que puede concluir en demoras en la entrega de los resultados, confusiones en las citaciones, etc. Además, datos importantes como la información de los estudiantes pueden estar en diferentes herramientas a la vez, como Excel, Word o Teams, lo que puede causar problemas en el seguimiento de la gestión del trabajo de grado.

Hoy por hoy, la solución que está usando la Universidad para este problema es utilizar Excel y Teams para almacenar los datos y la información de los trabajos de grado. En una investigación rápida se encontró que la mayoría de las instituciones universitarias utilizan softwares de terceros que deben pagar de forma mensual.

El problema con dichos softwares es que incluyen funcionalidades o herramientas que no están enfocadas específicamente al ámbito académico. En caso de que la Universidad tenga una necesidad adicional, es necesario contactar a la empresa desarrolladora, esperando que ofrezcan una solución, lo que podría resultar en usar un software incompleto, o incluso volver a la forma manual de realizar los procesos, que es precisamente lo que se busca evitar.

Por todo lo anterior, el proyecto desarrollado es un prototipo funcional de un sistema de gestión para apoyar el manejo de los trabajos de grado en el área de posgrado para la facultad de Ingeniería y Ciencias. En dicho sistema, los estudiantes pueden ingresar, editar y eliminar sus propios proyectos; también cuentan con un sistema de estados para saber en qué momento del proceso se encuentran. Para el personal de apoyo académico, el sistema ofrece la capacidad de automatizar el envío de correos, revisar los proyectos y acceder a toda la información desde un mismo lugar.

2.1.1. Formulación

¿Cómo desarrollar un sistema que apoye la gestión y el manejo de los trabajos de grado para posgrado?

2.1.2. Sistematización

¿Cuáles son los requisitos del prototipo funcional?

¿Cuál es el diseño más apropiado a partir de los requisitos encontrados?

¿Cómo desarrollar el prototipo funcional con base en el diseño?

¿Cómo evaluar el prototipo funcional?

2.2. Objetivos

2.2.1. Objetivo General

Desarrollar un prototipo de software para apoyar la gestión y el manejo de los trabajos de posgrado en la Universidad Javeriana Cali.

2.2.2. Objetivos Específicos

Identificar y determinar cuáles son los requisitos que necesitan ser tenidos en cuenta.

Diseñar el prototipo de software a partir de los requisitos.

Desarrollar el prototipo de software funcional con base en el diseño.

Evaluar el prototipo de software funcional desarrollado.

3. Marco de Referencia

3.1. Marco Teórico

Los sistemas de software complejos son esenciales para que las organizaciones gestionen eficientemente sus recursos. Este sistema de apoyo se diseñó con el propósito de almacenar, organizar, controlar y, en cierta medida, automatizar las tareas de una empresa para que realizar todo el proceso sea más fácil. En el contexto de este proyecto, se busca unificar en un único software todas las operaciones relacionadas con la gestión de un trabajo de posgrado. Las metas principales son facilitar el almacenamiento de los datos, mejorar la comunicación, agilizar el seguimiento y facilitar todo el proceso que conllevan los trabajos de posgrado en la Facultad de Ingeniería y Ciencias. Es importante señalar que la Facultad alberga nueve programas de posgrado, de los cuales las cinco maestrías y el doctorado requieren trabajo de grado. Dada esta diversidad, resulta imperativo definir algunos conceptos clave, entre ellos la ingeniería de software, el ciclo de vida del software y las herramientas a usar.

3.1.1. La ingeniería de software

"Es una disciplina de la ingeniería que se ocupa de todos los aspectos de la producción de software desde las primeras etapas, partiendo de la especificación del sistema, hasta el mantenimiento del mismo cuando ya ha sido entregado. En esta definición, hay dos frases clave. La primera es 'Disciplina de ingeniería', la cual indica que los ingenieros aplican de manera selectiva teorías, métodos y herramientas de manera que logren realizar sistemas funcionales. Además, siempre intentan descubrir soluciones a los problemas, incluso cuando no existen teorías o métodos aplicables. En segundo lugar, los ingenieros reconocen que deben trabajar con recursos organizativos y financieros determinados; a la hora de buscar soluciones deben tener en cuenta dichas limitaciones." [1].

3.1.2. El ciclo de vida del software

El ciclo de vida del software se compone de:

- **Identificación y determinación de requisitos:** Se identifican las necesidades del cliente a través de reuniones con él, y en el equipo de trabajo se determinan los requisitos del sistema. Esta parte es la más importante, ya que hacer bien este trabajo determina el éxito del proyecto y evita errores futuros. Por ende, se deben entender las expectativas y las necesidades del cliente de la mejor manera posible.

- **Diseño del sistema:** Con los requisitos claros y analizados, se procede a diseñar el sistema. En este punto se definen los componentes principales, se planifica cómo se van a desarrollar y se define la arquitectura del software.
- **Desarrollo del sistema (codificación):** Se escribe el código fuente basado en el diseño, desarrollando un prototipo funcional con ayuda de toda la información recopilada anteriormente.
- **Evaluación del sistema:** Una vez finalizado el desarrollo, se lleva a cabo una evaluación de todo el sistema, lo que incluye pruebas unitarias, de integración y de sistema para asegurarse que el prototipo cumple con lo requisitos iniciales y funciona de manera esperada.

Estos pasos se pueden repetir a lo largo del ciclo de vida del software.

3.1.3. Patrón de arquitectura MVC

"El patrón de arquitectura Modelo - Vista - Controlador fue acuñado por primera vez en la década de 1970 por Trygve Reenskaug. En la arquitectura MVC, el sistema se divide en tres componentes independientes entre sí. Los datos de la aplicación son gestionados por el modelo, que es responsable del almacenamiento y recuperación de datos. La tarea de la vista es presentar el modelo visualmente al usuario y obtener respuestas. El controlador es la parte central que actúa entre el modelo y la vista. Interpreta las peticiones del usuario y notifica a la vista y al modelo para que realicen los cambios oportunos" [2].

3.1.4. Node.js y Express

Node.js es un entorno de ejecución JavaScript multiplataforma y de código abierto. Es una herramienta popular para casi cualquier tipo de proyecto. Funciona ejecutando el motor JavaScript V8, el núcleo de Google Chrome, fuera del navegador [3]. Además, combina tecnologías de desarrollo front-end y back-end, ejecutando las aplicaciones dentro de su propio tiempo de ejecución en el servidor, lo que permite manejar grandes volúmenes de datos. Express es un framework minimalista para Node.js que facilita la creación de aplicaciones web, proporcionando un sistema flexible para gestionar rutas, middleware y peticiones HTTP.

3.1.5. MySQL

MySQL es un sistema de gestión de bases de datos de código abierto desarrollado por Oracle. Este sistema relacional permite almacenar, organizar y recuperar datos de manera eficiente. Funciona utilizando un lenguaje de consulta estructurado (SQL) para interactuar con la información.

3.1.6. Sequelize

Sequelize es un ORM (Object-Relational Mapping) para Node.js que facilita la interacción con bases de datos relacionales, permitiendo trabajar con ellas mediante objetos en lugar de consultas SQL directas. Gracias a su enfoque basado en modelos, la creación, consulta y eliminación de datos se vuelve más intuitiva [5].

3.1.7. Trabajos Relacionados en el Ámbito Estudiantil

- **Análisis, diseño, e implementación de un software, para la administración de los proyectos de grado en el programa de ingeniería de sistemas, aplicando una metodología ágil** [6].

  En este proyecto se desarrolló una herramienta para hacer consultas de tesis de manera rápida, evitando la redundancia en los temas de trabajos de grado futuros y permitiendo el mejoramiento o actualización de los trabajos ya publicados. Además permite hacer seguimiento a proyectos en curso, asignar proyectos a directores de tesis y difundir de actas. Se diferencia con el proyecto propuesto en que no solo se podrán ver los trabajos de grado en curso, sino que también se podrán gestionar, así como ver en qué estado se encuentra el trabajo, los posibles jurados, etc.

- **Software para el seguimiento, la gestión y el control de proyectos de grado en el departamento de electrónica (SSGPG)** [7]

  Este proyecto es una herramienta web para la gestión de trabajos de grado en el Departamento de Ingeniería Electrónica. También permite realizar seguimiento y control a cada proyecto, lo que ayuda a simplificar los tiempos en la entrega de documentos, asignación de jurados, elaboración de notificaciones, entre otras. Se diferencia con el proyecto propuesto en que este último se realizará de forma personalizada para los trabajos de grado en posgrado de la Facultad de Ingeniería y Ciencias de la Universidad Javeriana Cali.

- **Sistema de información Web que asiste el proceso de radicación y seguimiento de proyectos de grado de la Especialización en Ingeniería de Software de la Universidad Distrital Francisco José de Caldas** [8]

  Este proyecto aborda todas las etapas del ciclo de vida básico del software para el desarrollo de un sistema de información web que apoya el proceso de radicación de proyecto de grado de los estudiantes. Asimismo, respalda el proceso de asignación de directores, revisores, el proceso de seguimiento y ofrece herramientas funcionales que apoyen la gestión y búsqueda de trabajos ya existentes. El proyecto propuesto se diferencia en que se realizará específicamente para la Facultad de Ingeniería y Ciencias de la Universidad Javeriana Cali.

3.1.8. Trabajos Relacionados en el Ámbito Empresarial

- **Jira software de Atlassian [9]**

  Atlassian es una empresa de software con sede en Australia que crea productos para empresas y desarrolladores de software. Entre estas se encuentra Jira software, una herramienta de gestión de proyectos ágiles que ofrece una gran cantidad de funciones para planificar y plantillas que podrían ser utilizadas para apoyar con la gestión de trabajos de grado. Además cuentan con recursos para desarrolladores y se puede pagar de forma mensual. Se diferencia con el proyecto propuesto en que este último se trata de un sistema de gestión específico para trabajos de grado, por lo que tendrá las funciones necesarias y pertinentes para realizar con éxito dicha gestión.

- **Basecamp [10]**

  Es una herramienta de gestión de proyectos y colaboración en equipo que ha ganado popularidad en Estados Unidos debido a que es eficaz y cuenta con un diseño simple. En cada proyecto se puede crear una lista de tareas, lo que facilita la comunicación entre personas del proyecto; también proporciona funciones para la gestión de tareas, documentos compartidos, calendarios y cuenta con un precio fijo mensual por mes. El problema de esta herramienta, así como la diferencia principal con el proyecto propuesto, es que Basecamp carece de funciones avanzadas, por lo que para proyectos complejos que requieren una planificación detallada y seguimiento específico como lo es un trabajo de grado, no es una opción útil.

4. Requisitos del sistema de gestión

En este capítulo se realizó un diagrama de contexto para comprender el alcance y la interacción del sistema de gestión con su entorno. Además, se elaboraron diagramas de actividades para representar los pasos y estados que conforman el proceso dentro del sistema. También se diseñó un diagrama de casos de uso, el cual ofrece una visión general de las funciones que desempeñará cada actor dentro del sistema. Finalmente, toda esta información se consolidó en los requisitos funcionales, no funcionales y limitaciones, proporcionando una base clara para el desarrollo del prototipo funcional.

4.1. Diagrama de Contexto

El primer paso en todos los desarrollos de software es entender el contexto en el que el sistema se va a desenvolver; de este análisis podremos conocer las personas o entidades que van a hacer uso del sistema, así como las necesidades que necesita cubrir. Con esto en mente, se llevó a cabo una reunión con el director de posgrados de la Universidad Javeriana Cali, a partir de la cual se elaboró el siguiente diagrama de contexto en el que se concluyó quienes son los stakeholders, las entradas y salidas de información, y los límites del sistema de gestión. Esta visión general que proporciona el diagrama ayuda a dar una primera idea de los requisitos que requiere el sistema.

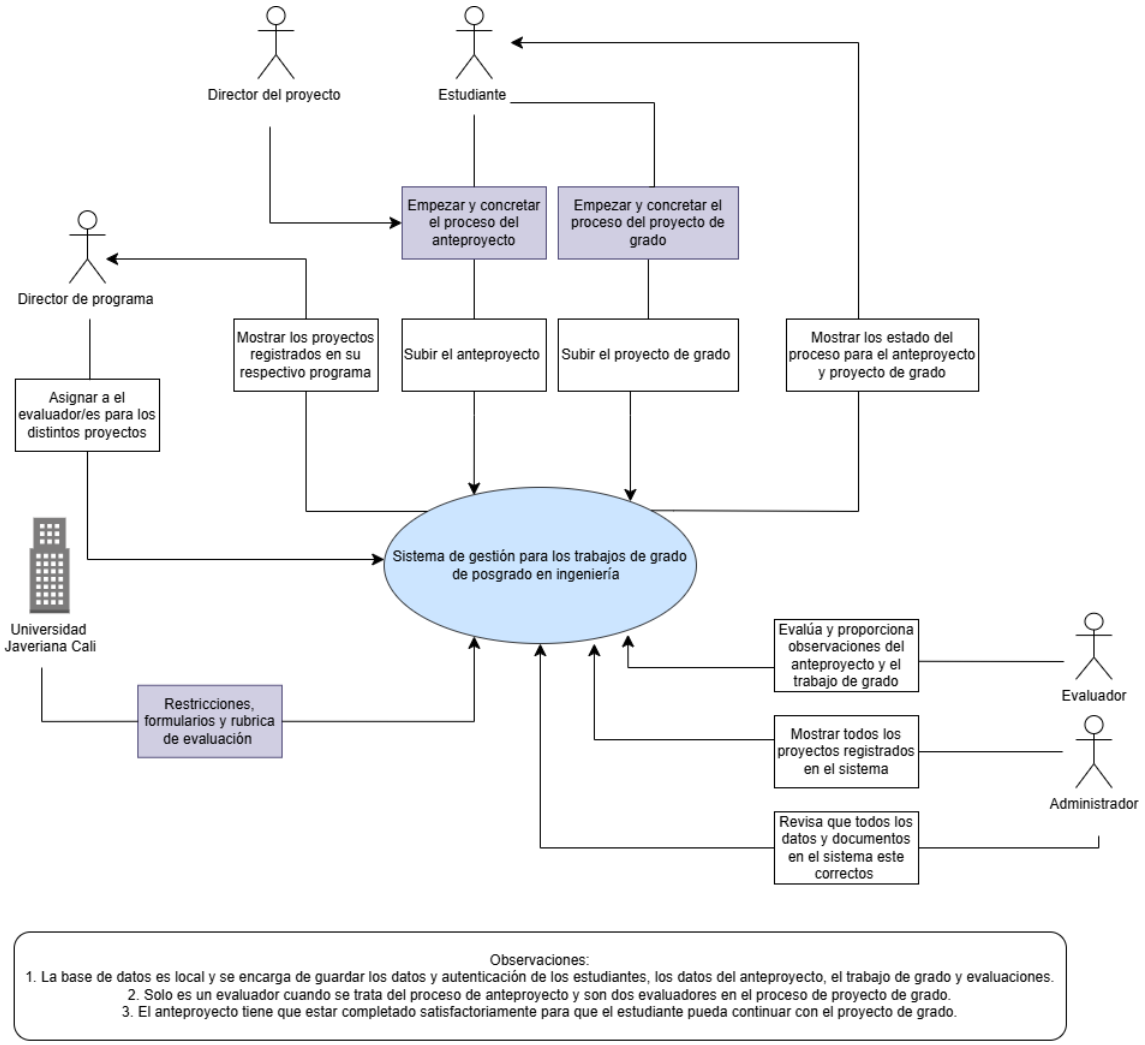


Figura 1: Diagrama de Contexto

Como se puede ver en el diagrama de la **Figura 1**, el sistema involucra a seis stakeholders, los cuales se componen de cinco actores principales y una entidad externa, cuyos roles se describen a continuación:

- **Estudiante:** Es una persona que hace parte de alguna maestría o doctorado en la Facultad de Ingeniería y debe entregar su proyecto de grado. Inicialmente, el estudiante debe comenzar y concretar el proceso del anteproyecto, un paso obligatorio para todos los alumnos. Una vez completado, el estudiante sube su anteproyecto al sistema. Tras finalizar satisfactoriamente este proceso, el estudiante lo repetirá con su proyecto de grado.
- **Administrador:** Corresponde al rol del personal de apoyo académico. En el sistema tiene la función de revisar cada documento de forma general para que cumpla con todas las restricciones que impone la Universidad Javeriana Cali.
- **Director del proyecto:** Es un profesor que cumple con la función de ser el guía y el acompañamiento del estudiante en todo el proceso del anteproyecto

y proyecto de grado, asegurándose de que el estudiante siga la metodología adecuada, cumpla con los plazos; también revisa los avances en ambos proyectos. Para que el sistema de gestión tenga el correcto funcionamiento en el todo proceso, el director del proyecto debe estar definido desde el inicio del anteproyecto.

