FACULTAD DE INGENIERÍA

UNIVERSIDAD DE TARAPACÁ

Universidad del Estado

Departamento de Ingeniería en Computación e Informática

Primer avance: Sistema de gestión de documentos para la Facultad de Odontología de la Universidad de Chile

Autor: Javier Mamani
Curso: Proyecto IV
Profesor: Diego Aracena
Empresa: Facultad de Odontología de la Universidad de Chile

ARICA, 12 de diciembre, 2022

Índice

1. Introducción

La administración y gestión de documentos en papel y digitales es fundamental para cualquier empresa que quiera mantener mayor control y orden sobre los archivos que se van transfiriendo de un lugar a otro en la empresa. Esto, además, es útil a la hora de agilizar procesos que se hacen tediosos hacerlos de manera tradicional, como lo son la administración de archivos manualmente.

Dada esta situación anterior, la Facultad de Odontología de la Universidad de Chile, plantea como proyecto para la asignatura correspondiente a Proyecto IV, el desarrollo de un sistema capaz de solucionar la problemática anteriormente mencionada. La finalidad del sistema entonces es la administración y gestión de recursos digitales para la Facultad de Odontología de la Universidad de Chile.

En el presente documento se expondrá el desarrollo de la propuesta de proyecto realizado por el cliente perteneciente a la Facultad de Odontología de la Universidad de Chile para la asignatura correspondiente a Proyecto IV, donde inicialmente se describe la empresa asociada al proyecto, luego se abordará el proyecto desde el punto de vista de la ingeniería de software, para posteriormente describir el sistema solución de manera sistémica, describiendo los casos de uso y descripciones respectivas para el desarrollo del modelo respectivo, como también los distintos diagramas que corresponden al diseño del tipo de sistema, y finalmente se implementará el sistema, realizando pruebas sucesivas, analizando resultados, y la entrega del producto final al cliente.

2. Objetivos

2.1 Objetivo general

Desarrollar un Sistema de gestión de documentos para la Facultad de Odontología de la Universidad de Chile, para eliminar en un porcentaje significativo la circulación de papel en las unidades

2.2 Objetivos específicos

- Estudiar el flujo de los documentos en la Facultad y sus archivados.
- Establecer los requerimientos funcionales y no funcionales del Sistema.
- Desarrollar el sistema y las herramientas necesarias para su implementación.
- Realizar las pruebas de funcionamiento y análisis de resultados.

3. Marco teórico

3.1 Patrón de diseño Model View Controller

MVC (Modelo-Vista-Controlador) es un patrón en el diseño de software comúnmente utilizado para implementar interfaces de usuario, datos y lógica de control. Enfatiza una separación entre la lógica de negocios y su visualización. Esta "separación de preocupaciones" proporciona una mejor división del trabajo y una mejora de mantenimiento.

Las tres partes del patrón de diseño de software MVC se pueden describir de la siguiente manera:

1. Modelo: Maneja datos y lógica de negocios.
2. Vista: Se encarga del diseño y presentación.
3. Controlador: Enruta comandos a los modelos y vistas.

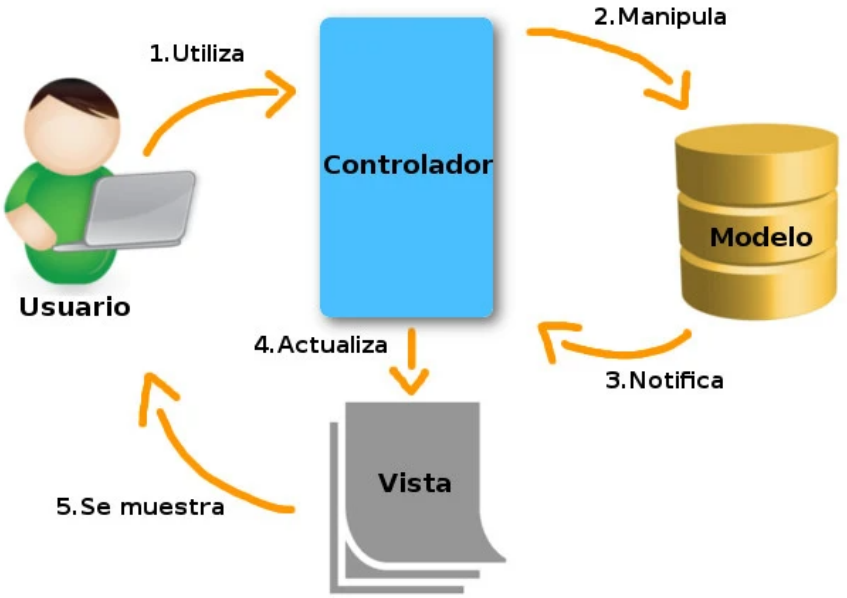

Figura 1. Patrón MVC

3.2 Patrón de diseño Model View ViewModel

El patrón Model View ViewModel (MVVM) ayuda a separar limpiamente la lógica de negocios y presentación de una aplicación de su interfaz de usuario (UI).
Mantener una separación limpia entre la lógica de la aplicación y la interfaz de usuario ayuda a abordar numerosos problemas de desarrollo y puede facilitar la prueba, el mantenimiento y la evolución de una aplicación. También puede mejorar considerablemente las oportunidades de reutilización del código y permite a los desarrolladores y diseñadores de interfaz de usuario colaborar más fácilmente al desarrollar sus respectivas partes de una aplicación.

Hay tres componentes principales en el patrón MVVM: el modelo, la vista y el modelo de vista. Cada uno sirve para un propósito distinto y se describirán a continuación:

- Vista: Es responsable de definir la estructura, el diseño y la apariencia de lo que ve el usuario en la pantalla.
- Modelo de vista: El modelo de vista implementa propiedades y comandos a los que la vista puede enlazar datos y notifica a la vista los cambios de estado a través de eventos de notificación de cambios.
- Modelo: Las clases de modelo son clases no visuales que encapsulan los datos de la aplicación. Por lo tanto, el modelo se puede considerar como que representa el modelo de dominio de la aplicación, que normalmente incluye un modelo de datos junto con la lógica de validación y negocios.

En la figura 2 se muestran las relaciones entre los tres componentes.

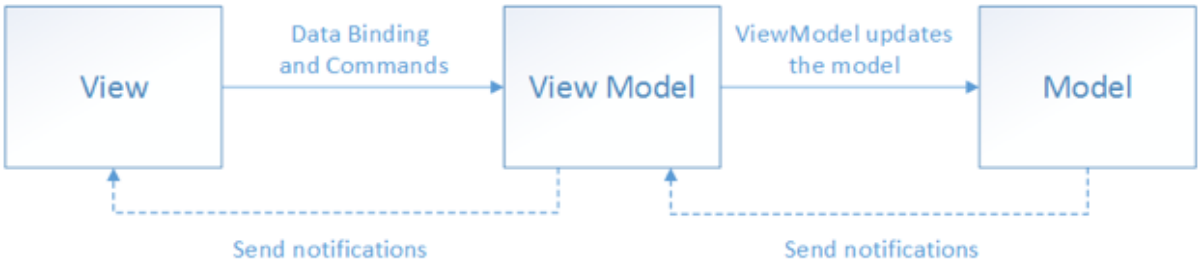

Figura 2. Patrón MVVM

3.3 Patrón de diseño publicador suscriptor

Este patrón yace en la capacidad de los suscriptores en conocer el estado del sujeto registrándose para ello a un publicador para recibir notificaciones cuando algún evento en este último suceda. Un objeto llame a los métodos de otro objeto directamente, ellos se suscriben a un evento en particular de ese otro objeto, recibiendo los mensajes de notificación cuando dicho evento ocurra.
Una aplicación publicador-suscriptor es aquella en la que un servidor da una serie de servicios, a los usuarios, para gestionar, almacenar y presentar todo tipo de información, y otros usuarios se suscriben a ella para ser avisados de nuevas actualizaciones o informaciones.

El sistema publicador-suscriptor es un paradigma de mensajes asíncronos donde los que envían (Publicador) mensajes no están programados para enviar sus mensajes a receptores específicos(Suscriptor), sino que se envían a algún tipo de servidor. Los mensajes publicados se caracterizan por clases, sin tener constancia de los suscriptores que pueda haber. Los suscriptores expresan interés en una o más clases, y solo reciben mensajes de ese mismo interés, sin tener constancia de qué publicadores hay. Esta relación independiente entre publicadores y suscriptores puede permitir una mayor escalabilidad.

3.4 El modelamiento y diseño de software

El modelado es el análisis y diseño de aplicaciones de software, mediante la creación de un conjunto de modelos que nos permiten especificar aspectos del sistema como los requisitos, la estructura y el comportamiento, permiten comprender y razonar de mejor forma el sistema, podemos hablar por ejemplo del modelo estático el cual aborda la visión estructural estática de un problema que no varía en el tiempo, un modelo estático describe la estructura estática del sistema que es modelado definiendo las clases en el sistema, los atributos de las clases y las relaciones entre las mismas, por otro lado tenemos al modelamiento dinámico que proporciona una vista de un sistema en el que el control y la secuenciación son considerados, dentro de un objeto o fuera de los mismos mediante el análisis de interacciones de objetos.

4. Descripción de la empresa

La Facultad de Odontología de la Universidad de Chile es el organismo académico y de gobierno, encargado de llevar a cabo labores específicas de Docencia, Investigación y extensión en las disciplinas odontológicas.

La Facultad es un referente en la educación odontológica en nuestro país, ya que sus egresados representan la mayoría de los profesionales nacionales y muchos de ellos destacan en el liderazgo del ámbito odontológico chileno, participando en instancias ministeriales, institucionales, públicas y privadas, y como decanos o directores de nuevas Facultades y Escuelas de Odontología de Universidades.

4.1 Propósitos

Los propósitos de la Facultad de Odontología corresponden a una elaboración y adecuación de los objetivos estratégicos de la Universidad para nuestra unidad académica. Estos son:

- Consolidar la calidad y pertinencia de los programas de Pregrado.
- Consolidar el liderazgo nacional y avanzar en el reconocimiento internacional del Postgrado.
- Fortalecer y consolidar la investigación científica y la creación
- Potenciar la Extensión como medio de vinculación con la comunidad.
- Mejorar y modernizar la administración universitaria y la situación patrimonial.
- Consolidar la vinculación externa y avanzar en la internacionalización de la institución.

4.2 Misión compartida

"La Facultad de Odontología de la Universidad de Chile es una comunidad universitaria que convoca a jóvenes talentos y profesionales encargándose de su formación integral, en Pregrado, Postítulo y Postgrado, con alto dominio de competencias científicas, técnicas y éticas, y del desarrollo del conocimiento científico mediante la investigación, docencia y extensión, desarrollándolos como recurso humano orientado a integrarse y participar activamente en las políticas de salud, además de satisfacer las necesidades de atención de salud de nuestro país."

4.3 Visión compartida

"La Facultad de Odontología de la Universidad de Chile será la institución de educación superior líder en la formación de cirujanos dentistas y contribuirá con la formación de profesionales de excelencia dentro del área de salud del país, desarrollando acciones de salud e impartiendo programas de Pregrado, Postítulo y Postgrado con excelencia académica, tecnología adecuada al mundo globalizado,

basada en los valores del compromiso, honestidad, respeto y solidaridad. Mantendrá sus altos estándares en investigación, transformándose en un referente en esta área, estableciendo políticas de desarrollo que la posicionen en el contexto nacional e internacional como una de las instituciones a la vanguardia en temas de salud, con alto compromiso social y líder en el área odontológica."