- **Director de programa:** Es la persona encargada de supervisar y coordinar alguno de los programas de posgrados de la Facultad de Ingeniería y Ciencias. Por esto el sistema debe mostrar todos los trabajos de grado iniciados por los estudiantes que pertenezcan al programa. En el sistema, el director de programa tiene la función de asignar uno o dos posibles evaluadores, dependiendo del caso, para cada anteproyecto o proyecto de grado.
- **Evaluador:** Es un profesor o un colaborador con la función de realizar las evaluaciones y observaciones de todos los anteproyectos y proyecto de grado a los que ha sido asignado.
- **Universidad Javeriana Cali:** La Universidad es el ente regulador que establece los campos requeridos para el registro y define la rúbrica de evaluación.

4.2. Diagrama de actividades

Después de entender el contexto del sistema, se realizó una reunión con la Asistente de la Maestría de Ingeniería para comprender a fondo la gestión involucrada en los anteproyecto y proyectos de grado de los estudiantes de posgrado. Con esto se buscaba analizar cómo el sistema podría integrarse en todo el proceso. Durante la reunión, se identificaron dos grandes procesos clave: el primero es el registro de los estudiantes, ya que necesita una confirmación adicional que cambia el flujo del sistema; el segundo abarca todo el proceso relacionado con el trabajo de grado, que comienza con el anteproyecto. A partir de lo aprendido en esta reunión, se desarrollaron diagramas de actividades para estos dos procesos.

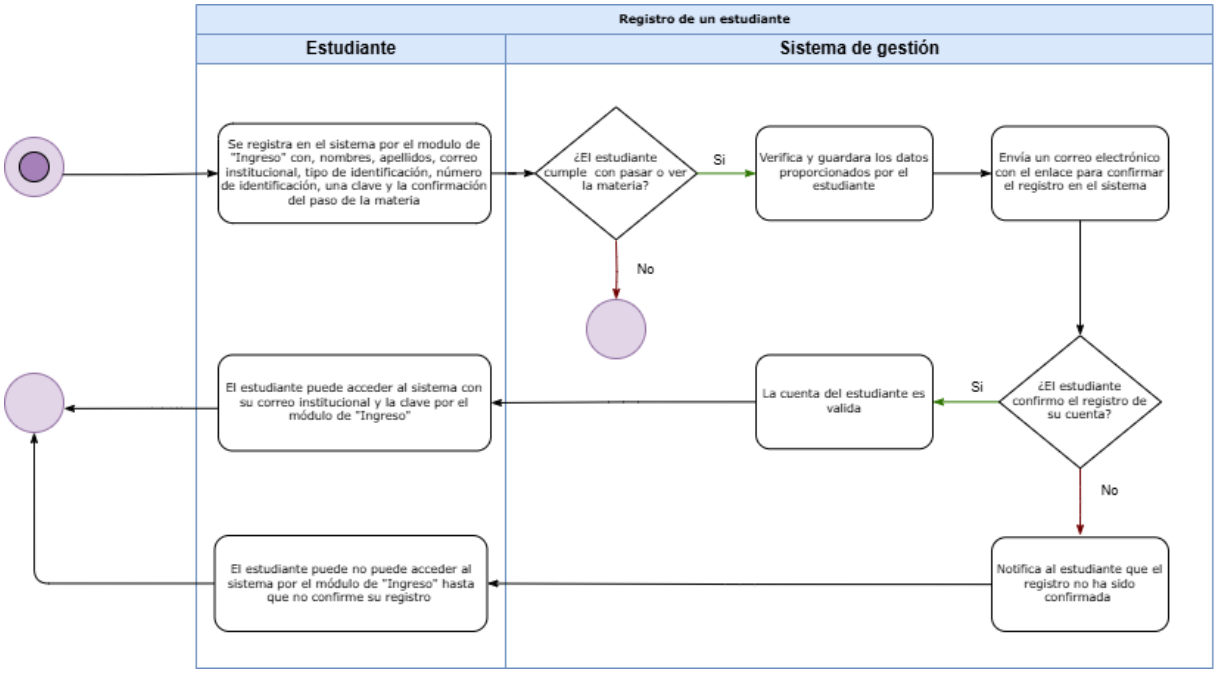


Figura 2: Diagrama de Actividades - Registrar un estudiante

Al momento en que un usuario se registra, el sistema se encarga de verificar y guardar el correcto almacenamiento de los datos para luego enviar una forma de validación del registro que el usuario debe confirmar. La **Figura 2** muestra como ejemplo el registro de un estudiante, ya que es el único caso en el que hay un paso extra que puede cambiar el flujo del proceso de registro. En este proceso, al momento de completar el formulario de registro, el sistema le preguntará si ya cursó, está cursando o aún no ha visto la materia “Anteproyecto para posgrados”. Si el estudiante indica que ha aprobado o está cursando la materia, el sistema enviará un correo de confirmación para que valide su registro en el sistema. En caso de que el estudiante no haya cursado la materia, no se le permitirá completar el registro en el sistema de gestión, ya que es un requisito obligatorio para iniciar este proceso.

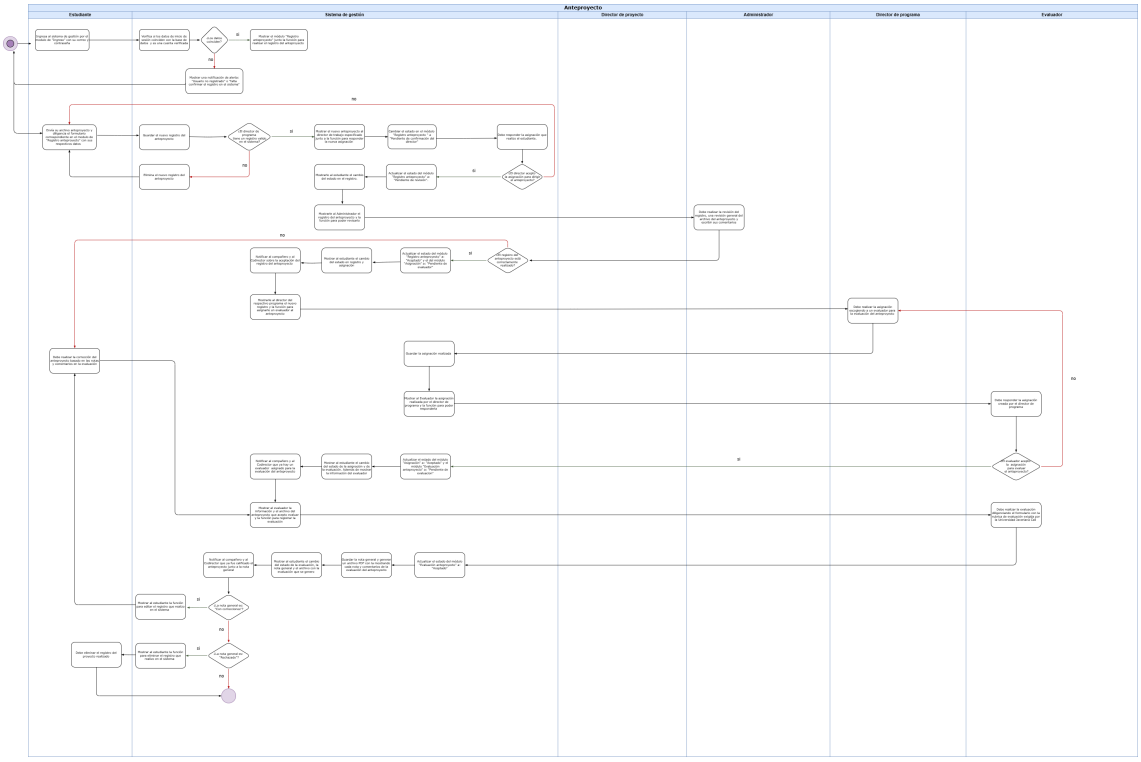


Figura 3: Diagrama de Actividades - Proceso completo del anteproyecto

Como se puede ver en el diagrama de la **Figura 3**, el sistema de gestión se encarga de almacenar y visualizar todo el proceso en el anteproyecto, que concierne a la primera parte del trabajo de grado. Para esto, brinda herramientas de edición, revisión, eliminación y asignación de manera que se pueda realizar todo el proceso de manera exitosa. Se realizó el esquema únicamente para el anteproyecto porque que este proceso y el proceso del proyecto de grado son muy parecidos, la única diferencia notable es el número de evaluadores. La gestión de las citas para las sustentaciones o reuniones que se soliciten en los trabajos de grado, ya sean virtuales o presenciales, es una acción que corresponde a la Asistente de la Maestría de Ingeniería, y es ajena al sistema de gestión.

4.3. Diagrama de casos de uso

El diagrama anterior permite ver de manera detallada todos los componentes de los módulos que se usarán para el anteproyecto y el proyecto de grado. Sin embargo, se necesita una visión a alto nivel de los comportamientos y usos de gestión por los stakeholders del sistema, lo cual se logra realizando un diagrama de casos de uso.

En el diagrama de casos de uso de la **Figura 4** se identificaron cuatro módulos que corresponden a los cuatro usos principales del sistema, cada uno con sus funciones bien definidas. El uso del término “módulos” para representar las partes más importantes del sistema permite una mejor organización y gestión del mismo,

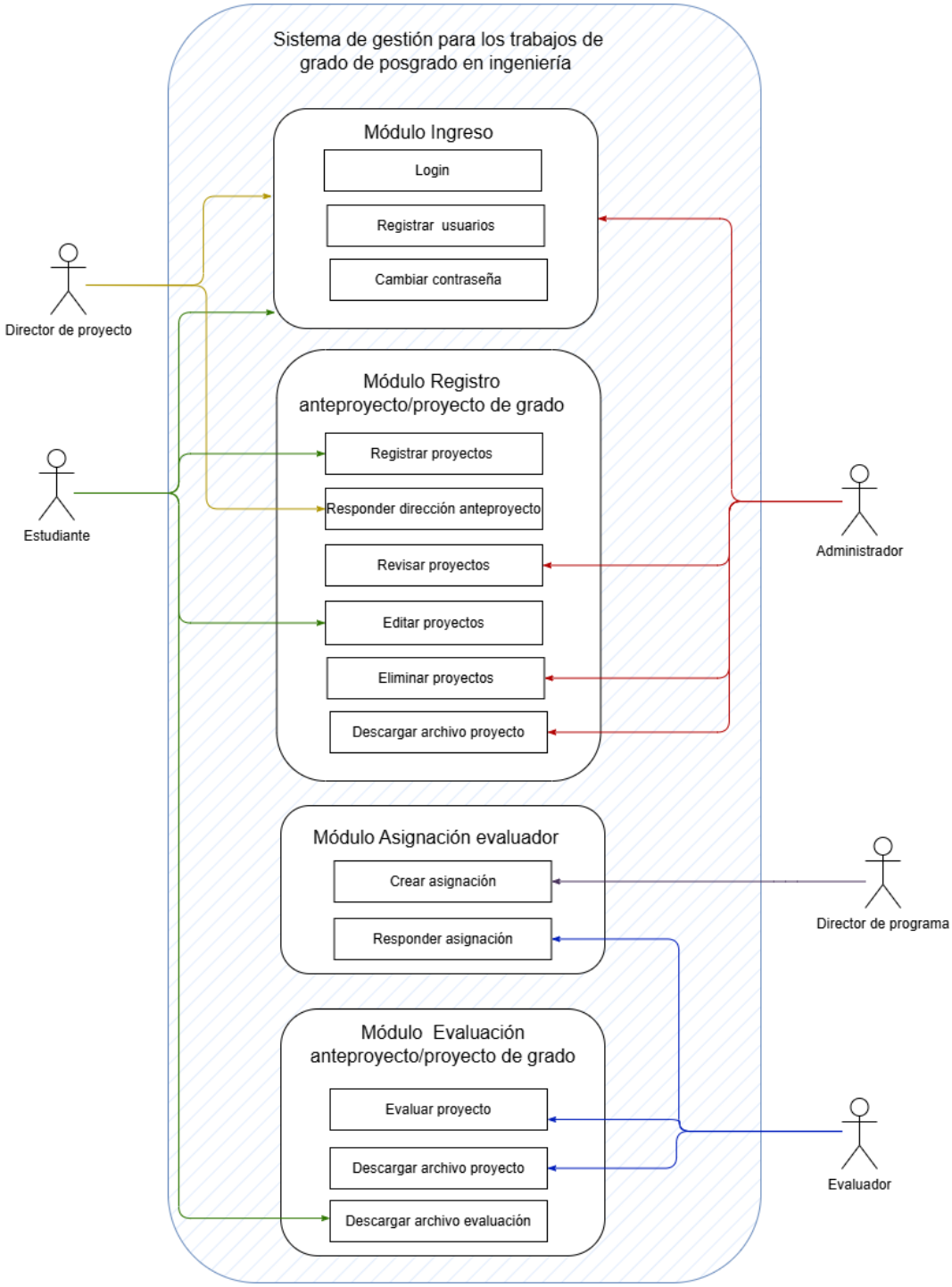


Figura 4: Diagrama de casos de uso

facilitando la división de responsabilidades. También se identificaron 5 roles, que se diferencian por sus comportamientos y usos del sistema de gestión.

El propósito de los diagramas es ayudar a entender cómo se hace toda la gestión para los trabajos de grado y de esta forma definir cómo se acoplaría el sistema en el proceso. Con el funcionamiento del sistema definido se pueden establecer los requi-

sitos, que serán la base para el desarrollo del prototipo del sistema. A continuación, se van a especificar los requisitos funcionales, no funcionales y las limitaciones que van a definir el correcto funcionamiento del prototipo del sistema de gestión.

4.4. Requisitos funcionales

Director de proyecto

- **RF-001:** El sistema debe mostrar a los directores de proyecto todos los anteproyectos y proyectos de grado registrados bajo su dirección de manera secuencial. Además, estos usuarios deben tener acceso a la información más importante de cada proyecto y de cada estudiante.
- **RF-002:** El sistema debe permitir que el director de proyecto acceda a una función para aceptar o rechazar una petición de dirección en caso de tener alguna pendiente. Al registrar un anteproyecto, el director especificado tendrá una solicitud para realizar la dirección de dicho trabajo. Si el director de proyecto no está registrado en el sistema, este debe enviarle un correo electrónico notificándole la necesidad de ingresar y completar su registro para que el estudiante pueda continuar con el proceso.

Administrador

- **RF-003:** El sistema debe mostrar al administrador todos los anteproyectos y proyectos de grado registrados de manera secuencial. Además, este usuario debe tener acceso a la información completa de cada proyecto y de cada estudiante.
- **RF-004:** El sistema permite al administrador acceder a la función "Revisar", dentro del módulo de "Registro anteproyecto/proyecto de grado". Esta función le permite visualizar el formulario con los datos ingresados por el estudiante. El administrador tiene la capacidad de revisar cada petición, descargar el archivo correspondiente y, en caso de encontrar errores, puede cambiar el estado de la solicitud en el sistema. Además, el módulo cuenta con una sección de observaciones donde el administrador puede detallar textualmente los errores que encontró y recomendaciones de cómo solucionarlo.
- **RF-005:** El sistema permite al administrador eliminar el registro de cualquier proyecto, siempre y cuando se encuentre en el módulo de registro anteproyecto/proyecto de grado y pendiente de revisión. De lo contrario no puede realizarlo para no comprometer el funcionamiento del resto del sistema.

Estudiante

- **RF-006:** El sistema permite a los estudiantes registrarse en el sistema por medio de un formulario del módulo de "Ingreso". Los datos que solicita este formulario son el código estudiantil, los nombres, los apellidos, el correo institucional y la contraseña, que deberá ser confirmada. Además, el formulario debe incluir un campo en el que el estudiante confirme si ha cursado o está cursando la materia "Anteproyecto para posgrados". En caso de cumplir con este requisito, puede continuar con el registro de su usuario.

- **RF-007:** El sistema debe mostrar a los estudiantes información general sobre cada uno de los módulos en los que se encuentre su proyecto. Asimismo, el estudiante debe tener acceso en todo momento a sus datos personales, a sus registros y a la información más relevante de los usuarios que participen en cada módulo de su proceso.

- **RF-008:** El sistema permite a cada estudiante registrar un único anteproyecto y un único trabajo de grado en el sistema por medio del módulo de "Registro anteproyecto/proyecto". El módulo se compone de un formulario con los datos necesarios para cada registro y contiene un campo en donde puede subir su respectivo archivo PDF cumpliendo con las especificaciones de formato que tiene la Universidad Javeriana Cali.

- **RF-009:** El sistema incluye una función en el módulo de "Registro anteproyecto/proyecto" que permite a los estudiantes editar y reenviar su archivo cuando el proyecto correspondiente muestre el estado "Con correcciones". Mientras esto suceda, el estudiante puede acceder al formulario, actualizar la información y cargar una nueva versión del archivo. El estudiante debe tener acceso al archivo que se encuentra en el sistema actualmente para que pueda diferenciarlo del que va a enviar.

- **RF-010:** El sistema permite al estudiante acceder a la función para editar un registro cuando el estado del anteproyecto sea "Con correcciones". Cuando el estado sea "Rechazado" el estudiante solo puede eliminar el registro y debe volver a empezar el proceso.

- **RF-011:** El sistema permitirá a los estudiantes descargar un archivo con la evaluación de su proyecto cuando el estado del módulo "Evaluacion anteproyecto/proyecto" sea "Aceptado". Este archivo debe cumplir con la rúbrica de evaluación que tiene la Universidad Javeriana Cali.

Director de programa

- **RF-012:** El sistema debe mostrar a los directores de programa todos los anteproyectos y proyectos de grado registrados en sus respectivos programas de manera secuencial. Además, estos usuarios deben tener acceso a la información más importante de cada proyecto.

- **RF-013:** El sistema debe tener una funcionalidad en el módulo de "Asignación de evaluador" para el director de programa donde puede crear las asignaciones

para cada proyecto en su programa. La asignación se registra mediante un formulario con los datos generales de un evaluador. En caso de que el usuario evaluador no esté registrado, el sistema de gestión debe enviarle la asignación por correo electrónico.

Evaluador/es

- **RF-014:** El sistema debe mostrar a los evaluadores que hayan aceptado sus asignaciones, todos los anteproyectos y proyectos de grado registrados que se encuentren en el módulo de "Evaluación anteproyecto/proyecto de grado". Además, estos usuarios deben tener acceso a la información más importante de cada proyecto y del director que los asignó.
- **RF-015:** El sistema deberá mostrar las asignaciones a los evaluadores registrados. Si un evaluador tiene una asignación pendiente, cuenta con una función para aceptar o rechazar la evaluación de un anteproyecto o proyecto de grado. En el caso de los anteproyectos, solo el primer evaluador que acepte la asignación puede realizar la evaluación. Mientras que, en los proyectos de grado, únicamente los dos primeros evaluadores que acepten serán los responsables de llevarla a cabo.
- **RF-016:** El sistema debe tener una funcionalidad en el módulo de "Evaluación anteproyecto/proyecto de grado" para los evaluadores donde pueden realizar la evaluación para el anteproyecto y trabajos de grado. Un evaluador solo puede evaluar los proyectos de las asignaciones que haya aceptado. Al momento de realizar la evaluación, el evaluador debe descargar el archivo correspondiente.

Sistema de gestión

- **RF-017:** El sistema cuenta con un módulo de "Ingreso" en el que todos los usuarios pueden iniciar sesión mediante su correo y contraseña. Desde este módulo, también pueden acceder al proceso de registro de usuarios, el cual solicita diferentes datos según el tipo de usuario seleccionado. Además, el sistema incluye una función que permite a los usuarios cambiar su clave cuando lo necesiten.
- **RF-018:** El sistema cuenta con un mecanismo de confirmación de registro para garantizar la autenticidad de los usuarios. Una vez que el usuario complete el proceso de registro, el sistema envía un correo electrónico con un enlace de verificación. El usuario debe acceder a dicho enlace para confirmar su cuenta y activar su acceso al sistema. El usuario no puede iniciar sesión hasta que la cuenta esté confirmada.
- **RF-019:** El sistema debe incluir una barra de navegación, la cual permite a los distintos tipos de usuario acceder a sus funciones específicas determinadas y cerrar sesión. Esta barra debe contener el logo de la Universidad Javeriana Cali y la interfaz debe utilizar los colores representativos de la institución para garantizar coherencia visual con la identidad institucional.

- **RF-020:** El sistema debe permitir que únicamente el estudiante que creó el anteproyecto o proyecto de grado pueda acceder a las funciones de edición y eliminación de su registro.
- **RF-021:** El sistema debe ser capaz de diferenciar entre los distintos tipos de usuario y proporcionar acceso a las funciones correspondientes según su rol dentro de la plataforma.
- **RF-022:** El sistema debe mostrar visualmente el progreso de cada proceso mediante una barra de avance o un indicador similar. Además, el estudiante podrá acceder al registro de su proyecto de grado sólo cuando el anteproyecto haya sido completado y aprobado.
- **RF-023:** El sistema debe ser capaz de mostrar el progreso del estudiante en cada uno de los módulos. El progreso en el sistema se muestra a través de estados: "Aceptado", "Pendiente", "Rechazado" o "Con Correcciones".
- **RF-024:** El sistema no debe permitir que los estudiantes hagan uso de módulos posteriores a los que aún no ha completado. Para saber que un módulo está completado, el estado debe ser "Aceptado".
- **RF-025:** El sistema debe contar con un mecanismo de envío de correos electrónicos para notificar al compañero y al codirector sobre los estados de cada módulo, tanto en el anteproyecto como en el proyecto de grado. Además, este sistema de notificaciones por correo también se debe utilizar para que los usuarios confirmen su registro en la plataforma.
- **RF-026:** El sistema debe contar con un mecanismo de validación de errores en los formularios, asegurando que se informe al usuario sobre los campos incorrectos o incompletos. Además, debe recordar los datos ingresados previamente para evitar que el usuario tenga que rellenarlos nuevamente en caso de error.

4.5. Requisitos no funcionales

- **RNF-001:** La interfaz de usuario debe ser intuitiva y fácil de usar, permitiendo a cualquier tipo de usuario completar las tareas más comunes del sistema. Los elementos visuales, un diseño de menús organizado, así como mensajes de errores claros serán indispensables para el correcto uso del sistema.
- **RNF-002:** La seguridad en el sistema debe tener los mecanismos adecuados para proteger la información sensible de los usuarios, como la autenticación de usuarios, encriptación de claves y controles de acceso basados en roles.
- **RNF-003:** La estructura del código debe ser limpia, modular y bien documentada, facilitando la corrección de errores y futuras actualizaciones del sistema.
- **RNF-004:** Debe ser compatible con la versión actual del navegador web Google Chrome, garantizando el correcto funcionamiento del sistema de gestión.

4.6. Limitaciones

El prototipo está diseñado exclusivamente para operar con los requisitos y rúbricas establecidos para el área de posgrados en ingeniería de la Universidad Javeriana Cali. No será compatible con otros programas académicos ni con criterios de evaluación diferentes a los definidos para esta área.

5. Diseño del sistema de gestión

En este capítulo se presenta el proceso de diseño del sistema, abordando aspectos clave como la selección de la tecnología para la base de datos, la definición del modelo de datos, la elección de la tecnología para el diseño de las vistas y la elaboración de los primeros borradores de las interfaces.

5.1. Diseño de la arquitectura del sistema

El prototipo del sistema de gestión se desarrolló con MVC como principal patrón de diseño para la arquitectura del software. Este patrón permite la correcta separación de las obligaciones de cada parte del código otorgando un mejor orden y, por lo tanto, facilitando el mantenimiento. El diagrama de la **Figura 5** representa de forma clara cómo el sistema recibe peticiones a través de internet generadas por las acciones de los usuarios, convirtiendo al controlador en la parte central del esquema que comunica el modelo con la vista. El modelo es el encargado de todas las interacciones con la base de datos que permite obtener, actualizar y eliminar los datos almacenados. Una vez consultado el modelo, el controlador recibe la información, hace las validaciones correspondientes para el correcto funcionamiento del sistema, consulta a la vista correspondiente y le pasa la información para que la vista posteriormente la muestre. La vista a su vez se comunica con el controlador por medio de peticiones, enlaces y botones para la realización de diferentes funciones y acciones que requiera el sistema completando el ciclo en el diagrama.

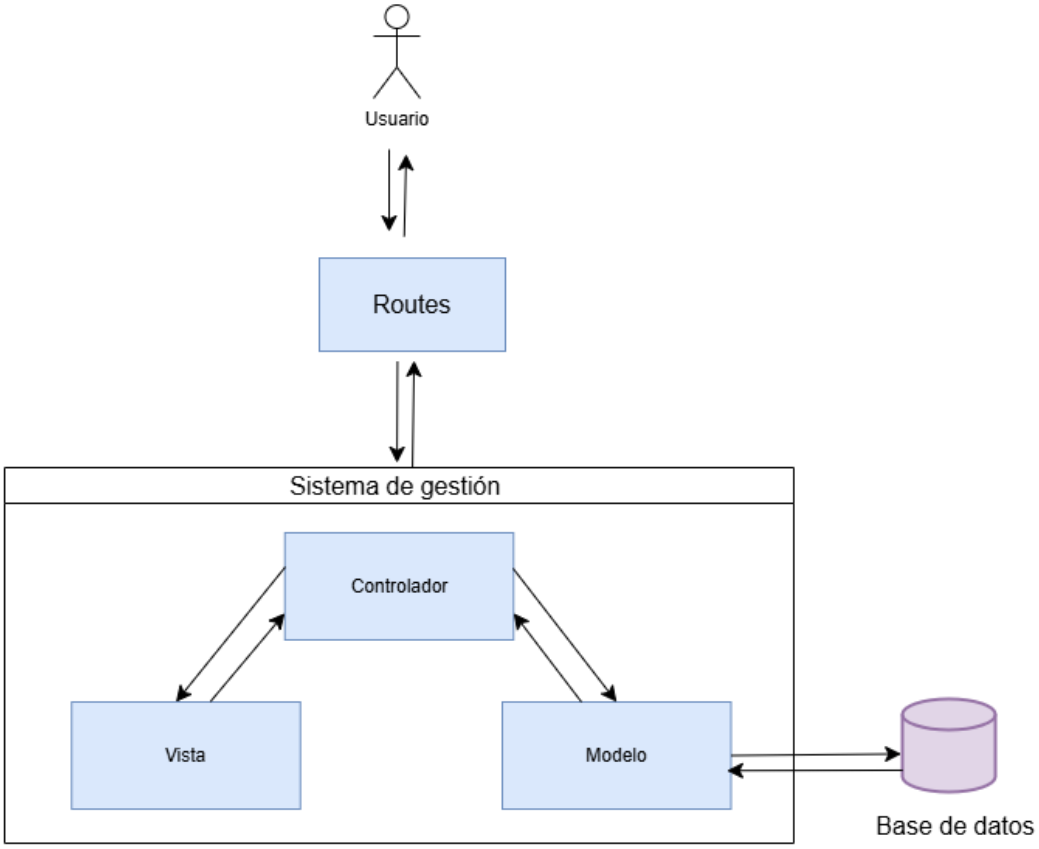


Figura 5: Diagrama de la arquitectura

Como se puede observar en el diagrama de la **Figura 5**, todo se genera con una o varias acciones de un usuario en el sistema. Por esta razón, se debe tener un "Routing" que defina las rutas válidas por las que un usuario puede navegar y acceder a las funciones correspondientes a la acción que este realice. Debido a lo anterior, el modelo MVC tiene el componente "Routes" en el que están registradas todas las URL o Endpoints que soporta el sistema de gestión. En Node Express una ruta válida se define especificando: el método HTTP, la URL en cuestión y la o las funciones que se van a ejecutar al momento de visitar esta ruta. Estas funciones se definen en el controlador, por eso este componente es el centro del diagrama.

5.2. Diseño de la base de datos

En esta sección se describe el diseño de la base de datos, incluyendo la tecnología seleccionada y la estructura definida para almacenar la información del sistema. Se presentan las tablas, sus atributos y relaciones, asegurando que cumplan con los requisitos del proceso de gestión de anteproyectos y proyectos de grado.

5.2.1. Selección de la base de datos

Para este proyecto, se utilizó MySQL como base de datos por diversas razones, destacando tanto sus ventajas técnicas como experiencias previas con bases de datos SQL. MySQL es una herramienta confiable, ampliamente utilizada y con una gran comunidad de soporte, lo que facilita la solución de problemas que aparecen en el desarrollo. Además, su capacidad para manejar relaciones entre tablas fue clave para estructurar el modelo de datos del sistema, donde los usuarios, anteproyectos/trabajos de grado, evaluadores y asignaciones están interconectados.

Durante el desarrollo, el entorno integrado MySQL Workbench fue esencial. Este software facilitó la creación y mantenimiento de la base de datos gracias a su interfaz intuitiva y herramientas prácticas.

Para simplificar la interacción con la base de datos, se utilizó el ORM Sequelize, una herramienta ampliamente utilizada en el ecosistema de Node.js. Los ORM permiten trabajar con bases de datos relacionales mediante objetos, su gran cantidad de métodos para listar, actualizar o eliminar datos simplificó en gran medida la comunicación entre los modelos y controladores.

5.2.2. Modelo de la base de datos

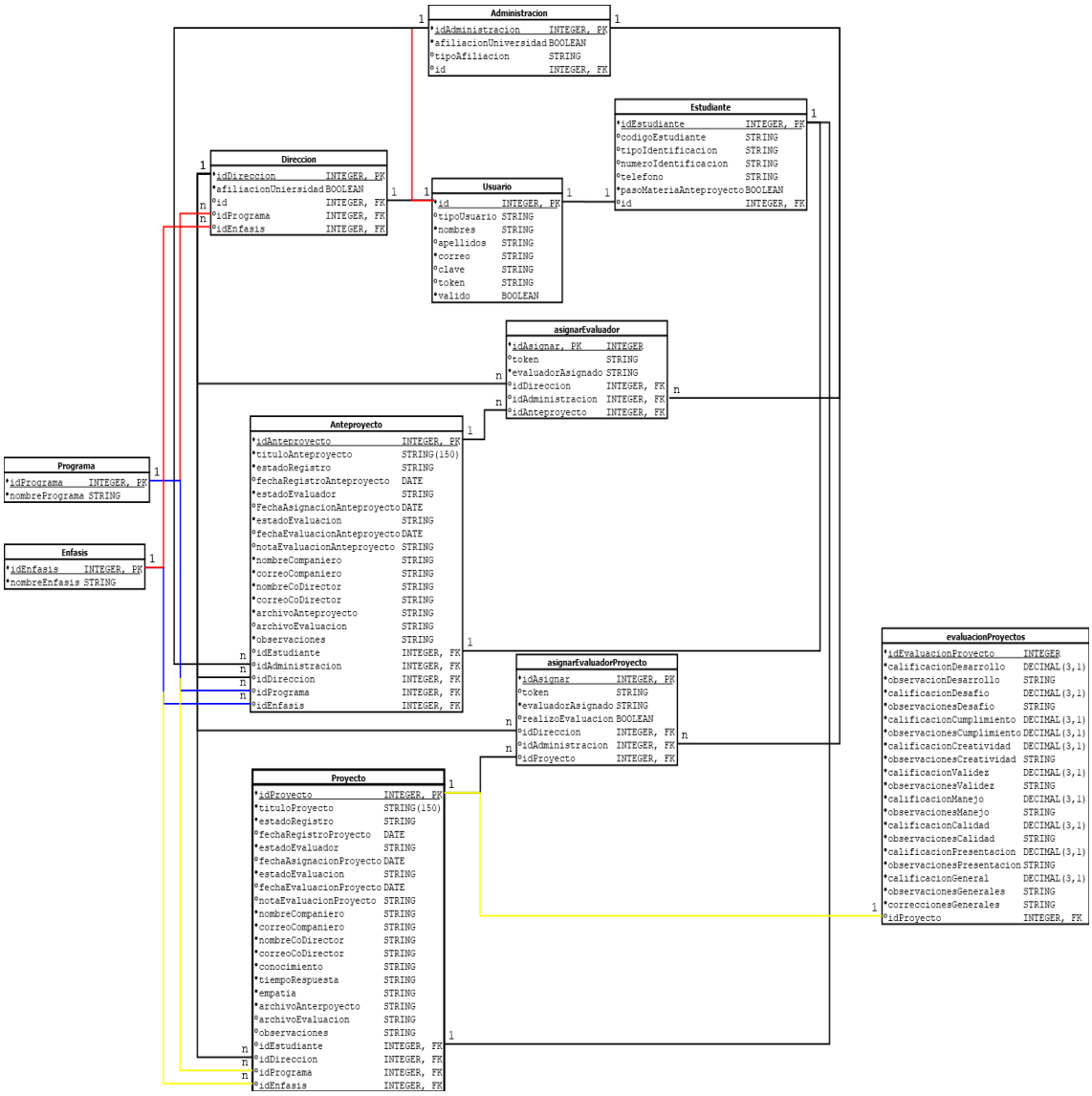


Figura 6: Modelo base de datos

El modelo de la base de datos en la **Figura 6** se desarrolló teniendo en cuenta los actores del sistema, sus funciones y las acciones que deben ocurrir en el proceso. Partiendo de esto, se realizó la transformación de los actores que hacen parte del diagrama de contexto en tablas con sus atributos correspondientes. Asimismo, se crearon las relaciones necesarias entre ellas para desarrollar todas las funciones establecidas en los requisitos del sistema de gestión. Los atributos de casi todas las tablas están basados en los campos de los formularios de registro y actas de evaluación para anteproyectos y proyectos de grado que utiliza actualmente la oficina de posgrados para llevar a cabo este proceso. Para explicar el modelo desarrollado, se va a dividir la explicación por grupos de tablas: Usuario y sus posibles roles, Anteproyecto, Proyecto, las asignaciones para el evaluador y la tabla EvaluacionProyecto.