4.4 Organigrama

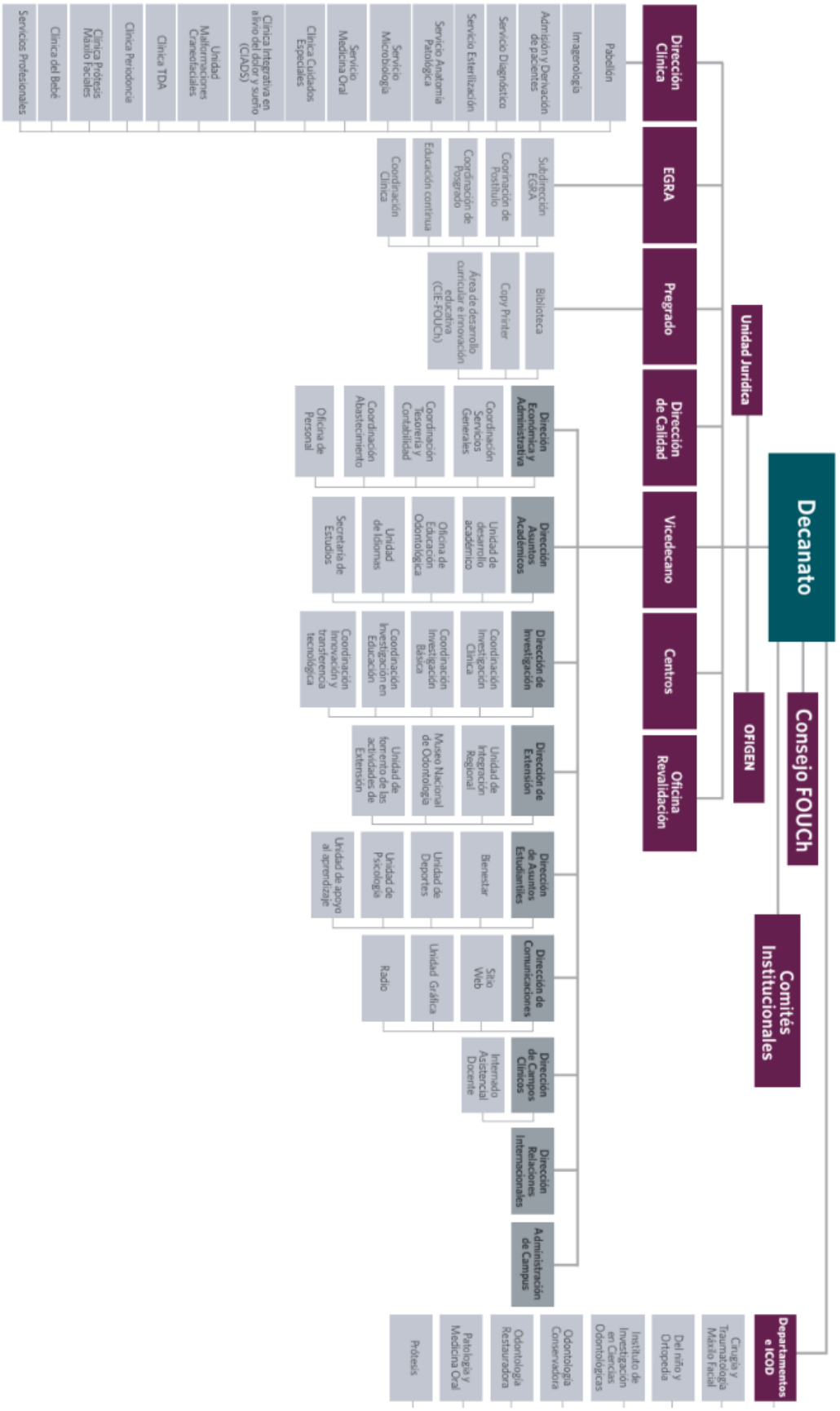

Figura 3. Organigrama empresa

5. Desarrollo

5.1 Descripción del problema

La Facultad de Odontología de la Universidad de Chile maneja grandes cantidades de documentos en papel, sin un tratamiento uniforme e integrado de éstos, lo que implica que no hay claridad de la forma en que se archivan ni quiénes son los encargados de mantener los registros. La gestión de documentos digital es optativa para las distintas unidades que componen la Facultad (Escuelas, Departamentos y Direcciones, y sus respectivas áreas), sin que haya una política de cómo almacenar y gestionar los documentos digitales. Lo anterior, debido a que la gestión de los documentos se hace de manera tradicional, no hay un sistema de gestión documental informático. Esto produce poca eficiencia en la comunicación entre las unidades, desconocimiento de dónde se encuentra la información y falta de acceso a ella por parte de los distintos miembros de la comunidad, así como gasto de excesivo tiempo y falta de organización de los documentos.

5.2 Descripción de la solución

El objetivo del presente proyecto es implementar un sistema web de gestión de documentos con los que se trabaja en las distintas unidades de la Facultad de Odontología de la Universidad de Chile (Escuelas, Direcciones y Departamentos), con el objetivo de apoyar la política de cero papel del Estado y de la actual autoridad de la Facultad, mejorando así los tiempos en la administración de estos documentos, como también teniendo un repositorio organizado con los documentos. El sistema permitirá trabajar a los usuarios de distinta manera dependiendo del tipo de usuario que sea (Decana, Directores, Secretarias, Académicos y funcionarios).

5.3 Descripción del proyecto y alcance

A continuación, expondremos las características, requisitos, alcance del ambiente y los sistemas respectivos al proyecto planteado para la asignatura Proyecto IV.

5.3.1 Requisitos de alto nivel

A continuación, se muestran los requisitos de alto nivel asociados al proyecto de desarrollo “Sistema de gestión de documentos para la Facultad de Odontología de la Universidad de Chile”. Cada uno de estos requerimientos se encuentra junto a su respectivo código de identificación.

Tabla 1. Requisitos de alto nivel

Código	Descripción

RA-1	Proveer de un repositorio de recursos para las distintas unidades de la Facultad de Odontología de la Universidad de Chile.
RA-2	Suministrar información referente a las distintas unidades de la Facultad de Odontología de la Universidad de Chile.
RA-3	Permitir a los usuarios la gestión de información de los datos de cada unidad de la Facultad de Odontología de la Universidad de Chile.
RA-4	Proveer al usuario herramientas de gestión de los datos generados.

5.3.2 Requisitos funcionales

A continuación, en la Tabla se presentan los requerimientos funcionales del sistema “Sistema de gestión de documentos para la Facultad de Odontología de la Universidad de Chile”. Cada uno de estos requerimientos se encuentra junto a su respectivo código de identificación y nivel de prioridad.

Tabla 2. Requisitos funcionales

Código	Descripción	Prioridad
RF-1	El sistema debe soportar cinco tipos de usuarios: Decana (Administrador), Directores, Secretarias, Académicos y Funcionarios.	Alta
RF-2	El sistema debe de verificar la sesión del usuario.	Alta
RF-3	El administrador podrá crear, editar y eliminar usuarios, asignándoles el tipo de usuario de este.	Media
RF-4	El sistema debe de asignar permisos a los diferentes tipos de usuarios que soporta, estos permisos determinarán las acciones que puedan realizar los usuarios en cada unidad del sistema.	Alta
RF-5	El sistema debe permitir a los usuarios visualizar sus datos correspondientes a su cuenta.	Media
RF-6	El administrador podrá crear, validar, editar, enviar, visualizar, crear secciones y eliminar archivos correspondientes a todas las unidades del sistema.	Alta
RF-7	Los usuarios podrán crear, enviar, editar, enviar, visualizar, validar, crear secciones o eliminar archivos de unidades permitidas. Esto dependiendo de los permisos del tipo de usuario.	Alta
RF-8	Los usuarios permitidos podrán crear secciones asignando el nombre de este y el grupo de personas, esto en cada unidad del sistema.	Baja
RF-9	Los usuarios podrán seleccionar los usuarios a notificar cuando se cree, edite, elimine, envíe o valide un archivo.	Baja

RF-10	El sistema debe notificar a los usuarios cuando se realicen cambios respecto a su unidad y sección correspondiente. Además, el sistema debe notificar los cambios en los archivos de cada sección a los usuarios seleccionados.	Baja
RF-11	El sistema debe guardar registros de los archivos editados.	Baja

5.3.3 Requisitos no funcionales

A continuación, en la Tabla se presentan los requerimientos no funcionales del sistema “Sistema de gestión de documentos para la Facultad de Odontología de la Universidad de Chile”. Cada uno de estos requerimientos se encuentra junto a su respectivo código de identificación.

Tabla 3. Requisitos no funcionales

Código	Descripción
RNF-1	El sistema debe ser amigable para el usuario.
RNF-2	El sistema debe cumplir con los estándares de seguridad requeridos para mantener la integridad y seguridad de los datos.
RNF-3	El sistema debe tener un enfoque web.
RNF-4	El sistema debe ser escalable y con la posibilidad de comunicarse con otros sistemas externos.
RNF-5	El sistema debe ser capaz de comunicarse con bases de datos.

5.3.4 Actividades del proyecto

Para llevar a cabo una definición del alcance del proyecto es necesario definir las actividades y/o trabajos necesarios para satisfacer los requerimientos, en la imagen 1 se muestran las actividades y las semanas respectivas sobre las que se proyecta su realización.

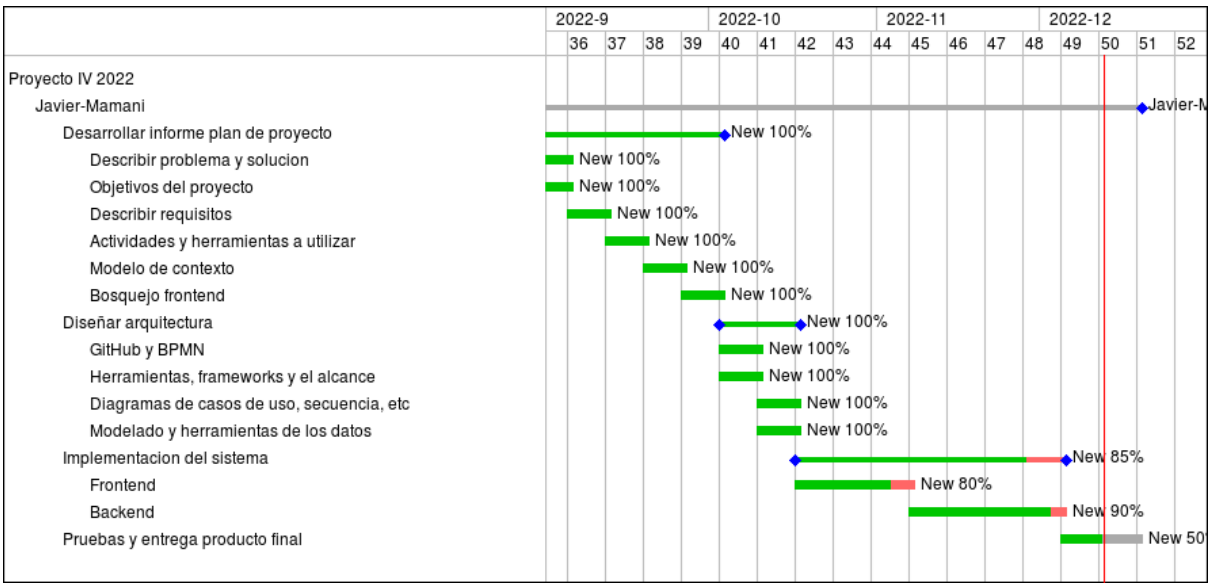

Figura 4. Carta Gantt

5.3.5 Herramientas administrativas

Para llevar a cabo la gestión del proyecto se han optado por herramientas con funcionalidades variadas y concretamente aquellas que permitan llevar a cabo una gestión de tareas y estimación del tiempo, a continuación, mencionamos estas herramientas.

Tabla 4. Herramientas administrativas

Herramienta	Descripción	Justificación
Notion	Plataforma con múltiples herramientas para la gestión de proyectos, actividades y tareas.	Para la gestión de actividades del proyecto.
Google drive	Servicio de alojamiento de archivos	Almacenar archivos.
Redmine	Plataforma con múltiples herramientas para la gestión de proyectos.	Para la gestión del proyecto en general.

5.3.6 Herramientas de gestión, configuración del software y mantenimiento

Cuando hablamos de un buen desarrollo del software se hace inevitable hablar de ingeniería de software en ese sentido nos podemos referir al ciclo de vida del software donde la mantención de un software cobra la mayor parte del tiempo, a continuación, mostramos las herramientas que se consideran para la gestión y el control de versiones del software.

Tabla 5. Herramientas de gestión y mantenimiento

Herramienta	Descripción	Justificación

Git	Es un software de control de versiones de aplicaciones.	Control de versiones
GitHub	Es un portal creado para alojar el código de las aplicaciones.	Alojamiento del software

5.3.7 Herramientas software, librerías y frameworks

Para llevar a cabo el desarrollo del proyecto se utilizará el stack de tecnologías y software MERN, donde se utiliza MySQL como base de datos relacional, ExpressJS como framework de Javascript para el entorno de ejecución NodeJS que será el encargado del backend y ReactJS como librería para el desarrollo del frontend.
Es liviano y ayuda a simplificar el lado del servidor de una aplicación web, ya que organiza el código del lado del servidor en una estructura MVC, por lo que ayuda a un desarrollo más rápido.

5.3.8 Metodología usada para el proyecto

Para llevar a cabo el desarrollo del proyecto se ha optado por la metodología de desarrollo KANBAN, la cual a comparación de otras metodologías como scrum proporcionan un flujo continuo de trabajo, otro punto a favor de Kanban es la flexibilidad que otorga ya que el cambio puede hacerse en cualquier momento, en cambio en scrum no hay cambios durante el sprint.