- **Usuario y sus posibles roles:** El diseño se basa en una estructura centralizada con una tabla principal llamada *Usuario*, que se encarga de almacenar los atributos generales que comparten todos los tipos de usuarios como los nombres, apellidos, el correo y la clave para acceder al prototipo. Para representar los diferentes roles que puede tener un usuario se crearon tres tablas especificas: *Estudiante*, *Administración* y *Dirección*. Los atributos de la tabla *Estudiante* corresponden a datos personales y académicos como el código del estudiante o el número del celular personal. El atributo más importante de esta tabla es *pasoMateriaAnteproyecto* porque este requisito asigna un valor booleano, "true" si el estudiante ya cursó o está cursando la asignatura de Anteproyecto, o "false" si no la ha cursado; de esta forma se determina si el estudiante puede crear el usuario.

  En la tabla *Administración* se agrupan los atributos de los roles Administrador y Evaluador, esto debido a que ambos necesitan el mismo atributo adicional, el cual es su tipo de afiliación con la Universidad. Siguiendo lo anterior, se adicionó el atributo *tipoUsuario* en la tabla *Usuario* para diferenciar los perfiles registrados en el sistema. De esta forma, se evitó crear dos tablas con diferente nombre pero con los mismos atributos.

  En la tabla *Dirección* se agrupan los atributos de los roles Director de programa y Director de proyecto por la misma razón explicada anteriormente. Lo importante de esta tabla es la referencia a dos tablas nuevas llamadas *Programa* y *Énfasis*. El propósito de crear estas dos tablas independientes en vez de representar ambos datos como atributos en una tabla, es para evitar la redundancia y mantener la integridad de los datos, debido a que estos mismos datos son necesarios en al menos tres tablas diferentes en el modelo de la base de datos. De esta forma, no solo se conserva la normalización en los datos, sino que para cualquier modificación que requiera realizar la Universidad en los programas y en los énfasis en el área de posgrados, se verán reflejados en todas las referencias que tienen relación con estas tablas en el modelo. Cada una de las tablas almacena un identificador único, el nombre correspondiente y todas las relaciones están definidas de manera que todos usuarios en *Dirección*, *Anteproyecto* y *Proyecto* tengan asociados un programa y un énfasis.

  La relación entre la tabla *Usuario* y las tablas *Estudiante*, *Administración*, *Dirección* es 1:1, ya que se debe garantizar que cada usuario tenga exactamente un tipo asignado, cumpliendo entonces con el requerimiento de que un usuario solo puede tener un rol, y manteniendo la estructura del modelo.

- **Anteproyecto:** Esta tabla se encarga de almacenar la información necesaria para completar el proceso que conlleva un anteproyecto. Además de tener los atributos necesarios del formulario de registro usado por la Universidad, cada módulo del sistema tiene su atributo correspondiente de estado; así cada usuario pueda saber cómo va el proceso, y el sistema puede saber qué función debe mostrarse en un momento determinado. El valor de cada estado se asig-

na de acuerdo con el diagrama de actividades de la **Figura 3** De igual forma, cada estado tiene un atributo "fecha" para registrar el día en que se acepta cada módulo del sistema. El atributo *observaciones* guardar los comentarios que el administrador realizar cuando revisa el registro de un anteproyecto y el atributo *notaEvaluacionAnteproyecto* guardar la nota general que le asigne el evaluador, este dato debe guardarse ya que dependiendo del valor de la nota, el flujo de actividades en el sistema cambia. Los atributos más importantes en esta tabla son el atributo para guardar el archivo del anteproyecto *archivoAnteproyecto* y el atributo para guardar el acta de evaluación del anteproyecto *archivoEvaluacion* ya que representan el principio y el fin del proceso de gestión del anteproyecto y la diferencia principal entre los dos procesos que componen el trabajo de grado.

La decisión de diseño que se tomó para los archivos relacionados con los proyectos y sus respectivas evaluaciones corresponde a que no sean almacenados directamente en la base de datos, sino en carpetas dentro del sistema de archivos del proyecto. En la base de datos sólo se guardan los nombres de estos archivos. Esta decisión de diseño hace que la gestión del almacenamiento sea más eficiente, lo que mejora el rendimiento y facilita el acceso a los archivos desde distintas partes del sistema sin sobrecargar las consultas.

La relación entre *Anteproyecto* y *Estudiante* es 1:1, lo que también fue una decisión de diseño y de desarrollo en base al requerimiento RF-008. En el sistema de gestión cada estudiante sólo puede estar asociado a un anteproyecto, de manera que, si el estudiante requiere cambiar de tema, se debe eliminar primero el registro actual. En caso de que un estudiante trabaje con alguien más en el proceso, solo uno de ellos deberá tener una cuenta en el sistema y llevará a cabo el proceso. Por esta razón, tanto la tabla *Anteproyecto* como la tabla *Proyecto* contienen atributos que permiten guardar el nombre completo y el correo electrónico del compañero; esto permite que el sistema lo mantenga informado mediante notificaciones por dicho medio de comunicación cada que un módulo presente el estado "Aceptado". Se sigue el mismo razonamiento para el caso en que haya un Codirector, ya que en ambas tablas hay atributos para almacenar su nombre y correo electrónico.

Las relaciones entre Anteproyecto y Dirección "Director de proyecto", así como entre Anteproyecto y Administración "Evaluador", fueron creadas de manera que un director pueda tener a cargo varios Anteproyectos; de la misma forma, un evaluador puede evaluar varios anteproyectos.

- **Proyecto:** Esta tabla se encarga de almacenar la información necesaria para completar el proceso que conlleva un proyecto de grado. Su diseño presenta solo dos diferencias importantes respecto a la tabla *Anteproyecto*. La primera siendo que *Proyecto* contiene los atributos: *conocimiento*, *tiempoRespuesta* y *empatia*, los cuales conforman una evaluación que realiza el estudiante al director del proyecto de grado, respetando los valores del formulario de registro

que utiliza la Universidad. La segunda diferencia es que no hay referencia a la tabla *Administración*, debido a que en Proyecto deben estar asociados dos evaluadores y no se puede referenciar dos veces la misma tabla. Por esta razón se hace referencia a la tabla de asignación *asignarEvaluadorProyecto*.

- **AsignarEvaluador, asignarEvaluadorProyecto:** Estas tablas se encargan de guardar las asignaciones que un director de programa puede crear para los evaluadores de los Anteproyectos/Proyectos que pertenezcan a su respectivo programa. En ambas asignaciones se tiene un atributo *evaluadorAsignado* que indica el estado actual de la asignación, ya sea que está pendiente por responder, ha sido rechazada o ha sido aceptada por el evaluador.

  El atributo *token* que hace parte de ambas tablas, incluido también en la tabla *Usuario*, cumple la función de identificador de un solo uso, similar a los utilizados en transacciones bancarias. El objetivo de esto es generar un control seguro sobre ciertas acciones dentro del sistema mediante un enlace enviado por correo electrónico. En el caso de los usuarios, el token permite la confirmación de la cuenta recién creada, garantizando que sólo el propietario del correo electrónico tenga acceso. Para las asignaciones de evaluadores, el token se utiliza para que los evaluadores puedan aceptar o rechazar la evaluación de alguno de los proyectos a través de un enlace único, asegurando así un proceso ágil y seguro sin necesidad de acceder directamente al sistema.

  Para la evaluación del proyecto de grado se requieren dos evaluadores, por lo tanto en su respectiva tabla de asignación se encuentra el atributo *realizoEvaluacion*; esto permite conocer para cada evaluador si ya realizó la calificación.

  Las relaciones en estas tablas están hechas de forma que un director pueda crear varias solicitudes para evaluadores. Asimismo, un Anteproyecto/Proyecto y un evaluador pueden hacer parte de varias solicitudes de asignación.

- **EvaluacionProyectos:** Actualmente cuando se califica un anteproyecto se entrega la evaluación en un archivo siguiendo un rúbrica específica; no se deben realizar más acciones con estos datos y ahí termina el "módulo evaluación anteproyecto". Por esta razón no se hizo una tabla para almacenar los datos de la evaluación del anteproyecto. En cambio, el proyecto de grado es calificado por dos evaluadores que posteriormente deben publicar una sola evaluación. Como no hay forma de controlar que se hagan las evaluaciones al mismo tiempo, es necesario diferenciar cuándo ocurre cada una. Con motivo de solucionar esto se decidió guardar la primera evaluación para que, cuando se realice la segunda, se pueda recuperar y operar la información guardada; esta es la función de la tabla *EvaluacionProyectos*.

5.3. Diseño de las vistas

En esta sección se aborda el diseño de las vistas, detallando la tecnología elegida para su desarrollo y los borradores iniciales de la interfaz. Se busca que la presentación del sistema sea intuitiva y funcional, facilitando la interacción de los usuarios con las distintas funcionalidades disponibles.

5.3.1. Selección de tecnologías para las vistas

Para el desarrollo del front-end se utilizaron Pug y TailwindCSS, ya que permiten un mejor entendimiento del código debido a su sintaxis corta y clara, así como una menor redundancia de estilos, porque no se repiten clases CSS personalizadas innecesarias. Pug facilita la escritura de HTML al eliminar la necesidad de usar etiquetas, permitiendo definir la estructura del documento a través de indentación; esto hace que el código sea más limpio y fácil de mantener. Además, al ser un motor de plantillas que renderiza en el servidor, permite manejar la información proveniente de la base de datos de forma sencilla y con mayor seguridad en comparación con opciones como React, con el que sería necesario implementar medidas adicionales. TailwindCSS, por su parte, permite agilizar el diseño sin necesidad de escribir hojas de estilo separadas, ya que proporciona clases utilitarias que permiten aplicar estilos directamente en el código.

5.3.2. Mock-ups para las vistas

A continuación se presentan y explican los mock-ups utilizados como base para el diseño de las vistas del sistema. Estos modelos van a servir como base para visualizar la estructura y funcionalidad de las interfaces antes de su desarrollo.

Para su elaboración, se utilizó la herramienta de diseño gráfico en línea Canva. Para el diseño de todas las vistas se utilizaron el azul, blanco y amarillo como colores principales en el prototipo; el color gris, por su parte, cumple la función de separador. Cada barra de navegación cuenta con el logo de la Universidad Javeriana Cali en la parte superior izquierda, evidenciando que este prototipo fue diseñado en base a las necesidades específicas y requisitos de la institución.

Login

La **Figura 31** corresponde al diseño del inicio de sesión, el cual se realizó en base al requisito RF-017, donde se especifica los datos necesarios para ingresar al prototipo (correo electrónico y contraseña). Desde esta vista también se puede acceder al registro de un usuario y a la función de recuperar la contraseña.

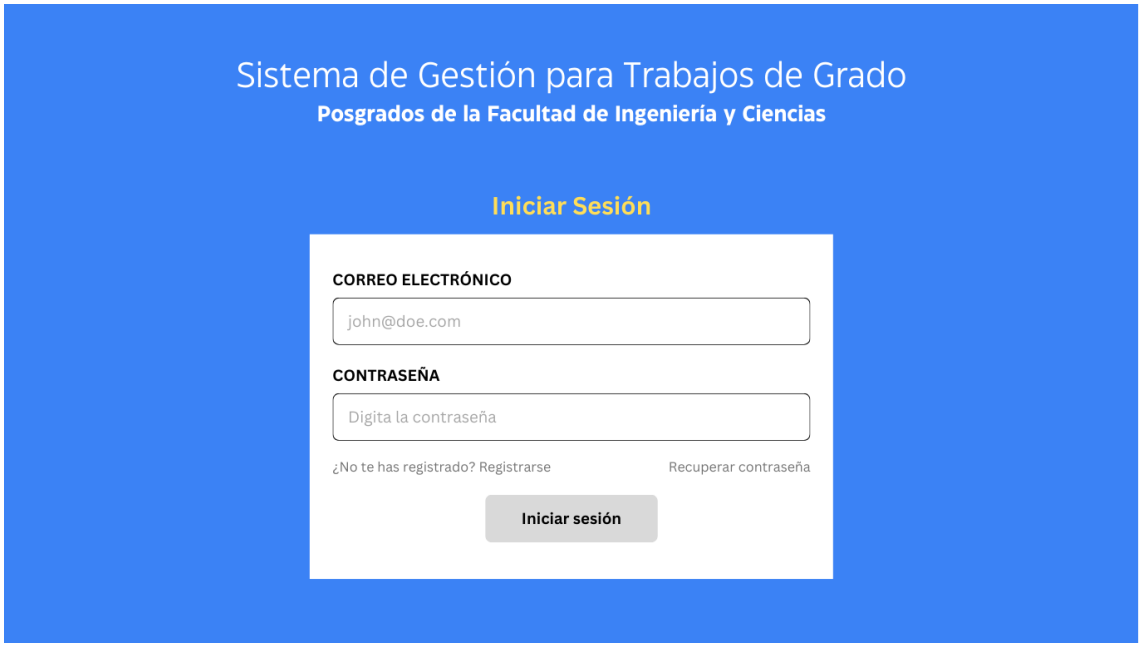


Figura 7: Diseño Login

Barra de avance

El diseño de la **Figura 8** hace parte de las vistas de los estudiantes y tiene como objetivo proporcionar una visión general de su progreso en los distintos estados de cada proyecto. Cumpliendo con el requisito RF-022, se evidencian los elementos visuales, que permanecen disponibles y permiten al estudiante acceder y utilizar cada módulo del proceso. El elemento visual correspondiente al proyecto de grado permanecerá bloqueado hasta que se complete el anteproyecto.

Figura 8: Diseño progreso

Formulario

El diseño en la **Figura 9** corresponde a la estructura que siguen todos los formularios del prototipo, diferenciando cada uno con su título y campos correspondientes. En este ejemplo se muestra el formulario para el registro de un usuario; dependiendo de la selección del tipo de usuario aparecerán nuevos campos en el formato.

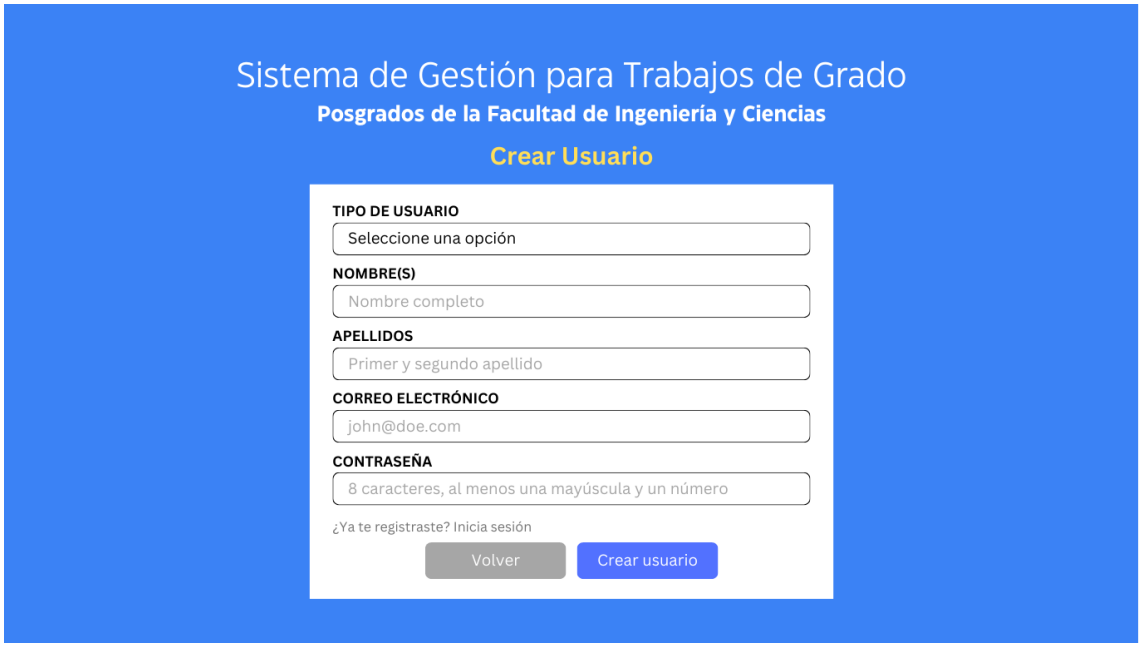


Figura 9: Diseño Formulario

Vista general de los usuarios

Como parte de los requisitos RF-001, RF-003, RF-007, RF-012 y RF-014, la información general de los proyectos se debe mostrar en recuadros de color blanco y de forma secuencial. Cada proyecto se va a diferenciar por medio de una línea divisoria de color gris. La vista en la **Figura 10** sirve como ejemplo de este diseño. Sin embargo, al tratarse de la vista del estudiante, la separación no es entre proyectos sino por módulos, cada uno con su respectivo título. Como se ve en el diseño, cada módulo tiene su respectivo estado, respetando el requisito RF-023. Cuando se completa el módulo de evaluación se muestra un enlace dónde el estudiante puede descargar su calificación, cumpliendo el requisito RF-011.

Con base en el requisito RF-019 se creó la barra de navegación. Esta permite que cada usuario tenga acceso a su función principal: las asignaciones de director para el director de proyecto, la revisión de registros para los administradores, la asignación de evaluadores para el director de programa y, por último, los evaluadores pueden acceder tanto a las asignaciones como a la evaluación de los distintos proyectos. Además, cada uno de los usuarios que cuenten con proyectos asociados podrán acceder a ellos mediante el enlace “verTodo”.

Figura 10: Diseño vista general de un estudiante

Vista para mostrar información detallada

Algunas páginas tienen enlaces que muestran un modal (cuadro de diálogo que oscurece el fondo) que permitirá a los usuarios ver información más detallada, ya sea de proyectos u otros usuarios como se muestra en la **Figura 11**.

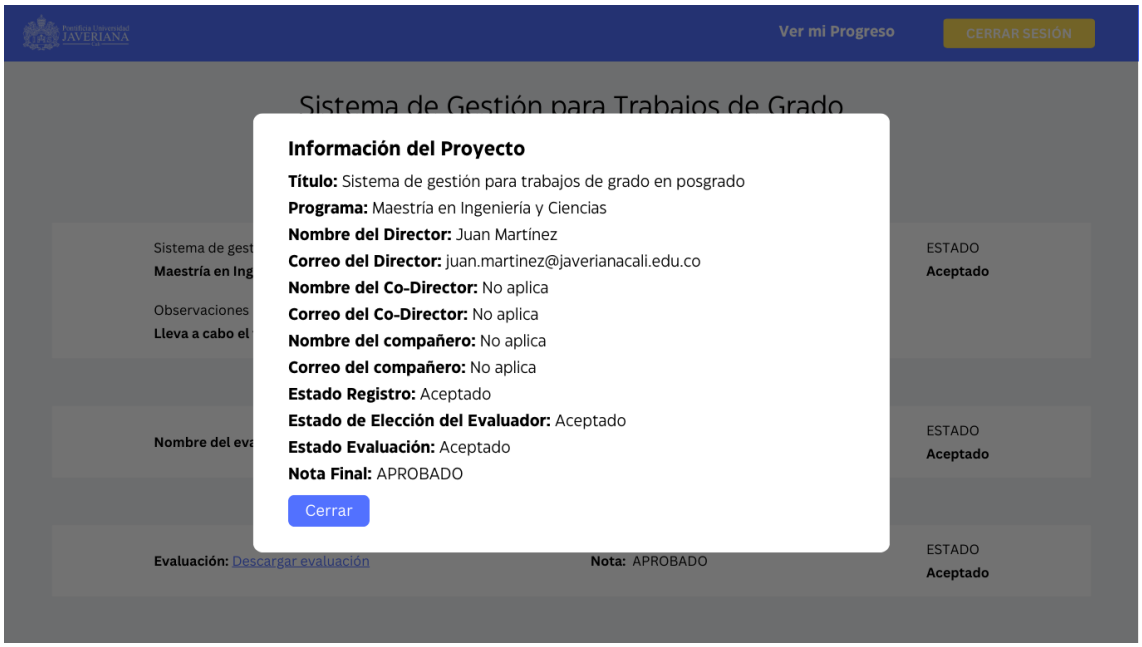


Figura 11: Diseño vista información detallada anteproyecto

6. Desarrollo del sistema de gestión

En este capítulo se abordará el proceso de desarrollo del prototipo, partiendo de los requerimientos establecidos en el capítulo cuatro. Se detallará cómo se implementaron estos requerimientos, y se justificarán las herramientas, dependencias y librerías utilizadas para garantizar el correcto funcionamiento del sistema.

6.1. Selección de la tecnología para el prototipo

Se seleccionaron Node.js y Express para el desarrollo del prototipo del sistema de gestión principalmente por su eficiencia y flexibilidad en aplicaciones web. Node.js permite manejar múltiples solicitudes de manera asíncrona, lo que mejora el rendimiento y la escalabilidad del sistema. Además, es un entorno ampliamente utilizado en la industria del desarrollo web actual, por lo que presenta actualizaciones de forma continua junto con JavaScript. Lo anterior se ve reflejado en la gran cantidad de documentación y guías de aprendizaje que se pueden encontrar; esto se presenta como una ventaja importante en el paso del desarrollo de software, por ejemplo para el manejo y corrección de errores, o creación de funciones complejas.

Express simplifica la creación de rutas y middleware, permitiendo una estructura clara y modular. A diferencia de otros frameworks, Express permite desarrollar lo que sea necesario de acuerdo con los requisitos del trabajo y, para funcionalidades específicas, descargar únicamente los paquetes indispensables mediante NPM, el gestor de instalación para dependencias de Node. Esto agiliza el desarrollo al evitar la carga de librerías innecesarias.

6.2. Configuraciones iniciales

El primer paso en el desarrollo del prototipo fue la instalación y configuración de las tecnologías necesarias para cada uno de sus componentes: back-end, front-end y base de datos.

6.2.1. Backend:

Se instaló la versión 20.17.0 LT de Node.js, que corresponde a la más completa al momento de desarrollar el prototipo. Además es la versión recomendada por la página oficial de Node. Al momento de instalar se creó un archivo nombrado package.js, el cual es de configuraciónm y contiene la información general del proyecto y

gestiona las dependencias necesarias.

La primera dependencia que se descargó fue Express, la cual permitió crear el servidor de desarrollo. Para hacer esto sólo se debe definir el puerto, en este caso: "3000", y enlazarlo con la aplicación web creada, que para este prototipo recibe el nombre de "sistema" como se puede ver en la **Figura 12**. Ahora cada vez que se ejecute el código, se podrá acceder al servidor a través de la ruta http://localhost:3000/; es necesario aclarar que esta es una ruta para el desarrollador.

```
import express from "express"

/*Se crea la aplicacion web*/
const sistema = express();

/*Definir un puerto y arrancar el sistema*/
const port = 3000;
sistema.listen(port, function(){
    console.log("El sistema esta funcionando en el puerto : " + port );
});
```

Figura 12: Código para iniciar el servidor en Express.js

6.2.2. Front-end:

Desde NPM se instaló y habilitó Pug en el sistema. TailwindCSS se instaló de la misma manera; sin embargo, al terminar la descarga se generó un archivo "tailwind.config.js" que solicita, en el atributo *content*, especificar el lugar donde van a estar almacenados los templates del sistema. Para almacenar las vistas se creó la carpeta "views". TailwindCSS escanea todos los "templates.pug" en la carpeta de vistas, buscando las clases definidas y transformándolas en un mismo archivo CSS como se configuró en la **Figura 13**.

```
/** @type {import('tailwindcss').Config} */
export default {
  content: ["./views/**/*.pug"],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

Figura 13: Código para configurar las vistas con Pug

6.2.3. Base de datos:

Al momento de instalar Sequelize, se creó la carpeta "config" así como el archivo "dataBase.js"; en este archivo se realizó la conexión con la base de datos. Como se puede observar en la **Figura 14** para la conexión se debe crear una nueva instancia, la cual toma 4 parámetros: el nombre de la base de datos creada, el usuario, la clave de la base de datos y un objeto en donde se especifica a Sequelize la configuración de la conexión. En dicha configuración se especificó el servidor, el tipo y el puerto que, por defecto para MySQL, es el "3306". Por último, se definió el atributo "pool", el cual se emplea en Sequelize para configurar el tiempo y la cantidad de conexiones asíncronas a la base de datos antes de detectarlo como un error. Para el prototipo que se desarrolló, el máximo de conexiones son cinco, no tiene mínimo de conexiones, deben pasar treinta segundos sin poderse conectar para marcar un error y deben pasar diez segundos para finalizar una conexión.

```
import { Sequelize } from "sequelize"

const dataBase = new Sequelize("sistema", "root", "Forzainter22!",{
    host: "localhost",
    port: 3306,
    dialect: "mysql",
    pool: {
        max: 5,
        min: 0,
        acquire: 30000,
        idle: 10000
    }
});
```

Figura 14: Código para configurar la conexión y conexiones de la base de datos

6.3. Desarrollo de los modelos

Dado que la arquitectura del proyecto sigue el patrón MVC, el desarrollo comenzó con la implementación de los modelos, los cuales representan la estructura de los datos en el sistema. En esta sección se explica cómo se definieron los modelos *Usuarios* y *Anteproyecto* a modo de ejemplo y como base para los demás modelos del sistema definidos en la base de datos.

6.3.1. modelo Usuario

Con la función "define" de Sequelize se instancia la tabla que se va a crear; esta función, como se ve en la **Figura 15**, recibe dos parámetros: el nombre que tendrá

la tabla en la base de datos y un objeto en el que se especifica cada atributo en la tabla. Cada atributo del modelo es configurable, pero es obligatorio especificar el tipo "type" del atributo; Sequelize simplifica considerablemente este apartado, al utilizar etiquetas sencillas y luego transformarlas en su equivalente en MySQL. Lo anterior se puede observar, por ejemplo, al usar "STRING" para definir un tipo "VARCHAR". La configuración "allowNull" en algunos casos tiene como valor "false", lo que implica que, al crear un nuevo usuario en el controlador, los atributos que contengan dicha configuración no pueden estar vacíos.

```
import { DataTypes } from "sequelize"
import dataBase from "../config/dataBase.js"

const Usuario = dataBase.define("usuario", {
    tipoUsuario:{
        type: DataTypes.STRING
    },
    nombres:{
        type: DataTypes.STRING
    },
    apellidos:{
        type: DataTypes.STRING
    },
    correo:{
        type: DataTypes.STRING,
        allowNull: false
    },
    clave:{
        type: DataTypes.STRING,
        allowNull: false
    },
    token:{
        type: DataTypes.STRING
    },
    valido:{
        type: DataTypes.BOOLEAN
    }
});

export default Usuario
```

Figura 15: Código para definir el modelo *Usuario*

Si se comparan los atributos del modelo con los del diseño de la base de datos se podrá notar que únicamente falta la llave de prioridad, *id*, para la tabla *Usuario*. Sequelize identifica la falta de la llave primaria y la crea automáticamente como un entero único que no puede ser vacío y es incremental. Aunque también se puede

definir la llave primaria desde el modelo como se muestra en el siguiente modelo.

6.3.2. Modelo Anteproyecto

Lo más importante en la creación de este modelo es el atributo *idAnteproyecto*; este ID se define como un tipo de dato "UUID", el cual genera una cadena de 32 caracteres hexadecimales que se dividen en 5 grupos, por ejemplo: "550e8400-e29b-41d4-a716-446655440000". Este cambio al ID numérico secuencial se realizó para que se pudieran utilizar los identificadores de los proyectos en las URL del sistema. Esto permite acceder a un dato que, a su vez, indique con qué proyecto se está trabajando, sin perder seguridad como podría pasar con un ID convencional. En la **Figura 16** se muestra de manera general la definición del modelo *Anteproyecto*. El tipo "DATEONLY" guarda en MySQL una fecha en el formato "AA-MM-DD" y con "defaultValue" se define un valor inicial al atributo.

```
import { DataTypes } from "sequelize";
import dataBase from "../config/dataBase.js"
import Estudiante from "./Estudiante.js";

const Anteproyecto = dataBase.define("anteproyecto",{
    idAnteproyecto:{
        type: DataTypes.UUID,
        defaultValue: DataTypes.UUIDV4,
        allowNull: false,
        primaryKey: true
    },
    estadoRegistro:{
        type: DataTypes.STRING,
        defaultValue: "Sin Registro"
    },
    fechaRegistroAnteproyecto:{
        type: DataTypes.DATEONLY
    },
    idEstudiante: {
        type: DataTypes.INTEGER,
        allowNull: false,
        unique: true,
        references: {
            model: Estudiante,
            key: "idEstudiante"
        }
    }
}
);

export default Anteproyecto
```

Figura 16: Código para definir el modelo *Anteproyecto*

Las relaciones se pueden definir directamente en los modelos, sin embargo, para este prototipo se crean en el archivo "Relaciones", dentro de la carpeta "models". En este caso, lo que se configura desde el modelo es la referencia *idEstudiante*; las referencias son creadas automáticamente por Sequelize cuando se define una relación, pero en este caso especial, al igual que en todas las relaciones 1:1, se debe configurar la referencia manualmente como única, evitando así que Sequelize cambie la cardinalidad de la relación.

6.4. Definición de las relaciones

El objetivo de esta sección es mostrar un ejemplo de cómo se define cada tipo de relación (1:1, N:1, 1:N) empleando Sequelize. Previamente en el capítulo 5 se explicaron las relaciones que debía tener cada una de las tablas en la base de datos, por lo tanto, no se mostrará cada una de las relaciones creadas para el prototipo.

```
/// Relación 1:1
Usuario.hasOne(Estudiante, {foreignKey: "id",});
Estudiante.belongsTo(Usuario, {foreignKey: "id"});
/// Relacion N:1
Proyecto.belongsTo(Direccion, {foreignKey: "idDireccion"});
/// Relacion 1:N
Direccion.hasMany(AsignarEvaluador, { foreignKey: "idDireccion" });
AsignarEvaluador.belongsTo(Direccion, { foreignKey: "idDireccion" });
```

Figura 17: Código para definir relaciones con Sequelize

Como se puede observar en la **Figura 17**, el modelo *Estudiante*, al igual que todos los tipos de usuario, tiene una relación 1:1. La función "hasOne" permite establecer que un *Usuario* sólo puede estar asociado a un *Estudiante*. La función "BelongsTo" determina que cada *Estudiante* pertenece a un *Usuario*. La definición de esta relación parece redundante, pero es necesaria en Sequelize para acceder a ambos modelos en ambas direcciones en caso de ser necesario. Algo muy similar sucede con la relación entre *Dirección* y *Asignación*: la función "hasMany" indica que un director de programa puede estar asociado a múltiples asignaciones, permitiendo acceder a la información desde ambos modelos.

6.5. Definición de las rutas

A continuación se mostrará cómo se define una ruta con express.js. De acuerdo con la **Figura 18**, como primer paso se creó una carpeta llama "routes" que aloja los archivos que definen las rutas por las que cada usuario del sistema puede navegar. La función "Router" instancia la creación de rutas en el archivo; al crear una nueva ruta se debe definir un método HTTP: "GET", "POST" o "PUT". Este método recibe la dirección URL junto con una o varias funciones que se van a ejecutar cada vez que se visite una ruta determinada. Dichas funciones se escribirán en los controladores de cada modelo.

La elección del método depende de la petición que se haga al servidor. Se empleó el método "GET" para mostrar imágenes, el método "POST" para enviar información al servidor y el método "PUT" para actualizar información del servidor.