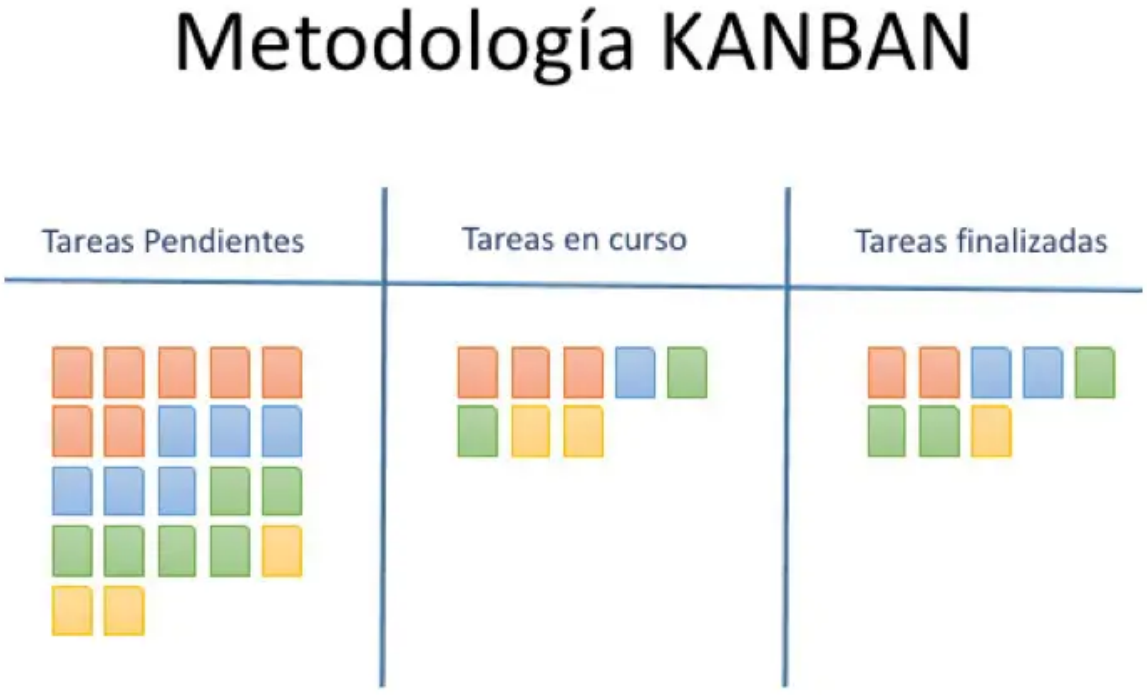

Figura 5. Metodología Kanban

5.4 Modelo de contexto del proyecto y descripción del sistema

5.4.1 Diagrama de contexto

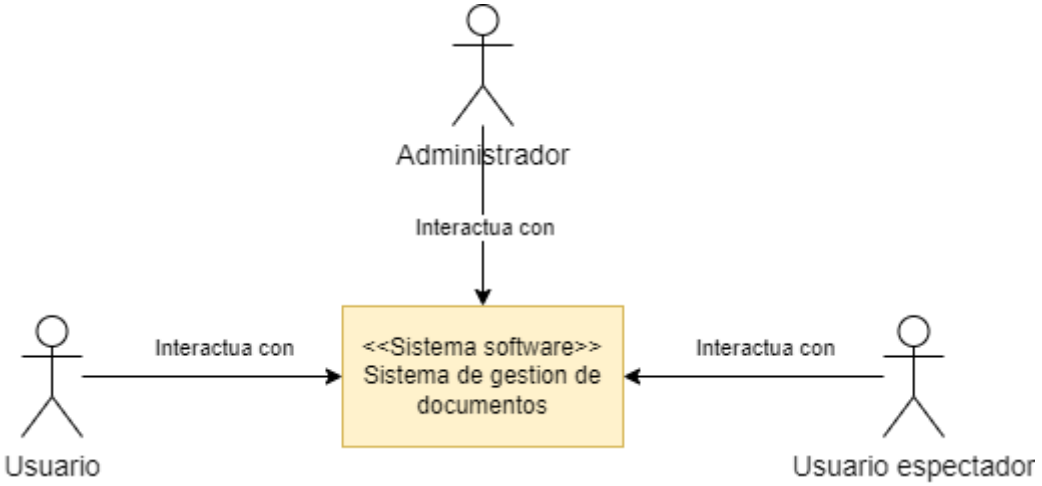

Figura 6. Diagrama de contexto

5.4.2 Identificación y descripción de subsistemas

Tabla 6. Descripción de subsistemas

Subsistema	Descripción
Interfaz de usuario	Encargada de interactuar con el usuario, ofreciendo múltiples vistas y recogiendo datos de entrada.
Análisis de datos	Encargada de recibir y manejar datos de entrada, necesita comunicarse con el subsistema de colección de datos para obtener los datos necesarios.
Control de usuario	Encargado de mantener un control de los distintos tipos de usuarios, asignando permisos y unidades correspondientes.
Recolección de datos	Encargado de gestionar los datos, interactuando con la base de datos, maneja documentos, usuarios, secciones, etc.
Notificaciones	Encargado de notificar a los usuarios.

A continuación, en la figura 7 se muestran los subsistemas identificados:

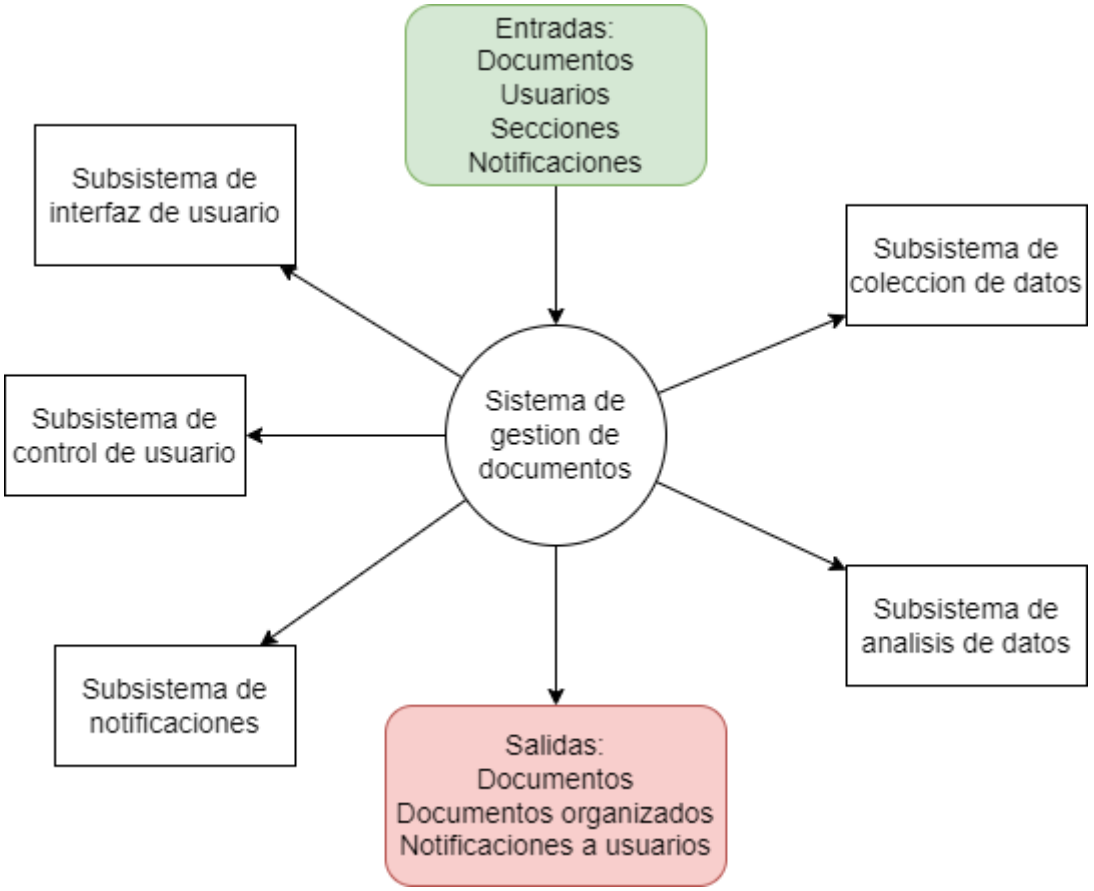

Figura 7. Diagrama de subsistemas

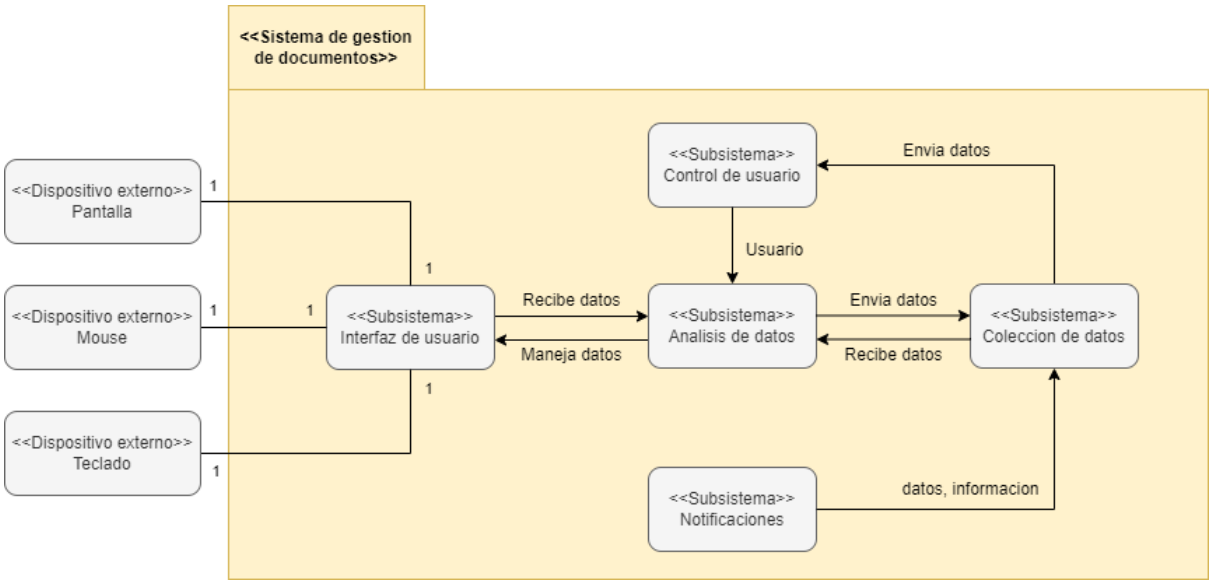

Figura 8. Diagrama de interacción de subsistemas

5.5 Aspectos iniciales del front end

5.5.1 Login

Para poder entrar al sistema el usuario deberá de iniciar sesión, ingresando el nombre y la contraseña correspondiente a su cuenta de usuario.

Figura 9. Login

5.5.2 Home

Una vez el usuario haya iniciado sesión en el sistema se le mostrará la vista del home de la aplicación, esta cuenta con tres secciones, estas son Unidades, Notificaciones y Cuenta respectivamente, además de un botón para cerrar sesión que se encuentra en la esquina inferior izquierda como se muestra en la figura 10. Al presionar Unidades se desplegarán subsecciones de este, estos son Mis unidades y Organigrama como se muestra en la figura 11, Organigrama será el encargado de mostrar el organigrama de unidades completo de la empresa y Mis unidades las unidades del usuario, esto ultimo se verá en la sección 5.5.3.

Figura 10. Home

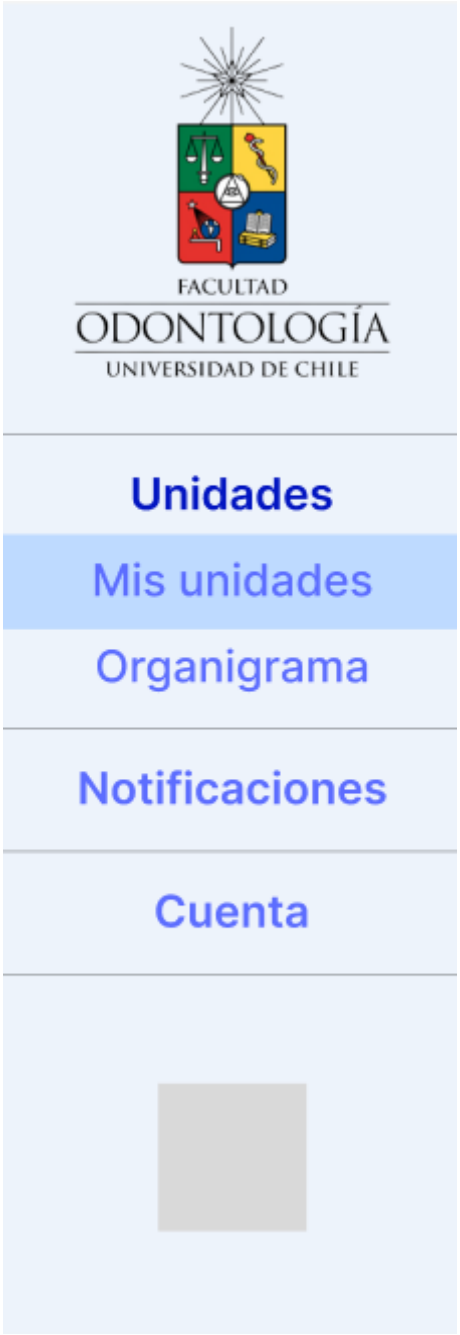

Figura 11. Menú desplegado

5.5.3 Mis unidades

Una vez seleccionada la subsección Mis unidades se mostrarán todas las unidades del usuario, estas corresponden a todas las unidades a las que tendrá acceso el usuario, como se muestra en la figura 12. El usuario podrá seleccionar alguna de sus unidades y se mostrará el gestor de archivos descrito en 5.5.4.

Figura 12. Mis unidades

5.5.4 Gestor de archivos

Una vez el usuario haya seleccionado alguna de sus unidades se desplegará el gestor de archivos correspondiente a esa unidad como se muestra en la figura 13, este gestor consta de una ruta de directorio, los archivos en sí, que podrían ser organizados en carpetas, un botón en la parte inferior para poder crear carpetas o subir archivos y una paleta de funciones a la derecha que afectará al archivo seleccionado.