```
import express from "express";
import {formularioLogin} from "../controllers/usuarioController.js"

const router = express.Router();

router.get("/login", formularioLogin);
```

Figura 18: Código para definir rutas con Express.js

Express maneja las solicitudes HTTP internamente y siempre necesita saber qué petición recibió y cómo se debe responder. Por esta razón, siempre que se define una función en una ruta, Express recibe automáticamente los parámetros "req" (solicitud) y "res" (respuesta). Estos parámetros permiten capturar los datos que envía el cliente y transformarlos de acuerdo con lo que necesite.

6.6. Desarrollo de los controladores

Continuando con el diseño de la arquitectura, en esta sección se explica cómo se desarrolló cada controlador, así como las funciones más importantes para el correcto funcionamiento del prototipo. Cada modelo de usuario en el sistema cuenta con un controlador correspondiente, encargado de gestionar operaciones como el manejo de solicitudes, la validación de información y la interacción con la base de datos.

6.6.1. Controlador del usuario

El archivo "usuarioController" es un controlador que contiene la implementación de las funciones y validaciones necesarias para la creación, el registro e inicio de sesión de los usuarios en el sistema.

Lo primero que se desarrolló en el controlador fue el registro de los usuarios, siguiendo los requisitos RF-017 y RF-018. Para esto se creó un formulario con todos los campos definidos en el modelo *Usuario*, que incluyera un campo de selección para elegir el tipo de usuario que se está creando; dependiendo del tipo elegido, se muestran los campos restantes correspondientes a cada modelo. El único rol que no se puede crear desde el registro es el de un evaluador, ya que desde el desarrolló se tomó la decisión de que únicamente el director de programa puede crear usuarios tipo evaluador a través de las de asignaciones que realice.

Todos los formularios en este prototipo siguen la misma estructura. Cada campo incluye el atributo *name = "correo"*, lo que asegura que el valor ingresado tenga un identificador. De este modo, al enviarse mediante el método "POST", puede

manipularse a través de *req.body.correo* en este caso específico. Las verificaciones de cada uno de los campos se hacen con la dependencia express-validator[11]; esta dependencia es un conjunto de middlewares de express.js que contiene una extensa colección de validadores, los cuales se pueden crear de acuerdo con las necesidades del prototipo empleando la función "check()" y especificando el atributo del *req.body*, permitiendo verificar condiciones como la existencia del campo, su formato y longitud, entre otras reglas de validación que van a depender del dato.

Las validaciones, por ejemplo, vuelven de cierta forma obligatorio que el estudiante indique al sistema si ya cursó o está cursando "Anteproyecto para posgrados". Esto es de gran importancia, ya que, en caso de que no cumpla con esta condición, no puede registrarse en el prototipo satisfaciendo el requisito RF-006.

Después de asegurar que se pasaron correctamente todas las validaciones se procede a verificar si el usuario no se ha registrado antes en el sistema. Para lograr esto, se hace una consulta con el correo registrado; se emplea el correo en lugar del nombre o apellido, ya que estos últimos son registros que se pueden repetir entre usuarios, sin embargo el correo debe ser único. Si el usuario no existe en el sistema se procede a crearlo: Sequelize utiliza la función "create()", que recibe un objeto. En dicho objeto se especifican los atributos definidos en el modelo, que deben corresponder a los datos recuperados del formulario. Luego se crea el tipo de usuario especificado en el formulario, validando primero que sus atributos correspondientes fueron registrados correctamente. La **Figura 19** muestra un ejemplo de cómo se implementó la creación un usuario nuevo.

```
const existeUsuario = await Usuario.findOne({ where: {
correo: req.body.correo } });

    const { tipoUsuario, nombres, apellidos, correo, clave} = req.body;
    const usuario = await Usuario.create({
        tipoUsuario,
        nombres,
        apellidos,
        correo,
        clave,
        token: generarId()
    });
```

Figura 19: Código para crear un usuario

La clave de un usuario es un dato que ni siquiera el desarrollador del sistema debe ver. Debido a esto, se encriptó la clave de todos los usuarios, utilizando la dependencia bcrypt[12]. Esta dependencia está diseñada para encriptar las claves mediante

un algoritmo de hashing. La función "hash()" va a realizar el proceso: recibe el valor a encriptar, que en este caso es *usuario.clave*, y recibe una variable "salt", la cual genera un número de rondas que corresponde a las veces que se va a encriptar la clave. Para el prototipo desarrollado, se dejó el valor por defecto (diez). A medida que el número aumenta, la encriptación se vuelve más segura; sin embargo, el uso de valores muy altos incrementaría la complejidad computacional del prototipo, limitando su viabilidad. Este proceso se encuentra definido dentro del modelo *Usuario*, configurado de manera que encripte la clave antes de la creación de un nuevo usuario.

Cada usuario tiene un token único que forma parte del proceso de confirmación del registro en el sistema, el cual se genera manualmente. En la carpeta "tools" se creó la función "generarId()", la cual crea un token usando "Math.random().toString(32)", que convierte un número entre 0 y 1 en una cadena de caracteres en base 32. Esta operación se hace dos veces y se le retira el punto decimal con "substring(2)".

Para la confirmación del registro de un usuario en el prototipo se creó el archivo "envioCorreos.js", en el que se utiliza la dependencia Nodemailer[13] de Node.js, la cual permite enviar correos electrónicos desde aplicaciones back-end mediante distintas plataformas. Con base en esta dependencia se desarrolló una función llamada "correoRegistro". Como se observa en la **Figura 20**, esta función recibe los datos del usuario, incluyendo su nombre, apellido, correo y un token único de verificación. Luego, transporta de correo con las credenciales de autenticación y lo envía al usuario con un enlace de confirmación. Dicho enlace contiene el token generado previamente, el cual redirige al usuario a una ruta específica del sistema donde podrá validar su cuenta. Hasta que el usuario no confirme su registro, no podrá acceder al sistema.

Durante el desarrollo se sigue utilizando esta función de envío de correos, para la explicación se va a referenciar nuevamente la **Figura 20**. Para las pruebas con el prototipo se empleó una la plataforma Mailtrap, que permite ensayar el envío de correos para desarrollo web.

```
import nodemailer from "nodemailer"

const correoRegistro = async (datos) => {
    const transport = nodemailer.createTransport({
        host: "sandbox.smtp.mailtrap.io",
        port: 2525,
        auth: {
            user: "cee913fe237feb",
            pass: "8e60d31679837a"
        }
    });
    const {nombres, apellidos, correo, token} = datos
    await transport.sendMail({ // Aqui se diseña el correo
        from: "Sistema de gestion para trabajos de grado",
        to: correo,
        subject: "Confirma tu cuenta en el sistema de gestion",
        text: "Confirma tu cuenta en el sistema",
        html:`
            <p> Hola ${nombres, apellidos}, este correo fue generado
            para que confirmes tu cuenta</p>
            <p> confirma tu cuenta:
            <a href= "http://localhost:3000/auth/validacion/${token}"
            >Confirmar Cuenta</a> </p> `
    })
}
```

Figura 20: Código para enviar correos

Para permitir que los usuarios cambien su clave en el sistema, se implementó un formulario en el que deben ingresar su correo electrónico. En el controlador se definió que, si el usuario está registrado en el sistema, se generará un token mediante la función "generarId()". Al igual que en el proceso de registro, se enviará un correo al usuario para que este confirme si desea cambiar su clave. Si es así, el enlace lo redirigirá a un nuevo formulario donde deberá ingresar su nueva contraseña, para luego completar el proceso con éxito. Finalmente, la nueva clave se encripta con bcrypt y se sobrescribe en la base de datos.

Para el inicio de sesión, se creó un formulario con los campos de correo electrónico y clave. Para verificar que el usuario que intenta ingresar se encuentra registrado, se realiza una consulta en la base de datos con el correo electrónico; luego se verifica la validez del usuario mediante el atributo *valido* = *true*, corroborando que ha confirmado su cuenta. En caso de que sea válido, se genera un JSON Web Token(JWT)[14], el cual es un estándar de seguridad basado en un token firmado digitalmente. De acuerdo con la **Figura 21**, el JWT se crea usando la función "jwt.sing()", que recibe un identificador, en este caso el ID del usuario, una clave y tiempo de expiración de

un día. El JWT generado se almacena en una cookie.

```
import jwt from "jsonwebtoken"

// Generar JsonWebToken
const generarJWT = function(id){
    return jwt.sign({id}, "septiembrediezdosmiltres",{expiresIn: "1d"});
}
```

Figura 21: Código para generar el JWT

Luego, cumpliendo con el requisito RF-021, se creó el archivo "proteccionRutas.js", el cual verifica si el JWT es válido mediante la función "verify()", que recibe el JWT y la contraseña. En caso de serlo, la función devuelve el identificador (ID) y luego realiza la consulta del usuario, así como de su tipo correspondiente. La información de las consultas del usuario que está navegando en el sistema se guarda como una solicitud (req). Esta función se exporta con el nombre "protegida" y se puede utilizar en todas las rutas del sistema que requieren autenticación.

6.6.2. Controlador de los Anteproyectos y Proyectos de grado

El archivo "procesoController" es un controlador que contiene las funciones que el estudiante necesita y requiere para llevar a cabo cada uno de sus procesos, como registro, edición y eliminación.

Con base al requisito RF-008, los formularios para registrar un anteproyecto y un proyecto de grado son muy similares con la excepción de algunos cambios, ya que al registrar un proyecto de grado no es necesario saber el programa, el énfasis o el director, debido a que estos datos se tienen desde que se registró el anteproyecto y no deben cambiar.

Al momento de registrar un anteproyecto se verifica que *idEstudiante* no tenga un registro existente en la tabla *Anteproyectos*, de esta manera se puede confirmar que tenga únicamente un proyecto asociado. En segundo lugar, se verifica que el director de proyecto se encuentre registrado en el sistema para entonces referenciarlo. En caso de que no lo esté, se cancela el registro y se enviará un correo al director de proyecto especificado con una función similar a la explicada en la **Figura 20**. El correo incluye los datos del estudiante, el título del anteproyecto, un mensaje explicando la solicitud y, por último, un enlace que lo llevará al inicio de sesión del prototipo para que realice el registro.

El registro de archivos para ambos proyectos se realiza utilizando la dependencia multer[15], un gestor para el guardado de archivos en Node.js. Antes de emplearlo, es necesario verificar que la extensión del archivo sea .pdf; luego, como se observa en la **Figura 22**, se define la función llamada "almacenarArchivo" que, usando "multer.diskStorage", configura la ruta en la que se guardarán los archivos y el nombre que esta va a tener. El lugar de almacenamiento en ambos casos será una carpeta ubicada en el apartado público del proyecto, de manera que si este prototipo llega a estar alojado en un servidor, todos los usuarios que tengan acceso al sistema de gestión podrán acceder también a estas carpetas. El nombre del archivo será un ID con la función "generarId()" explicada anteriormente; esto debido a que en una carpeta no pueden existir dos archivos con el mismo nombre.

Si el registro del anteproyecto se logra correctamente, el estado en el Módulo de Registro de Anteproyecto pasaría a "Pendiente de confirmación Director", ya que el director de proyecto debe aceptar esta nueva asignación. En caso de registrar un proyecto de grado, el estado en el Módulo de Registro proyecto pasa directamente a "Pendiente de revisión".

```
import multer from  "multer"
import { generarId } from "../tools/tokens.js";

const almacenarArchivoAnteproyecto = multer.diskStorage({
    destination: function(req, file, cb){
        cb(null, "./public/archivosAnteproyectos/");
    },

    filename: function(req, file, cb){

        cb(null, generarId() + ".pdf");
    }

});

const subirAnteproyecto = multer({
    storage: almacenarArchivoAnteproyecto
});
```

Figura 22: Código para almacenar archivos con Multer

Para cumplir con los requisitos RF-009, RF-010 y RF-020 se explicará cómo se desarrollaron las funciones de edición y eliminación de proyectos.

Para la edición de los proyectos, se envía el ID por medio de la etiqueta "href", mostrándolo así como un parámetro en la URL. Antes de darle acceso al estudiante,

se recupera el identificador con *req.params.id* y se realiza una consulta para verificar si el proyecto existe. Luego, con el *idEstudiante* almacenado en el archivo "proteccionRutas" (gracias al JWT durante el inicio de sesión), se confirma que el usuario que solicita la edición es de tipo estudiante y que, además, es el mismo que registró el proyecto.

Después de verificar la información, el estudiante puede acceder a un formulario de edición, en el que puede modificar los datos más importantes del registro. Mediante un enlace puede ver el archivo actualmente registrado, y cuenta con un apartado para subir un nuevo archivo si es necesario. Si las validaciones en el formulario son correctas, el controlador actualiza la información del proyecto con la función "set()", la cual se utiliza porque permite modificar de manera eficiente los valores de un objeto en la base de datos. Lo anterior asegura que solo los campos especificados sean actualizados sin afectar el resto de la información almacenada.

6.6.3. Controlador de los directores de programa y de proyecto

En este controlador se explicará cómo se desarrollaron las respuestas de asignaciones de dirección para los directores de proyecto, así como el desarrollo de las asignaciones para los evaluadores por parte de un director de programa.

Para el director de proyecto se desarrolló una función que permite responder las asignaciones para dirigir un anteproyecto, cumpliendo con el requisito RF-002. En la carpeta "panelDirector", se encuentra la vista "direccionAnteproyectos´´, la cual contiene una sección script en la que se puede escribir funciones en JavaScript y corresponde a la **Figura 23**; en esta función se seleccionan todos los botones con la clase "cambiar-direccion-aceptar". Cuando un director de proyectos interactúa con el botón, se obtiene el *idAnteproyecto* desde la clase "data-idAnteproyecto" creada únicamente para responder asignaciones. Luego, se construye una URL con este ID, y se realiza una solicitud PUT utilizando la función "fetch()".

```
var estadosAceptar = document.querySelectorAll(
".cambiar-direccion-aceptar");

estadosAceptar.forEach(boton => {
    boton.addEventListener("click", async function (e) {

        console.log("Presionando botón");

        // Obtén el atributo data-idAnteproyecto
        const id = e.target.dataset.idanteproyecto;
        console.log(id);

          // Construye la URL de la petición
        peticion = `/panel-director/aceptado/${id}`;