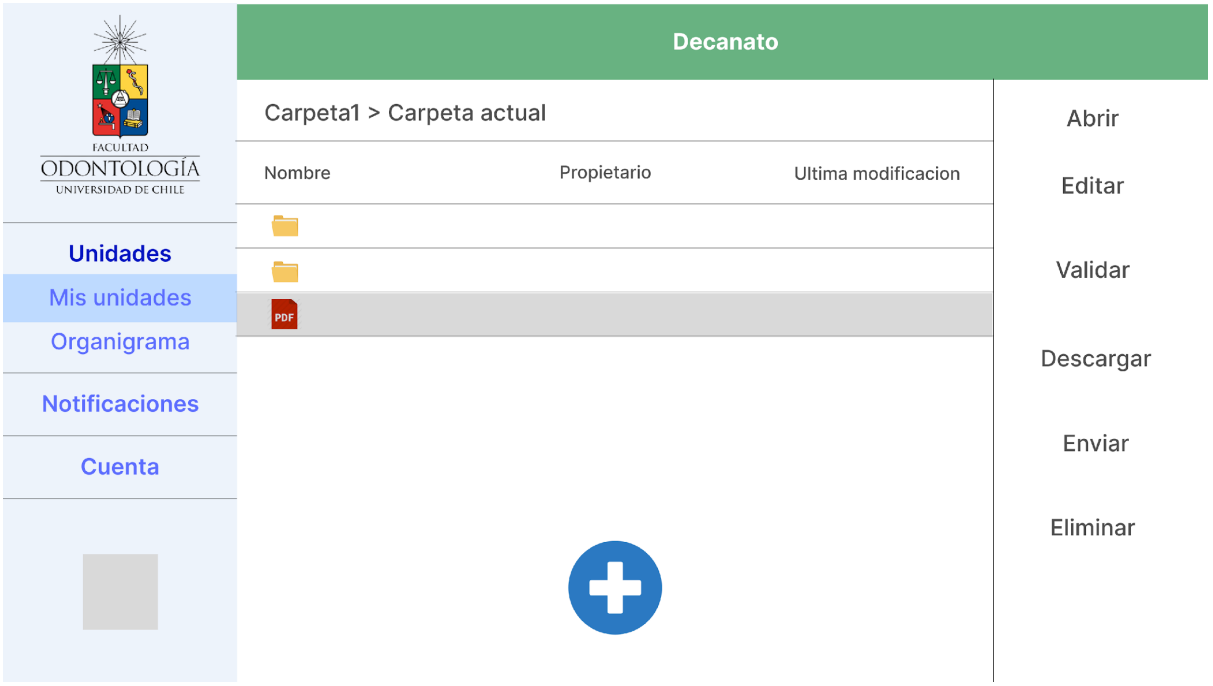

Figura 13. Gestor de archivos

5.6 Alcance de acuerdo con las herramientas

Las herramientas escogidas en su conjunto lograran crear un gestor de archivos escalable para la Facultad de Odontología de la Universidad de Chile, en este sistema se podrá realizar toda funcionalidad que anteriormente se debía de hacer manualmente, además de ayudar a la política cero papel, digitalizando todos los documentos que se comunican dentro de las unidades de la empresa, esto se consigue a la ayuda de cada herramienta escogida, en las siguiente subsecciones se describirán sus ventajas con respecto al proyecto.

5.6.1 React

- React es una de las librerías de JS más populares en todo el mundo, y la comunidad que lo apoya y desarrolla es enorme.
- El rendimiento de React ha mejorado enormemente con la introducción del DOM virtual. Dado que todos los árboles DOM virtuales son ligeros y están integrados en el servidor, se reduce la carga en el navegador. Además, dado que el proceso de enlace de datos es unidireccional, los enlaces no se asignan a los observadores (watchers) como en el caso de Angular. Respectivamente, no se crea una carga de trabajo adicional.
- React se basa en JavaScript ES6 + combinado con JSX script. JSX es una extensión para la sintaxis, lo que hace que un código JavaScript se parezca a un código escrito en HTML. Esto hace que el código sea más fácil de entender y los errores tipográficos son más fáciles de detectar.
- React es minimalista, sin inyección de dependencias, sin plantillas clásicas, sin funciones demasiado complicadas. La librería será bastante sencilla de entender si ya conoces bien JavaScript.
- React al ser una librería escalable al empezar un proyecto, este no pesa demasiado como si lo hacen algunos frameworks.

5.6.2 MySQL

- MySQL ofrece compatibilidad con la mayoría de las principales plataformas informáticas, como Linux, macOS, Microsoft Windows y Ubuntu. Además proporciona un alto rendimiento para el almacenamiento de grandes volúmenes de datos o Business Intelligence. Esta solución se ha utilizado por muchos años en todos los sectores, por lo tanto, hay muchos recursos disponibles para los desarrolladores.
- La seguridad de los datos está garantizada por las funciones de Access Privilege System y de User Account Management, además de la criptografía de contraseña. Por tanto, MySQL es altamente seguro, gracias a varias funciones de seguridad, algunas bastante avanzadas.

5.6.3 NodeJS

- La compilación de Node.js se realiza en tiempo de ejecución, Just In Time (JIT), esto trae consigo una mayor optimización a las funciones que más veces sean llamadas.
- Mediante clusters permite tener una escalabilidad alta.
- Podemos expandir nuestro código añadiendo módulos de forma fácil gracias al Node Package Manager (NPM).
- Un alto rendimiento en proyectos donde necesitemos ejecución en tiempo real.
- Podremos realizar front-end, back-end y hasta una aplicación móvil con un mismo lenguaje.

5.6.4 ExpressJS

- Express.js permite a los desarrolladores utilizar un solo lenguaje para el frontend y el backend. Es decir, es básicamente una plataforma completa, de pila completa. Como resultado, las empresas pueden crear aplicaciones rápidamente.
- La integración juega un papel vital en cualquier proyecto de desarrollo de aplicaciones. Con un marco de trabajo sin oposición, la integración de middleware y servicios de terceros es fácil. Encontrar soluciones a los problemas de desarrollo no será un gran desafío. Un marco de trabajo no orientado permite a las empresas construir su aplicación de la manera que deseen y elegir el middleware que sea ideal para ellos.
- Las negociaciones de contenido, las vistas dinámicas, el enrutamiento para múltiples peticiones y las vistas a nivel de aplicación son sólo algunas de las características que hacen que Express.js sea flexible y funcional.

5.7 Arquitectura del sistema

El tipo de arquitectura del sistema está basado en la arquitectura monolítica, lo que significa que la estructura del software y todos los aspectos de este, están compilados como una unidad unificada, un solo servidor y que ésta es autónoma e independiente de otras aplicaciones.
A continuación, se muestra un diagrama describiendo la arquitectura del sistema y cómo se comunica éste con el usuario.

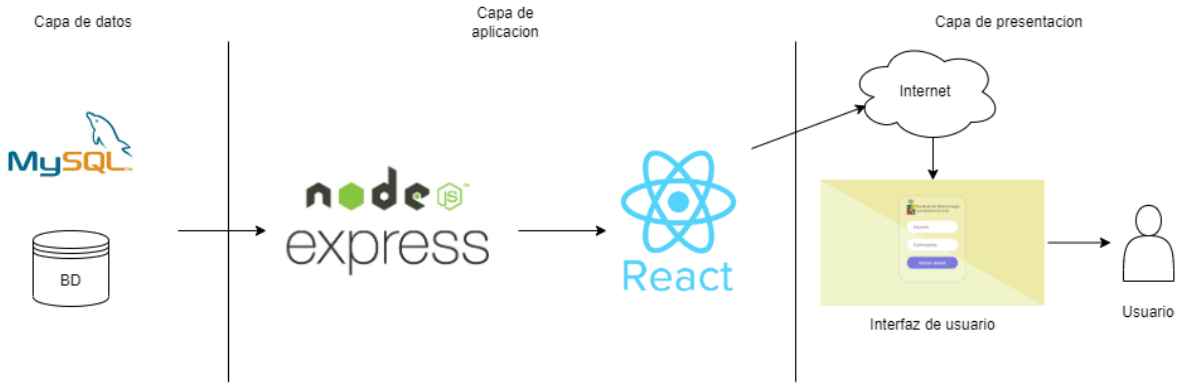

Figura 14. Arquitectura del sistema

5.8 Business Process Management

El diagrama BPMN correspondiente a la figura 15 describe el flujo principal del sistema de gestión de proyectos, que va desde iniciar sesión en el sistema hasta notificar a los usuarios cambios hechos respecto a los documentos, como es solo el flujo principal se ignoraron algunos procesos secundarios como lo son la creación de cuentas, permisos de usuario, funciones correspondientes al tipo de usuarios y unidad, vista de cuenta, organigrama y documentos oficiales, etc.

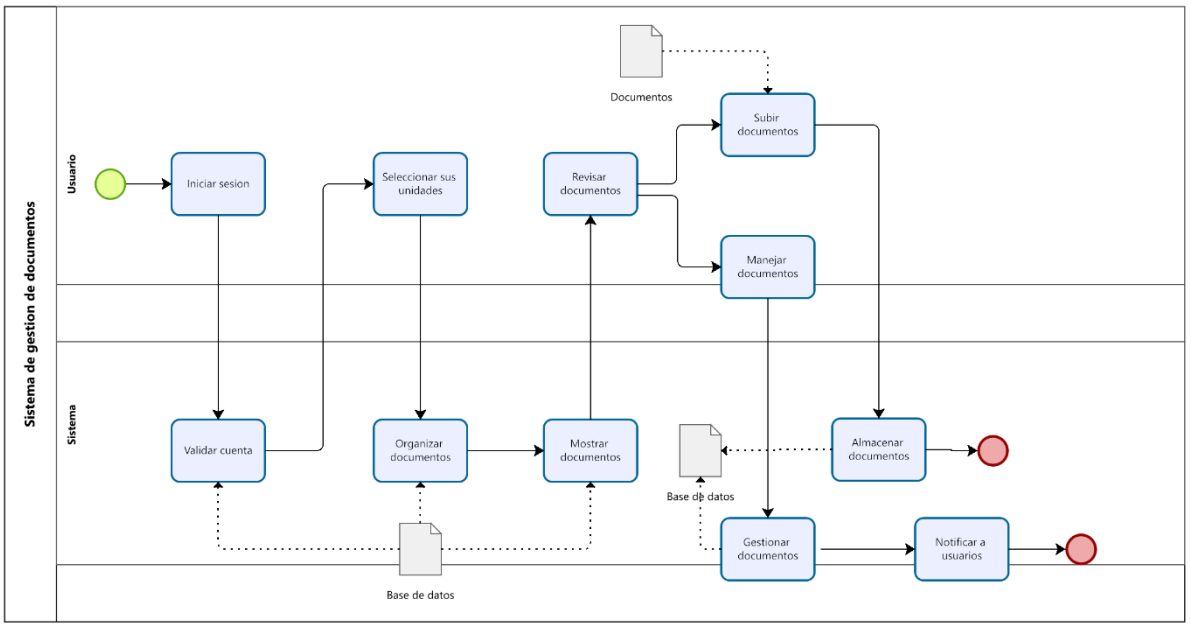

Figura 15. Diagrama BPMN

5.9 Repositorio de Github

El avance de este proyecto se puede encontrar en el repositorio remoto de Github correspondiente al siguiente enlace: https://github.com/ZeyronJ/gestor-documentos

A continuación, se muestran capturas del estado actual del proyecto en el repositorio.