        try {
            const respuesta = await fetch(peticion, {
                method: "PUT",
            });
            if (!respuesta.ok) {
                console.error("Error en la petición", respuesta.status);
            }
        } catch (error) {
            console.error("Error en la conexión:", error);
        }
        window.location.reload();
    });
});
```

Figura 23: Código para responder asignaciones

El método PUT se utiliza para actualizar o reemplazar un recurso existente en el servidor. En este caso, cuando un director de proyectos acepta una asignación, el servidor recibe la petición y la procesa mediante la función definida en "routes". La función encargada de esta acción es asignarDireccion, la cual primero verifica que el usuario tenga el rol de director de proyecto y que, además, sea el mismo director al que se le envió la asignación. Una vez confirmadas estas condiciones, se actualiza el estado del Módulo de Registro de Anteproyecto a "Pendiente de revisión", permitiendo que el estudiante continúe con el proceso.

Para rechazar la asignación se utiliza igualmente el método PUT, con la única diferencia de que la URL y la función en "routes" cambian. Luego de realizar las verificaciones descritas en el párrafo anterior, el estado del Módulo de Registro se actualiza a "Rechazo participación", impidiendo que el estudiante continúe con el

proceso y dejando como única opción la eliminación del registro del anteproyecto.

Para la eliminación de un registro se utiliza la misma petición de la **Figura 23**. En este caso, la función debe eliminar tanto los archivos como el registro. Para ello, se utiliza la función "unlink()" de Node.js, donde se especifica la ubicación y el nombre del archivo mediante el atributo *archivoAnteproyecto* o *archivoProyecto*, según corresponda. Finalmente, con la función "destroy()" de Sequelize, se elimina el registro del proyecto la base de datos.

Como se especificó en el requisito RF-013, cuando un proyecto se encuentra en el módulo de "Asignación de evaluador", el director de programa puede realizar las asignaciones de el o los evaluadores dependiendo del caso. En ambos proyectos, cuando el estado del módulo es "Pendiente de evaluador", el director de programa puede usar un botón que lo lleva a una página donde puede crear las asignaciones de los evaluadores.

La creación de las asignaciones comienza con un formulario, en el cual se solicita el nombre, apellido, correo electrónico y tipo de afiliación del evaluador con la Universidad. Cuando se procesa la información, la función encargada de guardar al evaluador realiza una consulta en la base de datos para verificar si el correo especificado ya está registrado en el sistema. Si el evaluador ya existe, se extrae su *idAdministracion* y se vincula a la tabla de asignaciones correspondiente. Una vez hecho esto, el evaluador podrá visualizar la asignación en su panel de evaluación y responderla a través de una función similar a la explicada en la **Figura 23** , pero con una URL y función específica para este propósito.

Si el evaluador no está registrado en el sistema, se crea automáticamente un usuario evaluador con los datos ingresados en el formulario y una clave genérica: "8216". Además, el sistema le envía un correo electrónico mediante una función similar a la de la **Figura 20**, explicando la solicitud y proporcionando un enlace en el que podrá para aceptar o rechazar la asignación. Si el evaluador acepta la invitación desde el correo, su cuenta quedará habilitada y podrá acceder al sistema. En caso de que rechace la asignación, el usuario creado es eliminado, evitando así registros innecesarios en la base de datos.

En cuanto a las restricciones de las asignaciones, si se trata de un anteproyecto, el director sólo puede crear dos asignaciones. En cambio, si se trata de un proyecto de grado, se pueden crear hasta cuatro asignaciones, ya que son dos evaluadores los que deben aceptar. Una vez completado este módulo, los evaluadores tendrán acceso a las funciones de evaluación de los proyectos, permitiéndoles realizar su labor dentro del sistema.

6.6.4. Controlador de los administradores y los evaluadores

El último controlador pertinente al capítulo es el "administracionController". A continuación se explicará la función de revisar un proyecto para los administradores y cómo se desarrolló cada función de evaluación para cada proyecto.

En base al requisito RF-004, la función "revisar" es muy similar a la función "editar" explicada en el controlador de Anteproyectos y Proyectos de grado. El ID del proyecto se envía como parámetro en una URL gracias a la etiqueta "href", lo que permite al administrador acceder a un formulario con los principales campos del registro. Este formulario tiene como objetivo permitir al administrador descargar el archivo del proyecto para su revisión y, posteriormente, agregar comentarios sobre el registro. Una vez finalizada la revisión, el administrador puede seleccionar el estado del proyecto en función de su evaluación: "Con correcciones" o "Aceptado". Por defecto, el estado inicial es "Con correcciones", y cada vez que el administrador hace clic en el botón correspondiente, el estado cambia. El desarrollo de este botón de estado es similar al código de la **Figura 23**, con la única diferencia de que se modifican de manera dinámica el color y el título del botón mediante etiquetas y condicionales.

La evaluación cambia dependiendo del proyecto, pero la forma de acceder al formulario de evaluación es la misma en ambos casos: a través de un botón y gracias a la etiqueta "href" se pasa el ID del proyecto como parámetro de la URL para saber qué proyecto es el que se va a evaluar.

Para la evaluación del anteproyecto, el formulario cuenta con campos de selección múltiple para que el evaluador pueda elegir la opción que considere y agregar comentarios en cada pregunta. Al final, se incluye un campo para seleccionar la calificación general del anteproyecto. La evaluación del anteproyecto no se guarda en la base de datos, sólo se almacena la nota general, ya que esta determinará si el estudiante podrá acceder a las funciones previamente explicadas para editar o eliminar el registro. Toda la información de la evaluación se convierte en un archivo PDF por medio de la dependencia pdfkit[16]. Esta es una biblioteca que permite generar documentos en formato PDF con funciones, facilitando la estructuración de informes y evaluaciones.

Para la evaluación del proyecto de grado, se hace necesario diferenciar los momentos en los que el primer y segundo evaluador realizan sus respectivas calificaciones, ya que ambas deben ser combinadas en un solo documento. La solución implementada consistió en almacenar la primera evaluación en su respectiva tabla en la base de datos. De esta manera, si al realizar una consulta el *idProyecto* no existe en la tabla de evaluaciones, significa que se trata de la calificación del primer evaluador. En cambio, si el registro ya existe, entonces se trata de la calificación del segundo evaluador. Para combinar ambas en un único PDF, se promedian los valores cuanti-

tativos de ambas evaluaciones y se concatenan los comentarios, diferenciándolos con las etiquetas de “Evaluador 1” y “Evaluador 2”, cumpliendo con el requisito RF-016 .

Ambos archivos generados se almacenan en sus respectivas carpetas utilizando el módulo “path” de Node.js, asegurando una correcta organización en el sistema de archivos. En la base de datos, en lugar de guardar el archivo completo, sólo se almacena su nombre, el cual se genera de manera única mediante la función “generarId()”, garantizando así que no haya conflictos con nombres repetidos.

6.7. Desarrollo de las vistas

Finalizando con este capítulo y la explicación del desarrollo, se explicará cómo se envían las vistas desde el controlador, cómo se realizó la vista para mostrar información detallada y cómo se muestran los errores en el prototipo.

Para mostrar la vista desde el controlador se utiliza el método “res.render” de Express que procesa la plantilla y envía el HTML con las clases de TailwindCSS resultante al cliente. Este método recibe la ubicación de la vista especificada así como uno o más objetos, los cuales pueden ser consultas o datos que van a ser utilizados de manera dinámica.

Por ejemplo, para mostrar todos los anteproyectos en el sistema se hace la consulta en la base de datos, la cual se guarda en una variable, que hace parte de un parámetro del método de Express. Para mostrar los proyectos de manera secuencial se itera este objeto mediante un ciclo. De igual forma, se muestran los errores de las validaciones, como lo pide el requisito RF-026, por medio de la función “validationResult()” de la dependencia de express-validator. Esto permite guardar todas las validaciones que han marcado error en el prototipo, transformándolas en un array, que luego pasa como un objeto en el “res.render”. Ya en la vista se verifica que el arreglo de errores no sea vacío; en ese caso, se itera con un ciclo y se va mostrando de manera secuencial cada error con color rojo como se ve en la **Figura 24**.

Para recordar un valor en un campo de un formulario y evitar que el usuario deba volver a escribir los campos que ya había llenado, se utiliza una etiqueta llamada “value”. Posteriormente, se hace un condicional con dicha etiqueta y “req.body” que se genera cuando se escribe un campo en el formulario. Si el atributo de “req,body” tiene algún valor, la etiqueta lo muestra, en caso de que no, muestra el espacio vacío.

Las vistas finales en el prototipo funcional se realizaron con base en los mock-ups creados en la sección de diseño. Dichos mock-ups abarcan las interfaces para el inicio de sesión y registro, así como las vistas para un usuario administrador, el seguimiento del progreso para estudiantes y la visualización detallada de la información. Estas

vistas finales se encuentran documentadas en el capítulo de anexos del presente documento.

Figura 24: Diseño de como se muestran los errores

7. Pruebas del sistema de gestión

En este capítulo se realizarán pruebas funcionales para validar el correcto funcionamiento del prototipo desarrollado. Dado que este trabajo de grado consiste en un prototipo funcional, las pruebas estarán enfocadas en verificar que cumple con los requisitos establecidos y que sus funcionalidades operan correctamente.

A continuación se especifica el nombre de la prueba, su propósito y los requisitos evaluados en cada prueba realizada, separadas por módulo:

7.1. Pruebas de caja negra

Las pruebas de caja negra son un tipo de prueba de software que se utilizan para evaluar el comportamiento de un sistema sin conocer su estructura interna. En el desarrollo del sistema, se emplearon estas pruebas para confirmar el correcto funcionamiento de las validaciones en los formularios, ya que estos representan el mecanismo principal de almacenamiento de información en el prototipo. Como ejemplo, en la **Figura 25** se muestra la validación de los campos del formulario de registro de un estudiante. En esta prueba, se verifica cada posible valor que un campo puede o no aceptar mediante representantes como “ABC”, que indica que el campo sólo debe contener letras, y “123”, que señala que sólo se permiten números. Cada representante está acompañado de un ejemplo práctico y un indicador que muestra si el dato es válido o no. En caso de ser inválido, se incluye una sección de respuesta que ejemplifica el tipo de mensaje de error que debería generarse.

Variable	Equivalencia	Tipo	Representante	Respuesta de error
Nombres	ABC	V	Juan	
	123	NV	1002324	El nombre deben ser letras
	NULL	NV		El nombre no puede ser vacío
Apellidos	ABC	V	Amaya	
	123	NV	1002324	El apellido deben ser letras
	NULL	NV		El apellido no puede ser vacío
Correo	ABC@DOMINIO.COM	V	amaya@amaya.com	
	ABC@DOMINIO	NV	amaya@amaya	Correo invalido
	NULL	NV		El correo no puede ser vacío
Clave	ABC+123	V	Clave123	
	ABC	NV	Clave	La clave debe tener letras y números
	NULL	NV		La clave no puede ser vacía
Código del estudiante	123	V	8949439	
	ABC	NV	Codigo	El código deben ser números
	NULL	NV		El código no puede ser vacío
Tipo de identificación	ABC	V	Cédula de ciudadania	
	NULL	NV		El tipo de identificación no puede ser vacío
Número de identifiación	123	V	100234	
	ABC	NV	Numero	El número de identificación deben ser números
	NULL	NV		El número de identificación no puede ser vacío
Paso anteproyecto	TRUE	V	Si	
	FALSE	V	No	
	NULL	NV		Este prerrequisito no puede ser vacío

Figura 25: Ejemplo prueba de caja negra - registro estudiante

Se eligió el formulario de registro de un estudiante como ejemplo ya que es el que contiene más campos variables en comparación con otros formularios del sistema. En su mayoría, los formularios cuentan con campos de selección múltiple, cuya principal validación consiste en garantizar que no queden vacíos, al igual que en el formulario de registro. No se incluyeron pruebas de validación para los formularios de evaluación, ya que son las mismas validaciones repetidas múltiples veces. Sin embargo, en ambos formularios de evaluación se probó que los comentarios no superaran los 300 caracteres para evitar problemas de visualización en los archivos generados. Para la evaluación de proyectos de grado, se verificó que la nota numérica estuviera en el rango de 1.0 a 5.0, permitiendo tanto la coma (,) como el punto (.) como separadores decimales. Además, para mostrar la nota final con un solo decimal, se utilizó la función “toFixed()” de JavaScript, que redondea el valor siguiendo las reglas estándar de aproximación.

7.2. Casos de prueba funcionales

Son escenarios diseñados para validar una funcionalidad específica del sistema, comprobando que el comportamiento obtenido coincida con el esperado. Para el prototipo, los casos de prueba se basarán en las funciones de cada módulo del sistema, establecidas en el diagrama de casos de uso mostrado en la **Figura 4**, para garantizar el correcto desempeño de las funciones más importantes. Estas pruebas se ejecutaron de forma manual, simulando las interacciones de los usuarios con el sistema.

7.2.1. Módulo Ingreso

- **Nombre de la prueba:** Registro exitoso de un estudiante.
- **Propósito de la prueba:** Verificar que un estudiante puede registrarse si cumple con los prerrequisitos y el sistema envía del correo para validar el registro.
- **Requisito:** RF-006, RF-018

- **Nombre de la prueba:** Registro fallido de un estudiante.
- **Propósito de la prueba:** Verificar que un estudiante no pueda registrarse si no cumple con el prerrequisito del curso de anteproyecto.
- **Requisito:** RF-006, RF-018, RF-026

- **Nombre de la prueba:** Inicio de sesión exitoso.
- **Propósito de la prueba:** Verificar que un estudiante puede iniciar sesión si ha confirmado la cuenta.
- **Requisito:** RF-017

- **Nombre de la prueba:** Inicio de sesión fallido.
- **Propósito de la prueba:** Verificar que un estudiante no puede iniciar sesión debido a que no ha confirmado la cuenta.
- **Requisito:** RF-017, RF-026

- **Nombre de la prueba:** Inicio de sesión con un JWT inválido.
- **Propósito de la prueba:** Impedir el ingreso del sistema a usuarios que intentan crear o cambiar su JSON Web Token.
- **Requisito:** RF-021

- **Nombre de la prueba:** Cambio de clave exitoso
- **Propósito de la prueba:** Verificar que un usuario puede realizar el cambio de su clave.
- **Requisito:** RF-017

7.2.2. Módulo Registro anteproyecto/proyecto de grado

- **Nombre de la prueba:** Registro exitoso de un anteproyecto.
- **Propósito de la prueba:** Verificar que un estudiante puede registrar su anteproyecto y se envían los correos correspondientes.
- **Requisito:** RF-007, RF-008, RF-022, RF-025

- **Nombre de la prueba:** Registro fallido de un anteproyecto.
- **Propósito de la prueba:** Verificar que un estudiante no puede registrar su anteproyecto si el director de proyecto no está registrado.
- **Requisito:** RF-007, RF-008, RF-026

- **Nombre de la prueba:** Revisión exitosa de un anteproyecto.
- **Propósito de la prueba:** Verificar que el administrador puede realizar la revisión de un anteproyecto.
- **Requisito:** RF-004

- **Nombre de la prueba:** Edición exitosa de un anteproyecto.
- **Propósito de la prueba:** Verificar que un estudiante puede realizar la edición del registro de su anteproyecto.
- **Requisito:** RF-009, RF-010

- **Nombre de la prueba:** Edición de un anteproyecto con cambio de estado por parte del administrador.
- **Propósito de la prueba:** Verificar que no se guarda la edición de un estudiante cuando un administrador cambia el estado de registro a: “Aceptado”.
- **Requisito:** RF-004, RF-009, RF-010

- **Nombre de la prueba:** Eliminación exitosa del registro de un anteproyecto por parte de un administrador.
- **Propocito de la prueba:** Verificar que un administrador puede eliminar el registro de un anteproyecto.
- **Requisito:** RF-005

- **Nombre de la prueba:** Eliminación del registro de un Anteproyecto que no existe por parte de un administrador.

- **Propósito de la prueba:** Verificar que un administrador no pueda eliminar registros de Anteproyectos que no existen.
- **Requisito:** RF-005, RF-026

7.2.3. Módulo Asignación Evaluador

- **Nombre de la prueba:** Asignación a un evaluador registrado por parte de un director de programa.
- **Propósito de la prueba:** Verificar que un director puede registrar la asignación de un evaluador que existe como usuario válido en el sistema.
- **Requisito:** RF-013

- **Nombre de la prueba:** Asignación a un evaluador sin registro en el sistema por parte de un director de programa.
- **Propósito de la prueba:** Verificar que un director puede registrar la asignación de un evaluador que no existe en el sistema.
- **Requisito:** RF-013

- **Nombre de la prueba:** Responder asignación de evaluación.
- **Propósito de la prueba:** Verificar que un evaluador puede aceptar la asignación realizada por el director de programa.
- **Requisito:** RF-013, RF-015

- **Nombre de la prueba:** Eliminar una asignación que ya ha sido aceptada.
- **Propósito de la prueba:** Verificar que un director no pueda eliminar una asignación si el evaluador ya la aceptó.
- **Requisito:** RF-013

7.2.4. Módulo Evaluación anteproyecto/proyecto de grado

- **Nombre de la prueba:** Evaluación exitosa de un anteproyecto.
- **Propósito de la prueba:** Verificar que un evaluador puede evaluar un anteproyecto y se genera el archivo correspondiente.
- **Requisito:** RF-016

- **Nombre de la prueba:** Evaluación exitosa de un proyecto de grado.
- **Propósito de la prueba:** Verificar que un evaluador puede evaluar un proyecto de grado y se genera el archivo correspondiente.
- **Requisito:** RF-016

Cada prueba se llevó a cabo siguiendo un proceso metódico que inició con la asignación de un nombre representativo y un programador responsable de su ejecución. Durante el desarrollo, se definieron los pasos a seguir, los criterios de éxito y se registraron los datos de prueba junto con los resultados esperados. Al finalizar cada prueba, se emitió un veredicto (por ejemplo, "Pass" o "Fail"), acompañado de comentarios relevantes para justificar el resultado o sugerir posibles mejoras. En el siguiente enlace se encuentra el desarrollo completo de todas las pruebas mencionadas: Casos de prueba funcionales.

En total, se llevaron a cabo 23 pruebas funcionales enfocadas en validar los requisitos de las principales funciones del sistema de gestión. Gracias a la ejecución de estas pruebas, se logró verificar el correcto funcionamiento de cada proceso clave dentro del sistema, asegurando que todas las etapas de la gestión de un trabajo de grado se completaran de manera exitosa. Lo más importante de estas pruebas es que se logró completar todo el proceso de gestión que conlleva un trabajo de grado de principio a fin.

7.3. Prueba Final con el Usuario

Como prueba final, el prototipo se presentó a la asistente de Maestría en Ingeniería, quien representa el perfil del usuario final para el cual fue desarrollado el prototipo. El propósito de esta prueba fue simular el proceso completo de gestión de un trabajo de grado, permitiendo que ella experimentara cada una de sus etapas y brindara su retroalimentación. En particular, se buscó conocer su opinión sobre los aspectos positivos del prototipo, las áreas de mejora y su percepción general sobre la utilidad del sistema para la gestión de trabajos de grado en el área de posgrado de la Facultad de Ingeniería de la Universidad Javeriana Cali.

Para recopilar esta información, se diseñó un formulario en Microsoft Forms con preguntas relacionadas a lo que se explicó anteriormente, la facilidad de navegación y la efectividad del sistema en el cumplimiento de los objetivos. Las respuestas obtenidas están documentadas en la imagen de la **Figura 26**.

1. Nombre completo *

Linda Carolina Suárez Morelos

2. Correo institucional *

linda.suarez@javerianacali.edu.co

3. De 1 a 10, considera que el prototipo cumple con su función? (Siendo 1, no cumple, y 10, cumple completamente) *

1	2	3	4	5	6	7	8	9	10

4. De 1 a 10, considera que el prototipo es útil para el área de Posgrados de Ingeniería de la Pontificia Universidad Javeriana Cali? (Siendo 1, no es útil, y 10, siendo muy útil) *

1	2	3	4	5	6	7	8	9	10

5. ¿Qué considera que podría mejorar o añadírsele al prototipo? *

Añadir notificaciones, corregir las fechas, añadir una columna para ver el estado del proceso de evaluación (administrador), el flujo de aprobación de los jurados para que no se limite el diligenciamiento de la evaluación, añadir el historial de envíos (primera, segunda o tercera entrega)

6. ¿Qué características le gustaron del prototipo? (Diseño, facilidad de navegación, funciones, etc.) *

Diseño, simplicidad, fácil navegación, transiciones del flujo. En general es un excelente trabajo, muchas gracias.

Figura 26: Opinión del usuario final

8. Conclusiones

El desarrollo de este prototipo funcional en Node.js y MySQL permitió la gestión del proceso de trabajo de grado en las etapas de registro, revisión, asignación de evaluadores y evaluación. Su implementación apoya el registro, seguimiento y visualización entre los actores involucrados, especialmente el personal de apoyo académico, asegurando que cada etapa del proceso se realice correctamente. En este sentido, el principal objetivo del proyecto ha sido cumplido, proporcionando una herramienta que optimiza la administración del trabajo de grado.

Todos los requisitos especificados fueron abordados y desarrollados con éxito, garantizando que cada funcionalidad implementada tiene un propósito en el proceso y cumple con ello. A lo largo de los capítulos de este documento se abordó cada etapa del desarrollo de software que concluye en el prototipo desarrollado. Aunque el sistema aún necesita mejoras y ajustes para su implementación como una solución definitiva dentro de la Universidad Javeriana Cali, el prototipo desarrollado cumple plenamente con el propósito de este trabajo de grado, sirviendo como una base sólida para futuras iteraciones o mejoras.

El proceso para realizar el presente trabajo de grado no fue particularmente difícil, pero sí demandó tiempo, principalmente debido a la necesidad de buscar información, adaptarla a los requerimientos específicos identificados y aplicarla correctamente en cada uno de los objetivos planteados. A lo largo de este camino, todo lo aprendido durante la formación en la carrera sirvió como base para abordar cada fase del desarrollo de software dentro del sistema de gestión, consolidando los conocimientos adquiridos y aplicándolos en una solución estructurada para una problemática real en el ámbito institucional. Si se tuviera que repetir este proceso, se seguiría el mismo enfoque, pero con el apoyo y colaboración de un compañero o un equipo de trabajo, ya que el desarrollo de software es una tarea colectiva en la que distribuir responsabilidades no solo mejora la calidad del proyecto final, sino que también optimiza los tiempos de entrega.

En definitiva, este trabajo de grado no solo permitió la creación de un prototipo funcional para la gestión de trabajos de grado, sino que también representó una valiosa experiencia formativa. Se espera que futuros trabajos puedan llevar este prototipo a un sistema funcional que sirva como solución definitiva a la gestión de trabajos de grado en posgrado para la Facultad de Ingeniería y Ciencias de la Universidad Javeriana Cali.

9. Trabajo futuro

Esta sección pretende detallar el trabajo al que se aspira seguir con el prototipo funcional que se dio como resultado de este proyecto de grado. Al ser un prototipo, se entiende que no es la versión final para el sistema; tal versión deberá tener un sistema de notificaciones propio, que no dependa únicamente del envío de correos electrónicos. De esta manera, el usuario podrá saber si ha sucedido alguna actualización respecto a su proceso.

Otra mejora importante a considerar para futuras versiones del sistema es la implementación de un doble factor de autenticación, lo que fortalecería la seguridad en el inicio de sesión. Este mecanismo permitiría que los usuarios validen su identidad a través de un segundo método, como el uso de una aplicación de autenticación, como se hace en otras aplicaciones de la Universidad (Google Authenticator). Además, se debería incorporar un proceso de verificación adicional en el registro de nuevos administradores, asegurando que sólo personas autorizadas puedan obtener este rol.

Por último, en próximos trabajos se debería subir el sistema a un servidor, en lo posible que sea el de la Universidad Javeriana Cali, de manera que se logre un acople y migración para usar los datos de la Universidad y se pueda usar como una solución al apoyo de gestión en el sector de posgrados de la Facultad de Ingeniería y Ciencias.

10. Bibliografía

Referencias

[1] Ian Sommerville, *Software Engineering [Ninth Edition]*, 2011, pp. 24 - 73. Disponible en: https://engineering.futureuniversity.com/BOOKS%20FOR%20IT/Software-Engineering-9th-Edition-by-Ian-Sommerville.pdf [visitado 18-02-2025]

[2] Aditya Singh, Piyush Chawla, Karan Singh, Ashutosh Kumar Singh, *Formulating an MVC Framework for Web Development in JAVA*, 2018, pp. 1. Disponible en: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8553746 [visitado 18-02-2025]

[3] OpenJS, *Introduction to Node.js*. Disponible en: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs [visitado 18-02-2025]

[4] Oracle Corporation, *What is MySQL?*. Disponible en: https://dev.mysql.com/doc/refman/8.4/en/what-is-mysql.html [visitado 18-02-2025]

[5] Oracle Corporation, *Sequelize*. Disponible en: https://sequelize.org/ [visitado 18-02-2025]

[6] Juan Pablo Roche y Julián Suárez, *Análisis, diseño, e implementación de un software para la administración de los proyectos de grado en el programa de ingeniería de sistemas, aplicando una metodología ágil*, 2009. Disponible en: https://repositorio.utp.edu.co/items/e1160723-3252-4a7c-96d8-7839d6075a32 [visitado 18-02-2025]

[7] J. P. Rodríguez, *Software para el seguimiento, la gestión y el control de proyectos de grado en el departamento de electrónica (SSGPG)*, 2014. Disponible en: http://hdl.handle.net/10554/16430 [visitado 18-02-2025]

[8] Murillo Galvis, Diego Dayiber, Mosquera Díaz, Javier, Barragán Bohórquez, Hugo Leonardo, *Sistema de información Web que asiste el proceso de radicación y seguimiento de proyectos de grado de la Especialización en Ingeniería de Software de la Universidad Distrital Francisco José de Caldas*, 2015. Disponible en: http://repository.udistrital.edu.co/handle/11349/8270 [visitado 18-02-2025]

[9] Atlassian, *Jira Software*. Disponible en: https://www.atlassian.com/es/software/jira/features [visitado 18-02-2025]

[10] Basecamp, *37signals*. Disponible en: `https://basecamp.com/` [visitado 18-02-2025]

[11] Express.js, *express-validator*. Disponible en: `https://www.npmjs.com/package/express-validator` [visitado 18-02-2025]

[12] GitHub, *node.bcrypt.js*. Disponible en: `https://www.npmjs.com/package/bcrypt` [visitado 18-02-2025]

[13] EmailEngine, *NODEMAILER*. Disponible en: `https://www.nodemailer.com/` [visitado 18-02-2025]

[14] auth0, *node.bcrypt.js*. Disponible en: `https://jwt.io/` [visitado 18-02-2025]

[15] GitHub, *multer*. Disponible en:`https://www.npmjs.com/package/multer` [visitado 18-02-2025]

[16] Devon Govett, *PDFkit*. Disponible en:`https://pdfkit.org/` [visitado 18-02-2025]

11. Anexos

Figura 27: Diseño final del Login

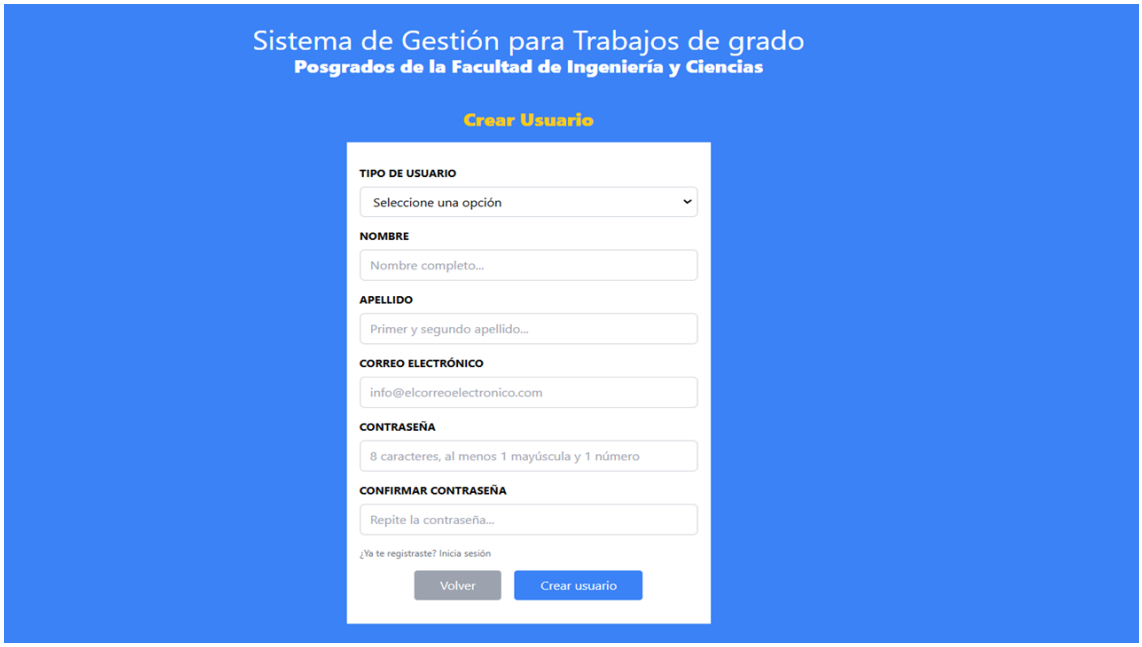


Figura 28: Diseño final del registro

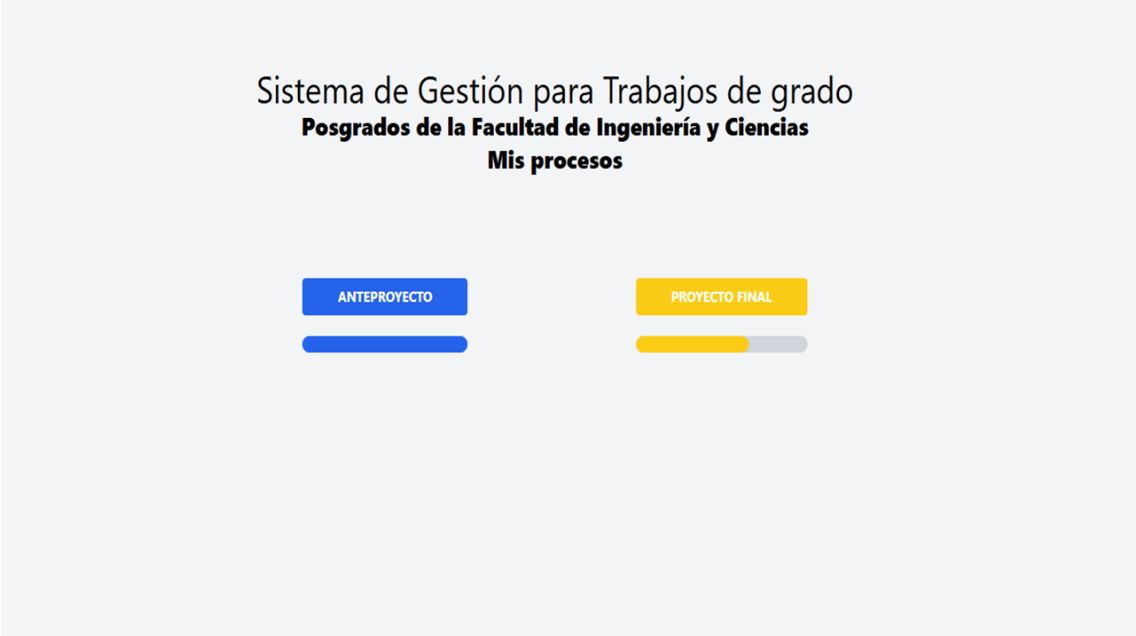


Figura 29: Diseño final del Progreso de un estudiante

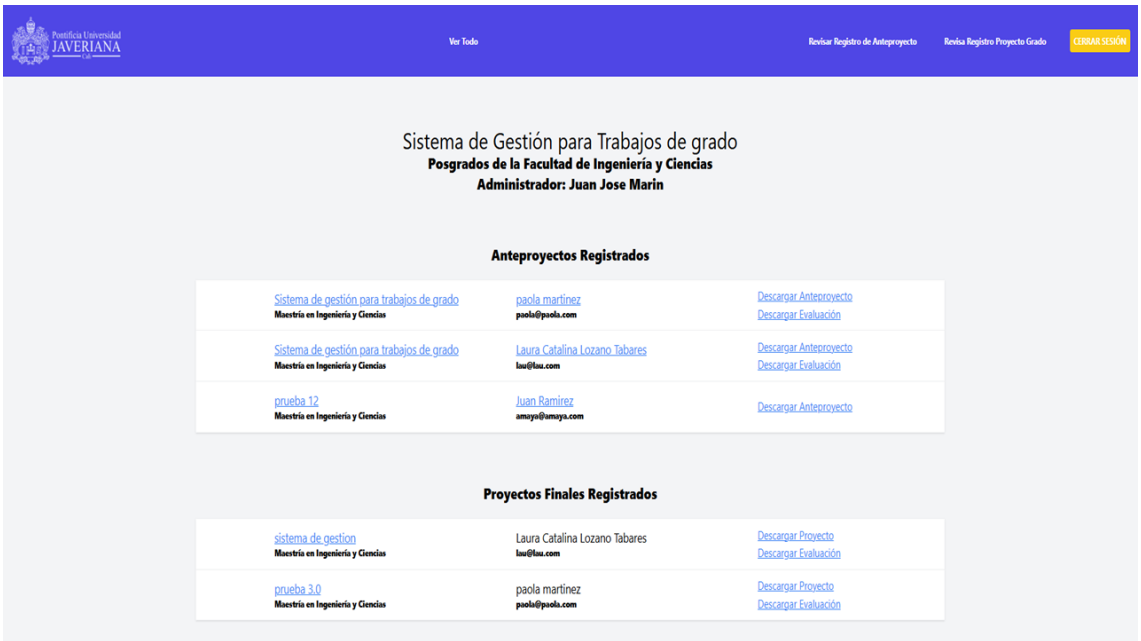


Figura 30: Diseño final de la vista principal para un administrador

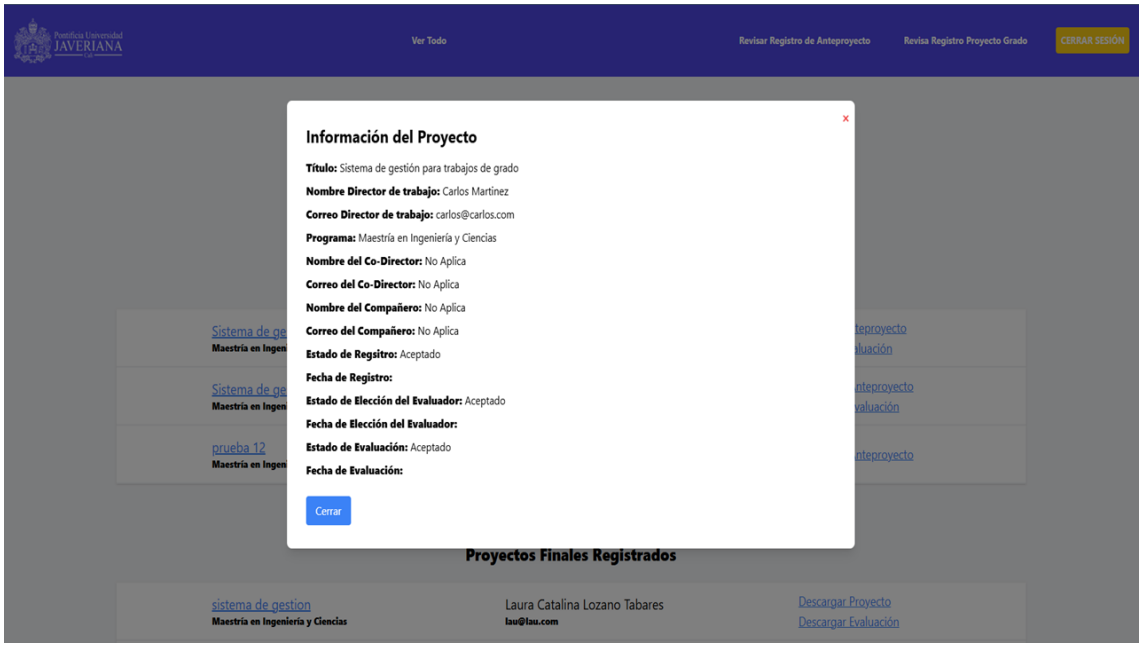


Figura 31: Diseño final para ver información detallada