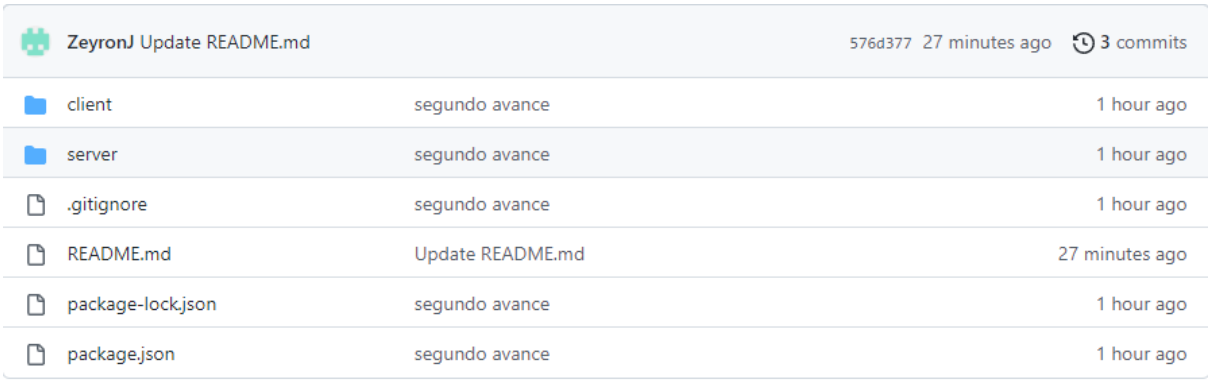

Figura 16. Repositorio GitHub: Vista general

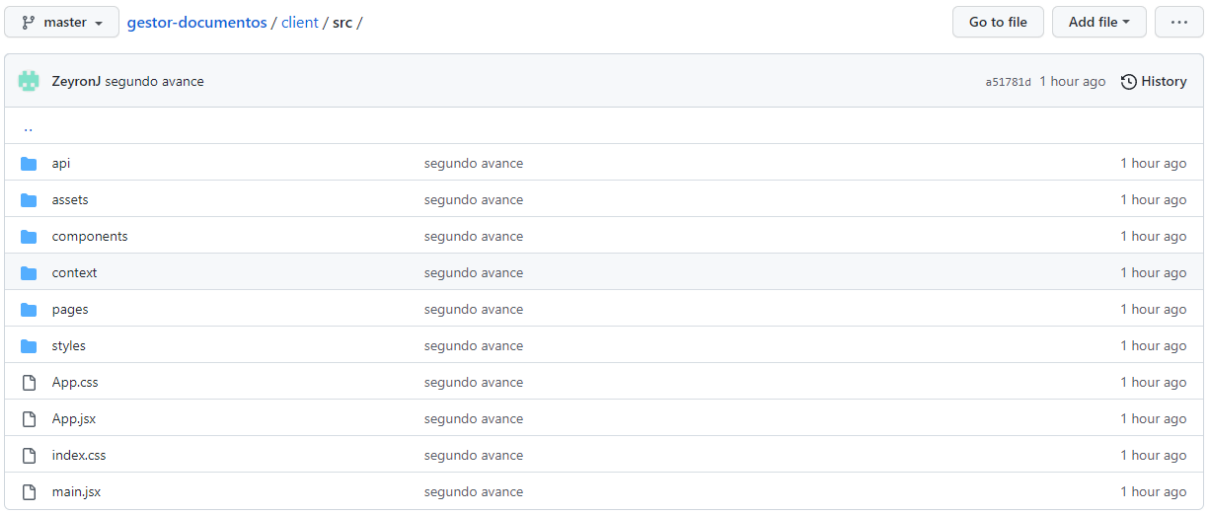

Figura 17. Repositorio GitHub: Vista cliente

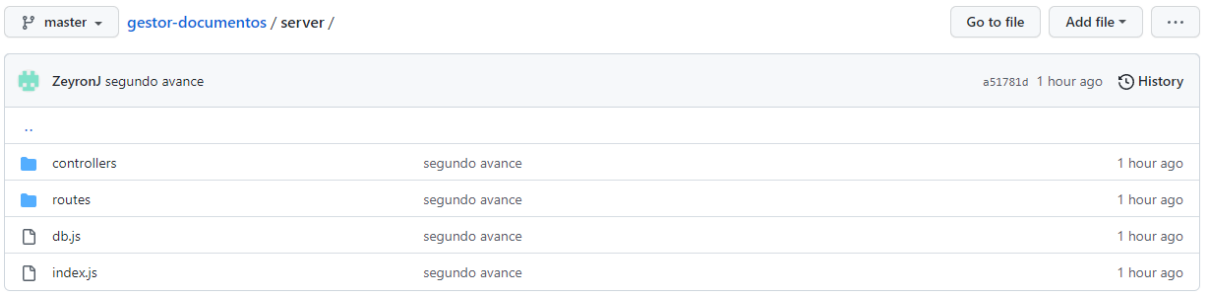

Figura 18. Repositorio GitHub: Vista server

5.10 Diagramas de casos de uso

Figura 19. Casos de uso

INICIAR SESIÓN	
Actores	Usuario

Descripción	El usuario inicia sesión en el sistema
Condición previa	El usuario debe tener una cuenta creada en el sistema
Flujo normal	1. El usuario entra al sistema. 2. El usuario ingresa los datos necesarios para iniciar sesión (usuario y contraseña) y presiona “Iniciar sesión”. 3. El sistema valida la cuenta de usuario. 4. El sistema lo redirecciona hacia el home del sistema.
Flujo alternativo: Cuenta de usuario invalida	4.1. El sistema muestra el mensaje de error “cuenta invalida”.
Post condición	El usuario inició sesión y se encuentra en el home del sistema.

CREAR USUARIO	
Actores	Decana (Administrador)
Descripción	Se crea un nuevo usuario
Condición previa	La decana tiene los datos del nuevo usuario y debe de haber ingresado al sistema.
Flujo normal	1. La decana se posiciona en el apartado de “Crear nuevo usuario”. 2. La decana ingresa los datos necesarios para crear el usuario (nombre completo, nombre usuario, contraseña, teléfono, rut). 3. La decana ingresa el tipo de usuario que es, a qué unidades tendrá acceso y presiona “Crear usuario”. 4. El sistema valida que el usuario no esté repetido. 5. El sistema muestra un mensaje de éxito al crear usuario.
Flujo alternativo: Usuario repetido	5.1 El sistema muestra mensaje de usuario ya existente.
Post condición	Se creó y agregó un nuevo usuario al sistema.

EDITAR USUARIO

Actores	Decana (Administrador)
Descripción	Se edita un usuario ya existente
Condición previa	La decana posee los nuevos datos del usuario y debe de haber ingresado al sistema.
Flujo normal	1. La decana se posiciona en el apartado de “Editar usuario”. 2. El sistema muestra todos los usuarios ya existentes en el sistema junto a un botón “Editar”. 3. La decana presiona “Editar” en el usuario correspondiente. 4. El sistema muestra los datos del usuario pudiendo sobreescribirlos. 5. La decana edita los datos correspondientes del usuario y presiona “Guardar”. 6. El sistema muestra un mensaje de éxito al guardar usuario.
Flujo alternativo: Usuario repetido	6.1 El sistema muestra mensaje de usuario ya existente.
Post condición	Se editó los datos de un usuario ya existente en el sistema.

ELIMINAR USUARIO	
Actores	Decana (Administrador)
Descripción	Se elimina un usuario ya existente en el sistema
Condición previa	La decana posee conocimiento del usuario a eliminar y debe de haber ingresado al sistema.
Flujo normal	1. La decana se posiciona en el apartado de “Eliminar usuario”. 2. El sistema muestra todos los usuarios ya existentes en el sistema junto a un botón “Eliminar”. 3. La decana presiona “Eliminar” en el usuario correspondiente. 4. El sistema muestra un mensaje de éxito al eliminar usuario.
Post condición	Se eliminó un usuario.

SUBIR DOCUMENTOS

Actores	Usuario
Descripción	Se sube un documento al gestor de documentos
Condición previa	El usuario posee el documento a subir, el usuario debe tener los permisos correspondientes y el usuario debe de haber iniciado sesión.
Flujo normal	1. El usuario debe de seleccionar “Mis unidades”. 2. El sistema muestra las unidades permitidas del usuario. 3. El usuario selecciona la unidad correspondiente. 4. El sistema muestra la interfaz del gestor de documentos. 5. El usuario selecciona la carpeta correspondiente donde subirá el archivo y presiona “Subir documento”. 6. El usuario selecciona el documento a subir y guarda. 7. El sistema muestra un mensaje de éxito al subir el documento.
Post condición	Se subió un nuevo documento al gestor de archivos.

ELIMINAR DOCUMENTO	
Actores	Usuario
Descripción	Se elimina un documento existente del gestor de documentos.
Condición previa	El usuario posee conocimiento del documento a eliminar, tiene los permisos necesarios y debe de haber iniciado sesión.
Flujo normal	1. El usuario debe de seleccionar “Mis unidades”. 2. El sistema muestra las unidades permitidas del usuario. 3. El usuario selecciona la unidad correspondiente. 4. El sistema muestra la interfaz del gestor de documentos. 5. El usuario selecciona la carpeta correspondiente, selecciona el documento a eliminar y presiona la opción “Eliminar”. 6. El sistema muestra un mensaje de éxito al eliminar documento.
Post condición	Se eliminó un documento del gestor de documentos.

BAJAR DOCUMENTO	
Actores	Usuario
Descripción	Se descarga un documento existente en el sistema.
Condición previa	El usuario debe de haber iniciado sesión.
Flujo normal	1. El sistema muestra los documentos existentes en el sistema. 2. El usuario selecciona y presiona “Descargar”. 3. El sistema empieza la descarga del documento.
Post condición	Se descargo un documento.

EDITAR DOCUMENTO	
Actores	Usuario
Descripción	Se edita un documento existente en el gestor de documentos.
Condición previa	El usuario tiene los permisos necesarios y debe de haber iniciado sesión.
Flujo normal	1. El usuario debe de seleccionar “Mis unidades”. 2. El sistema muestra las unidades permitidas del usuario. 3. El usuario selecciona la unidad correspondiente. 4. El sistema muestra la interfaz del gestor de documentos. 5. El usuario selecciona la carpeta correspondiente, selecciona el documento a editar y presiona la opción “Editar”. 6. Dependiendo del tipo de archivo se abrirá un editor donde se podrá editar el documento, una vez editado se debe de guardar los cambios. 7. El sistema muestra un mensaje de éxito al guardar cambios.
Post condición	Se editó y guardó los cambios de un documento existente en el gestor de documentos.

ENVIAR DOCUMENTOS	
Actores	Usuario
Descripción	Se envía un documento a otro lugar dentro del sistema.

Condición previa	El usuario tiene los permisos necesarios y debe de haber iniciado sesión.
Flujo normal	1. El usuario debe de seleccionar “Mis unidades”. 2. El sistema muestra las unidades permitidas del usuario. 3. El usuario selecciona la unidad correspondiente. 4. El sistema muestra la interfaz del gestor de documentos. 5. El usuario selecciona la carpeta correspondiente, selecciona el documento a enviar y presiona la opción “Enviar”. 6. El sistema muestra un gestor de rutas donde se encuentran todas las unidades y secciones/carpetas. 7. El usuario selecciona la ruta a la que desea enviar el documento y presiona “Aceptar”. 8. El sistema envía el documento y muestra un mensaje de éxito “Documento enviado”.
Flujo alternativo: Ruta no permitida	8.1 El sistema muestra un mensaje de “Ruta no permitida” y vuelve al gestor de documentos.
Post condición	Se envió un documento existente en el gestor de documentos a otra unidad y/o sección.

CREAR SECCIONES	
Actores	Usuario
Descripción	Se crea y agrega una nueva sección/carpeta dentro de alguna unidad u otra sección del gestor de documentos.
Condición previa	El usuario tiene los permisos necesarios y debe de haber iniciado sesión.
Flujo normal	1. El usuario debe de seleccionar “Mis unidades”. 2. El sistema muestra las unidades permitidas del usuario. 3. El usuario selecciona la unidad correspondiente. 4. El sistema muestra la interfaz del gestor de documentos. 5. El usuario selecciona la sección correspondiente y presiona la opción “Crear sección”. 6. El usuario ingresa los datos necesarios para crear la sección (nombre, descripción y usuarios permitidos) y presiona “Aceptar”. 7. El sistema crea la sección correspondiente.
Post condición	Se creó una nueva sección dentro del gestor de

	documentos.

NOTIFICAR USUARIOS	
Actores	Usuario
Descripción	Un usuario manda una notificación a otros usuarios que se almacena en el apartado de “Mis notificaciones”.
Condición previa	El usuario tiene los permisos necesarios y debe de haber iniciado sesión.
Flujo normal	1. El usuario selecciona el apartado de Notificaciones/Enviar notificación. 2. El usuario redacta la notificación, selecciona a qué usuarios notificar y presiona el botón “Notificar”. 3. El sistema notifica a los usuarios correspondientes. 4. El sistema muestra un mensaje de éxito al notificar a los usuarios.
Post condición	Se envió y almacenó notificacion a usuarios

Visualizar documento	
Actores	Usuario
Descripción	El usuario visualiza algún documento dentro del sistema (En unidades o en documentos oficiales).
Condición previa	El usuario debe de haber iniciado sesión.
Flujo normal	1. El usuario selecciona algún documento dentro del sistema. 2. El usuario presiona el botón “Abrir” o da doble click sobre el documento. 3. El sistema muestra el contenido del archivo.
Post condición	El usuario visualizo un documento.

5.11 Modelamiento de datos

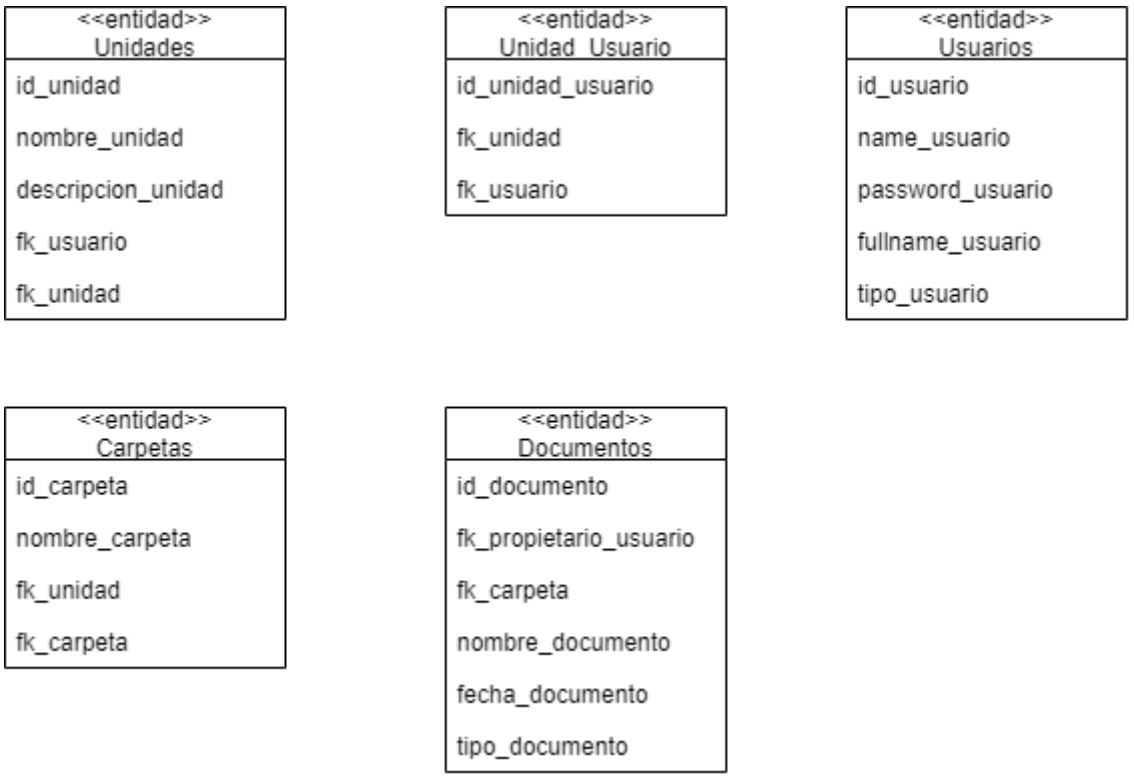

Figura 20. Diagrama de clases

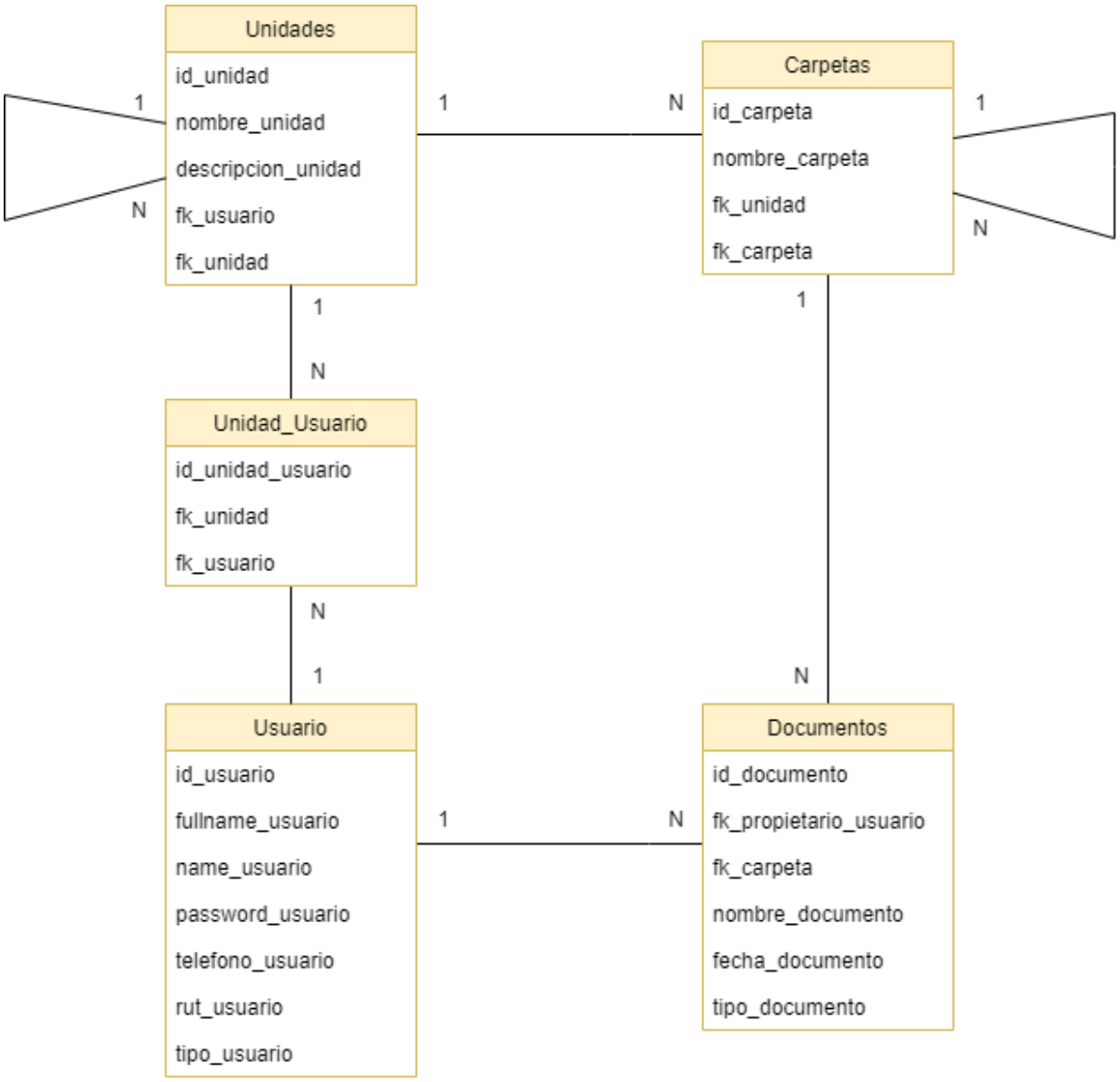

Figura 21. Modelo relacional

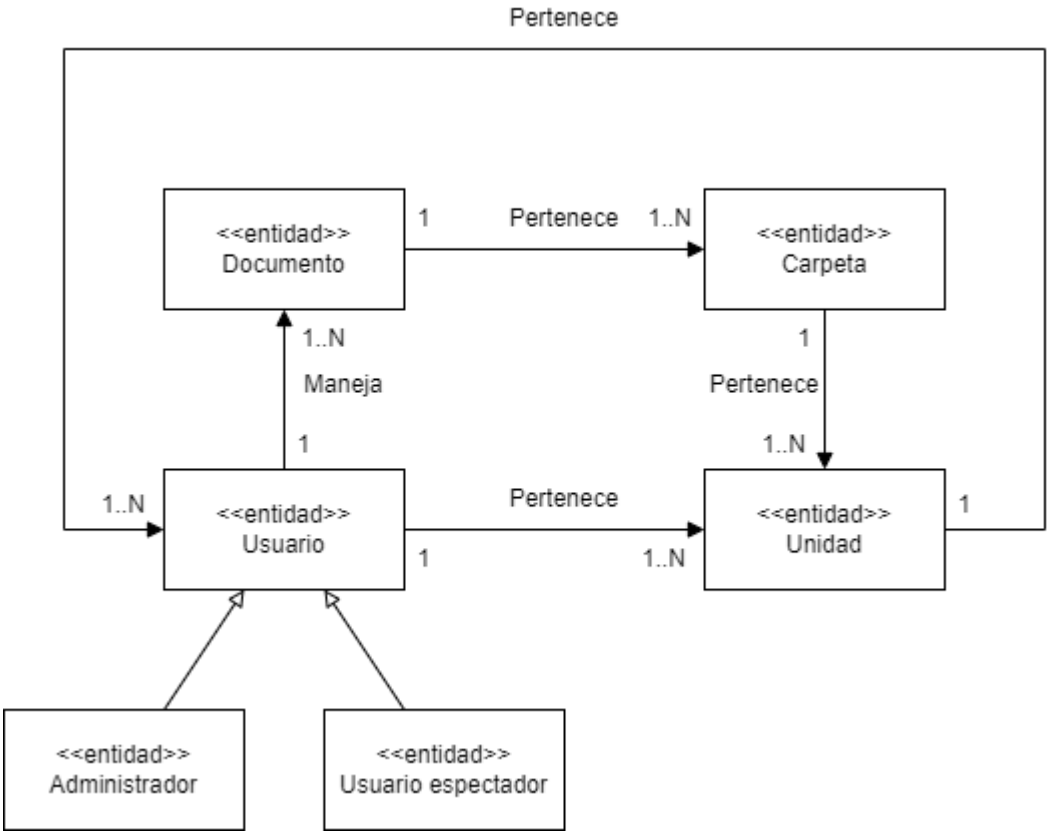

Figura 22. Diagrama de clases con asociaciones

5.12 Diagramas de secuencia

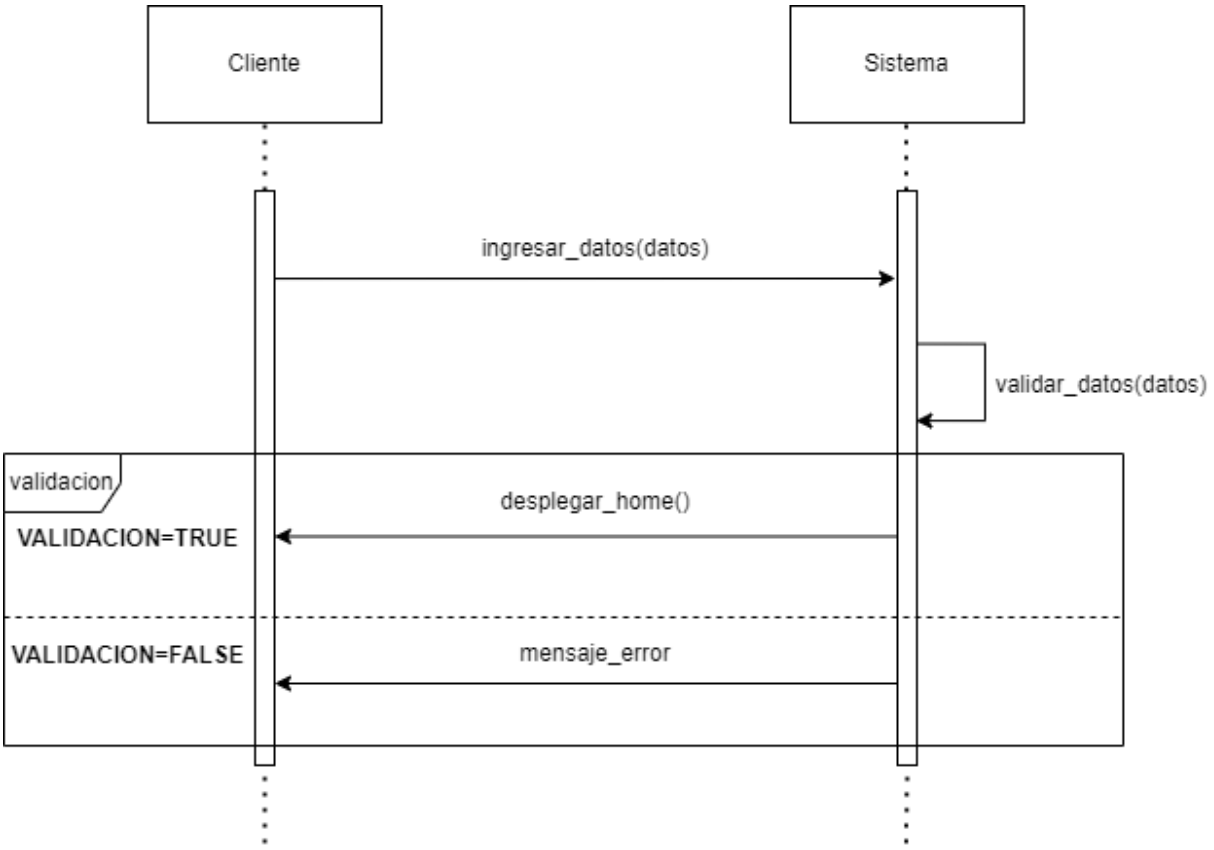

Figura 23. Diagrama de secuencia: Iniciar sesión

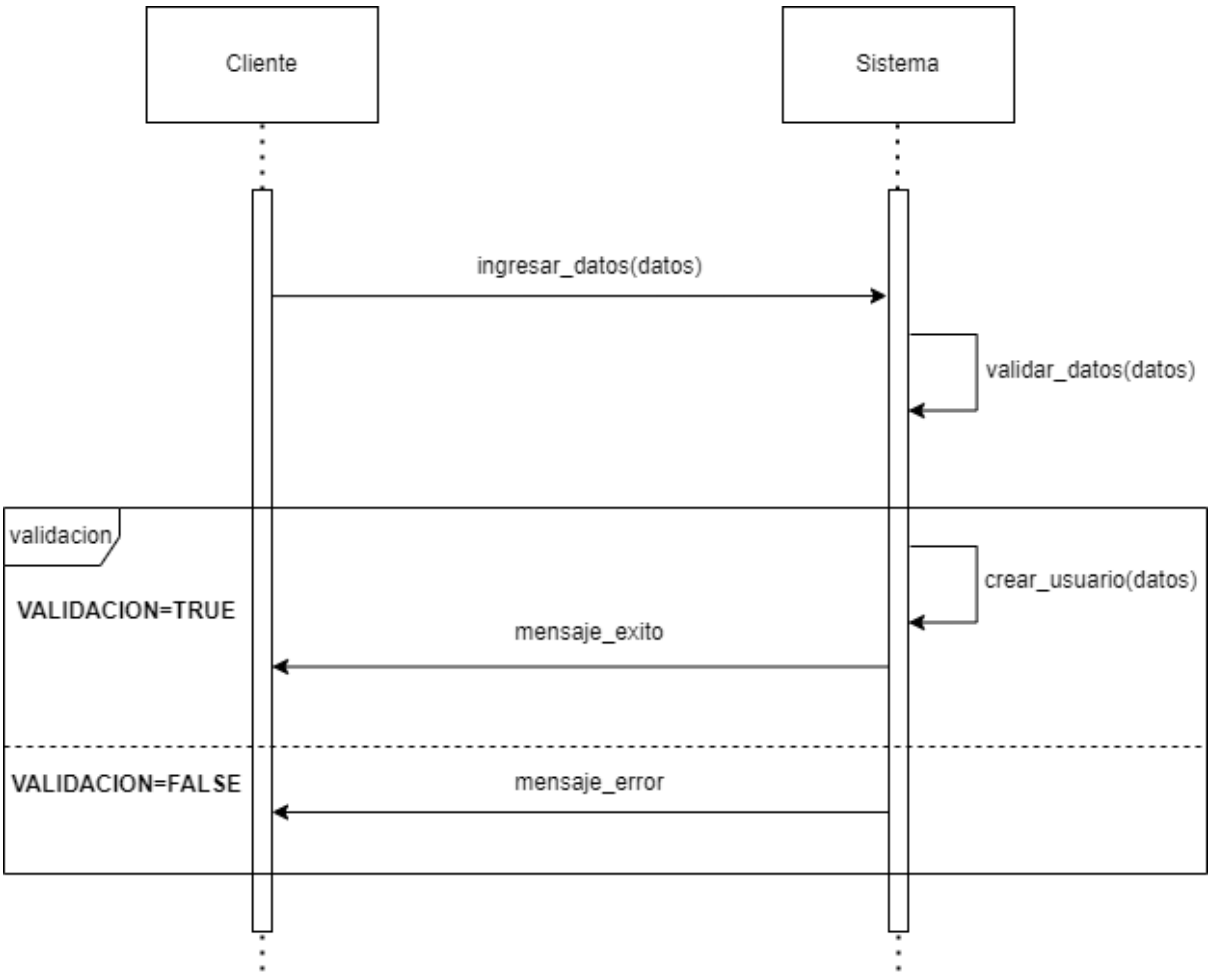

Figura 24. Diagrama de secuencia: Crear usuarios

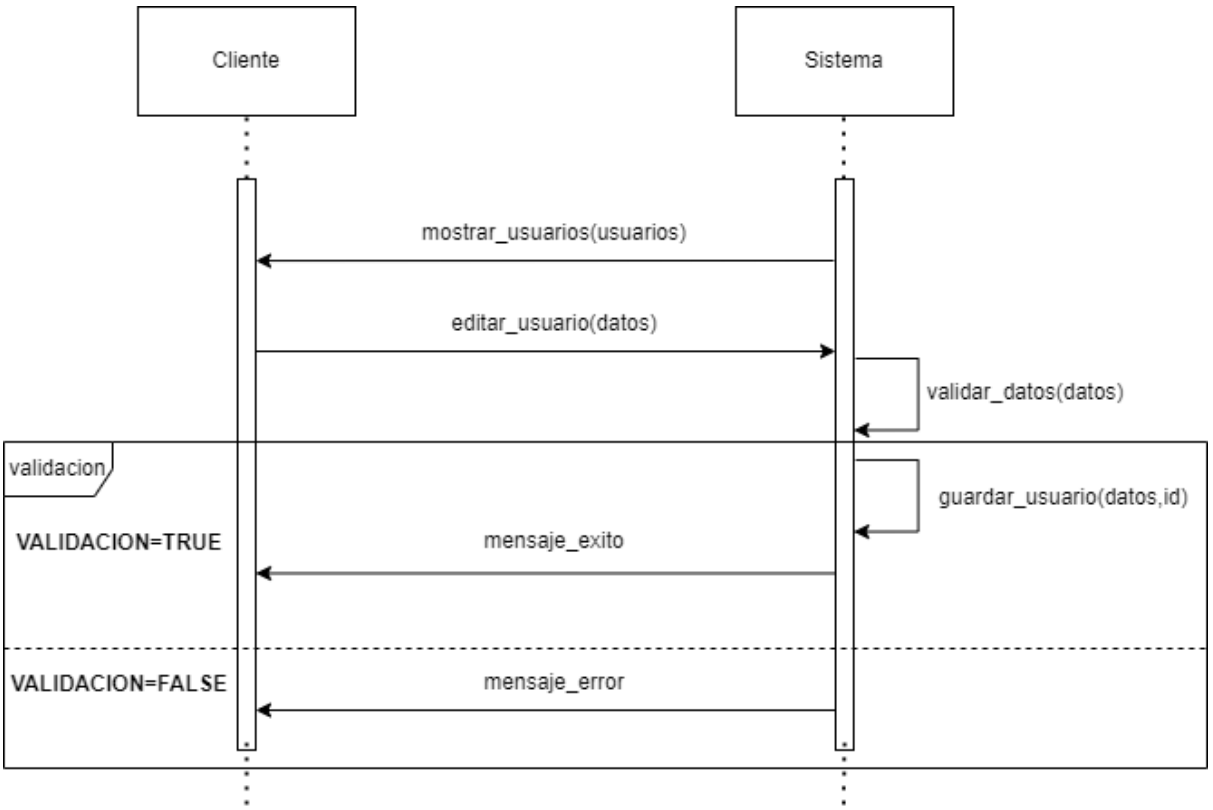

Figura 25. Diagrama de secuencia: Editar usuarios

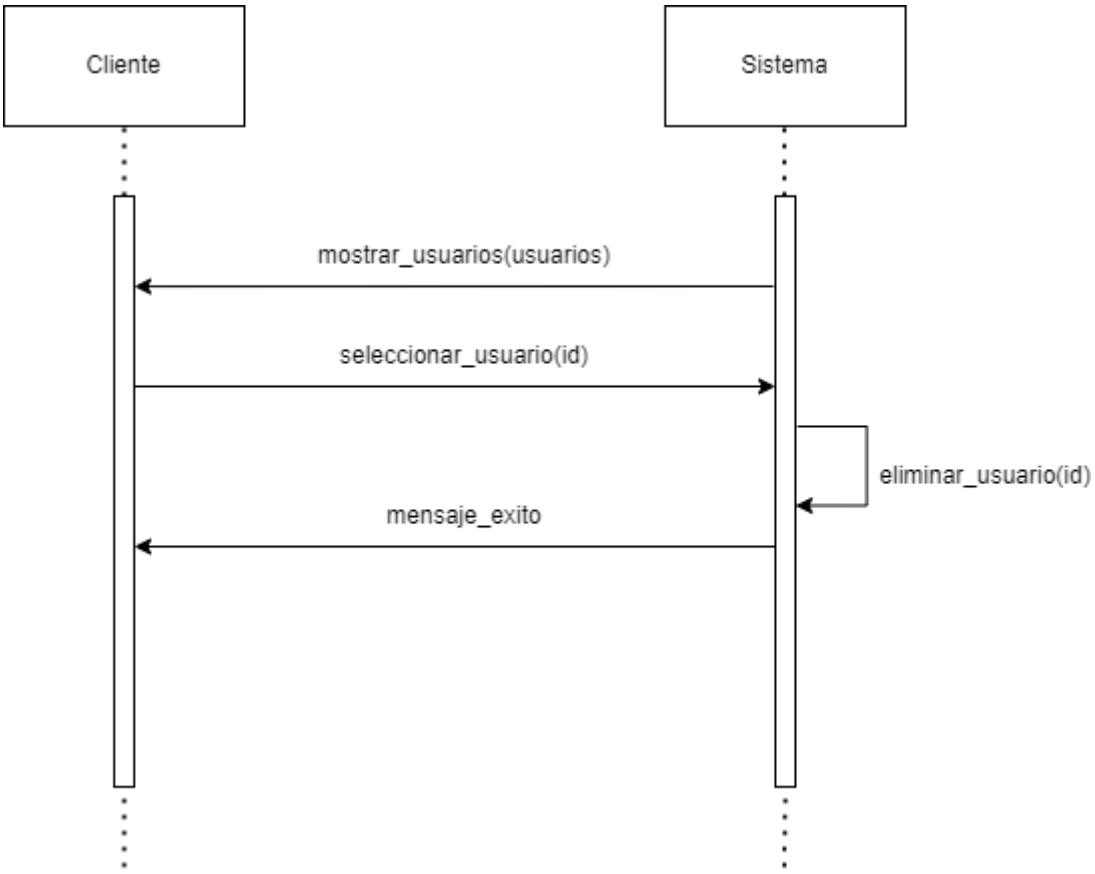

Figura 26. Diagrama de secuencia: Eliminar usuarios

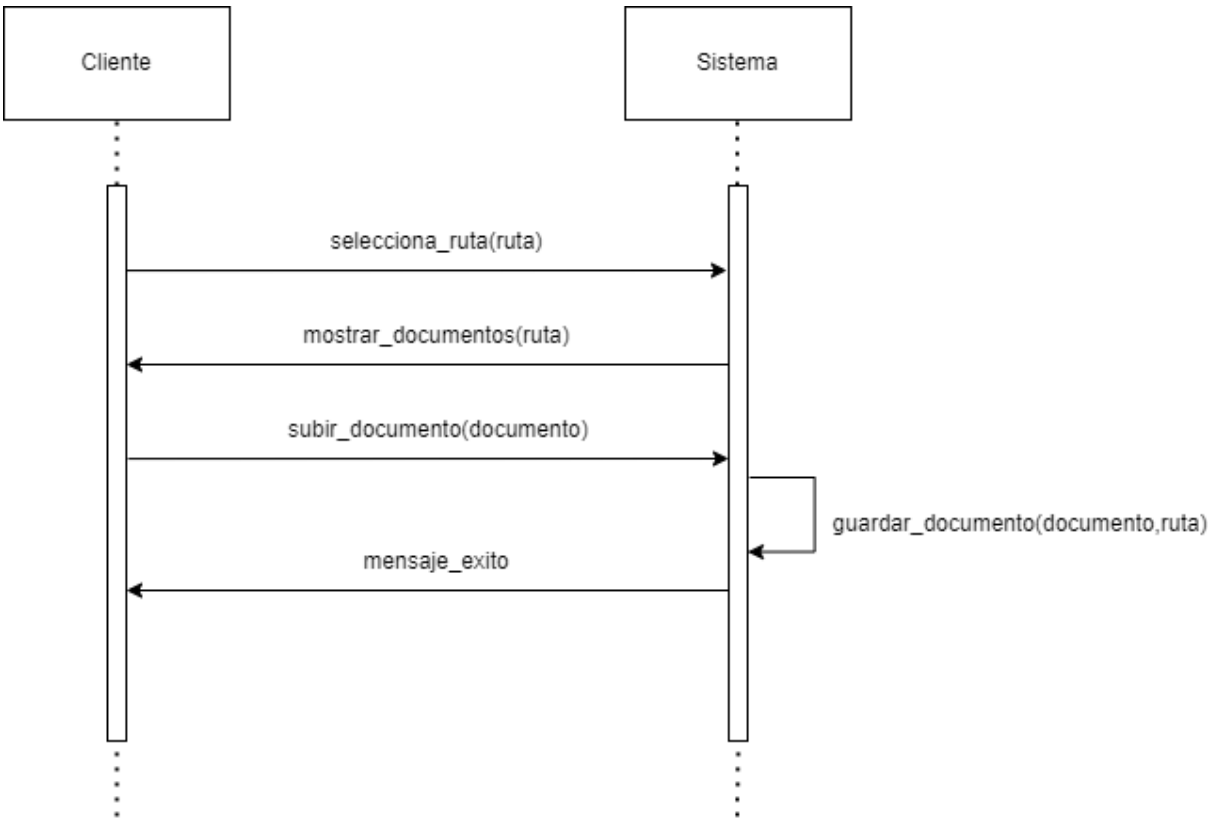

Figura 27. Diagrama de secuencia: Subir documentos

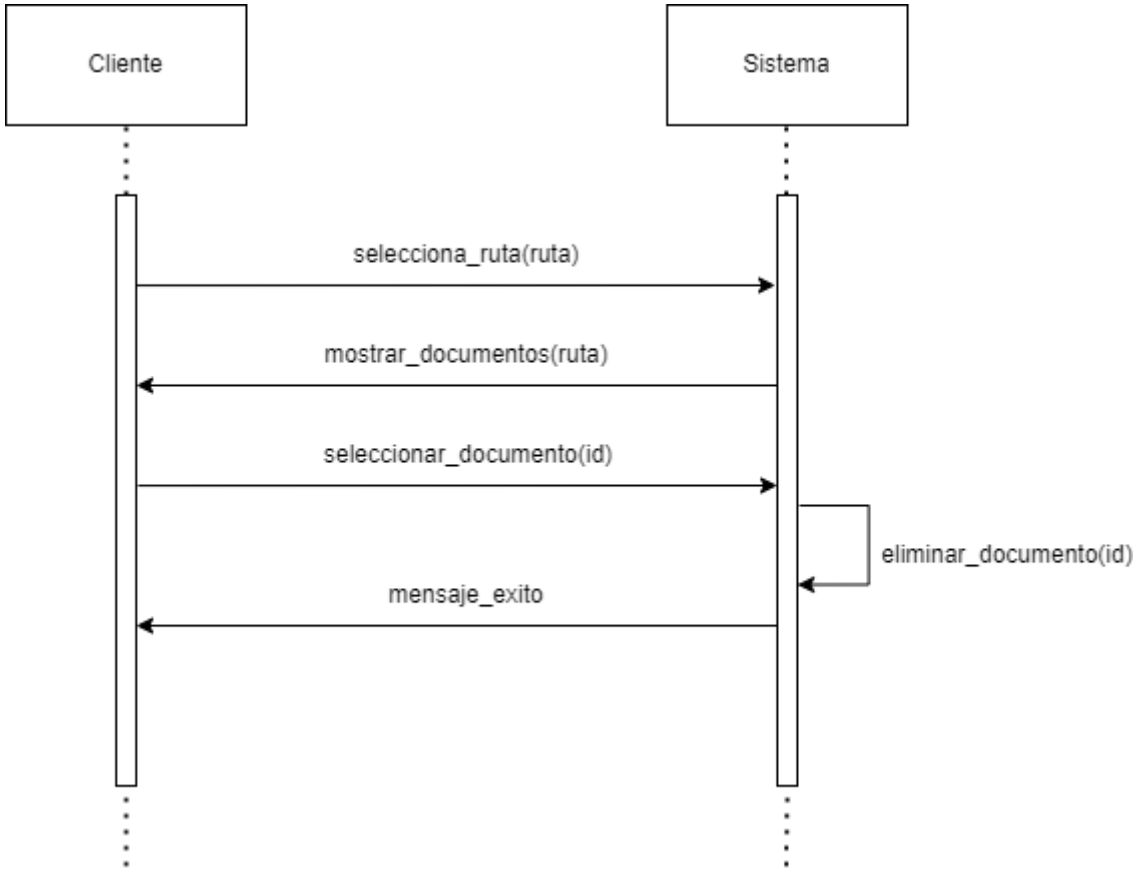

Figura 28. Diagrama de secuencia: Eliminar documentos

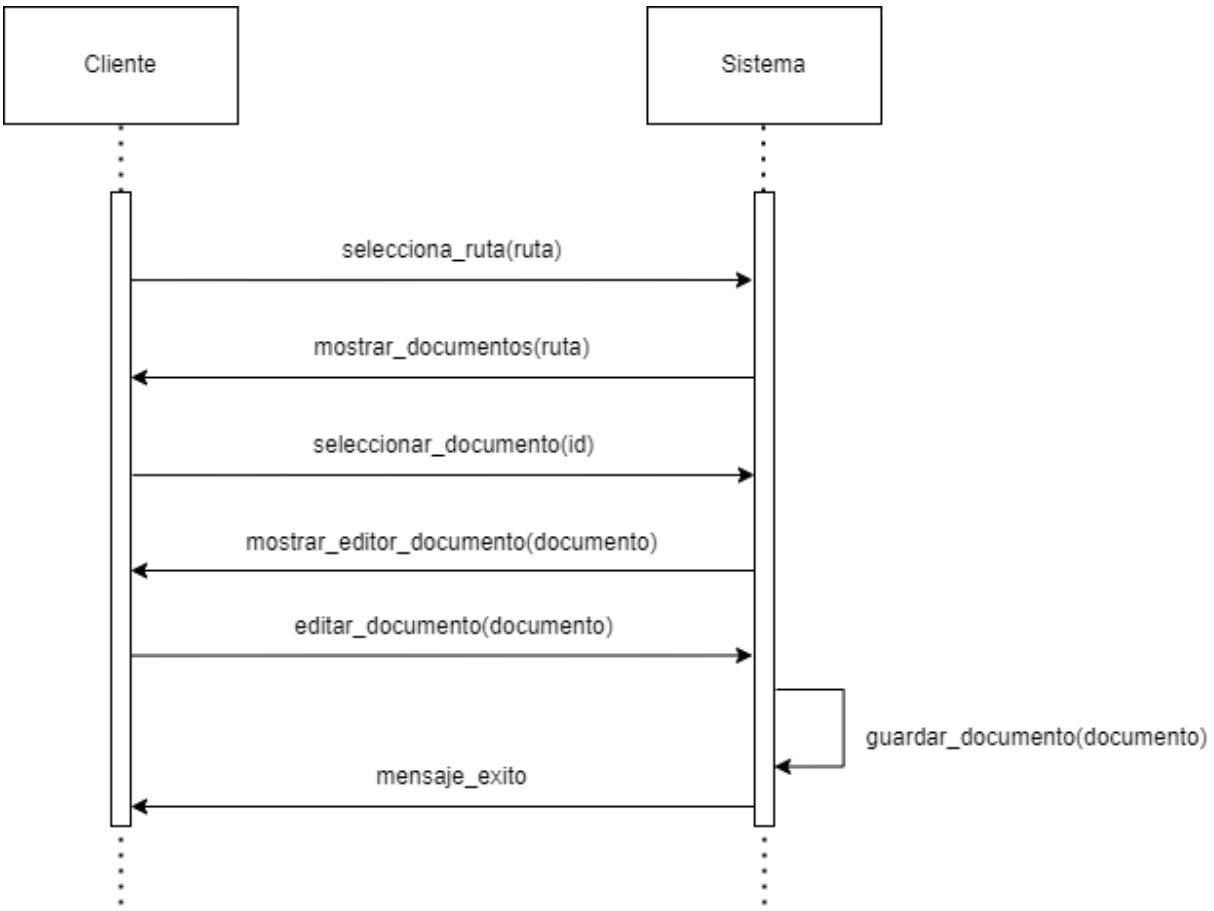

Figura 29. Diagrama de secuencia: Editar documentos

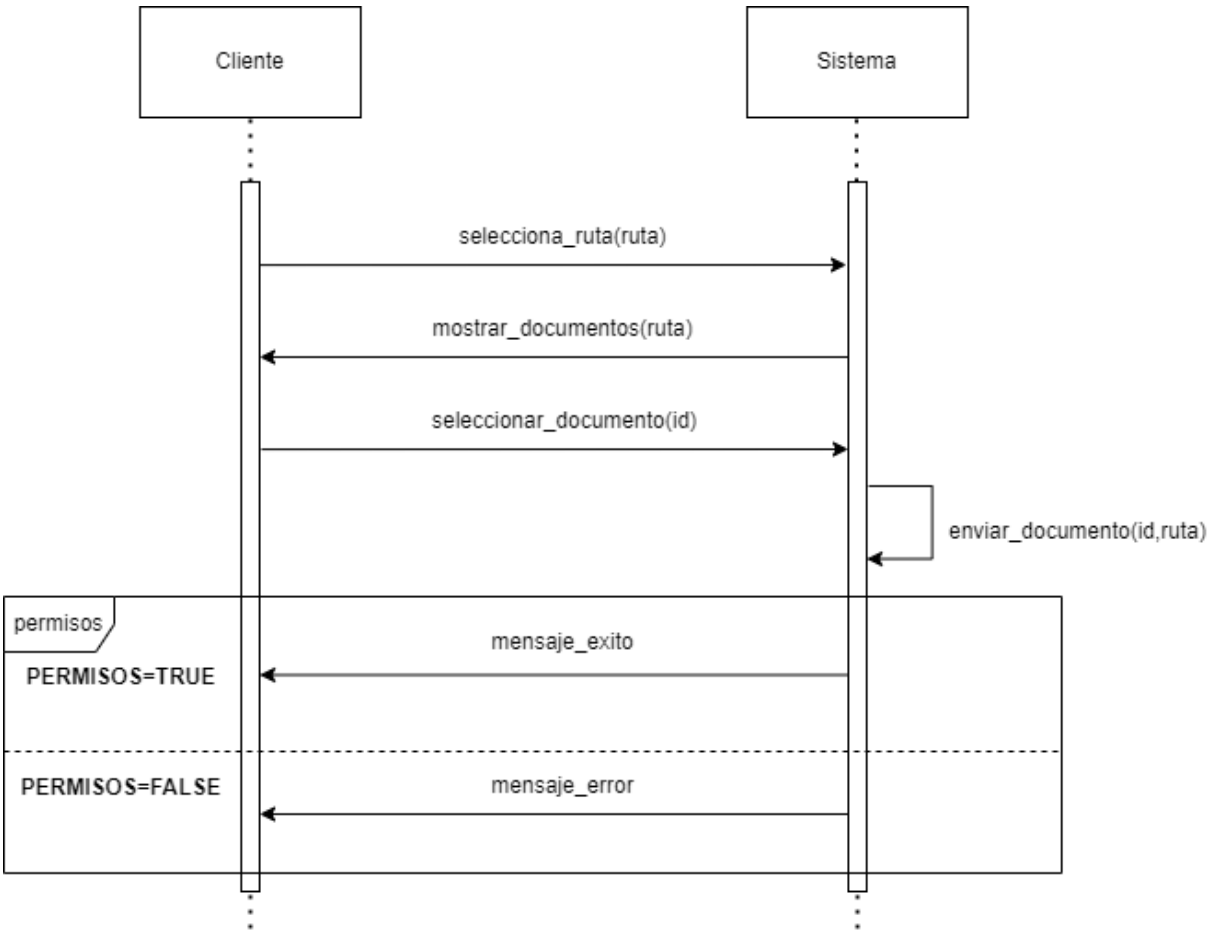

Figura 30. Diagrama de secuencia: Enviar documentos

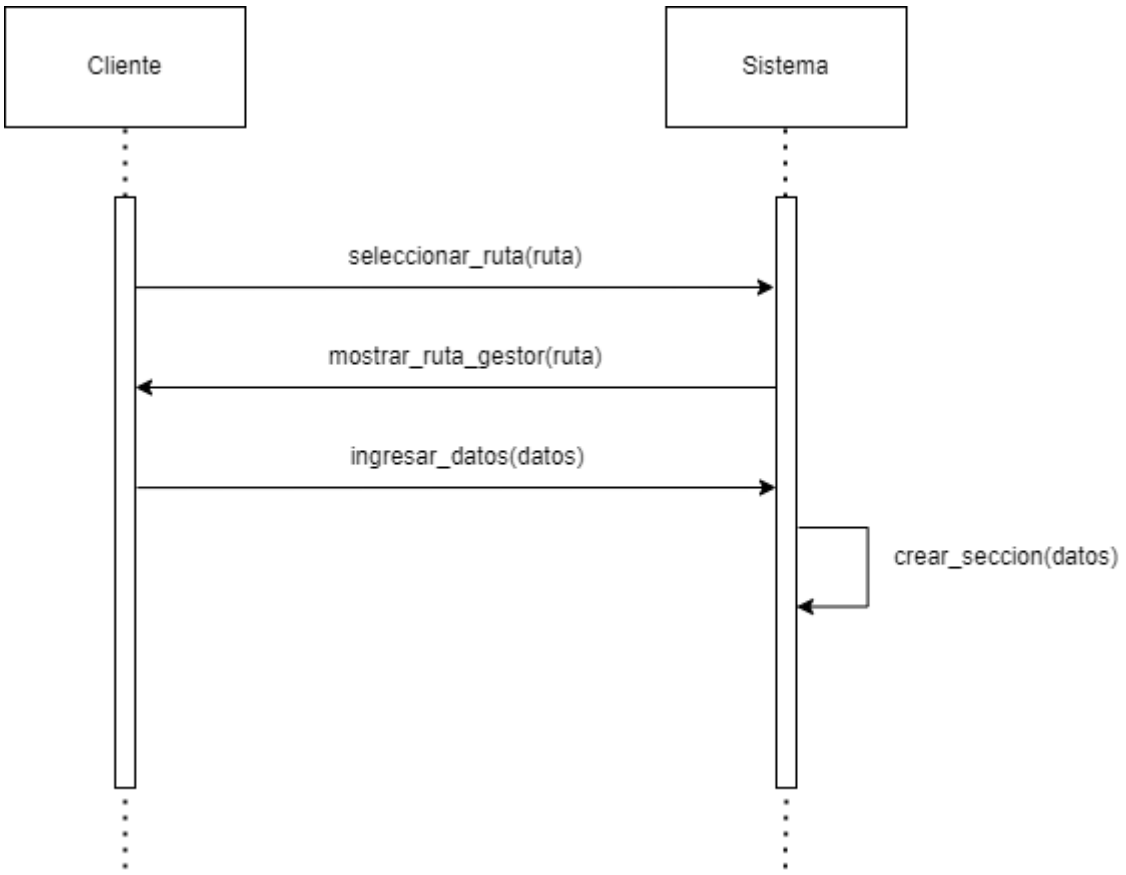

Figura 31. Diagrama de secuencia: Crear secciones

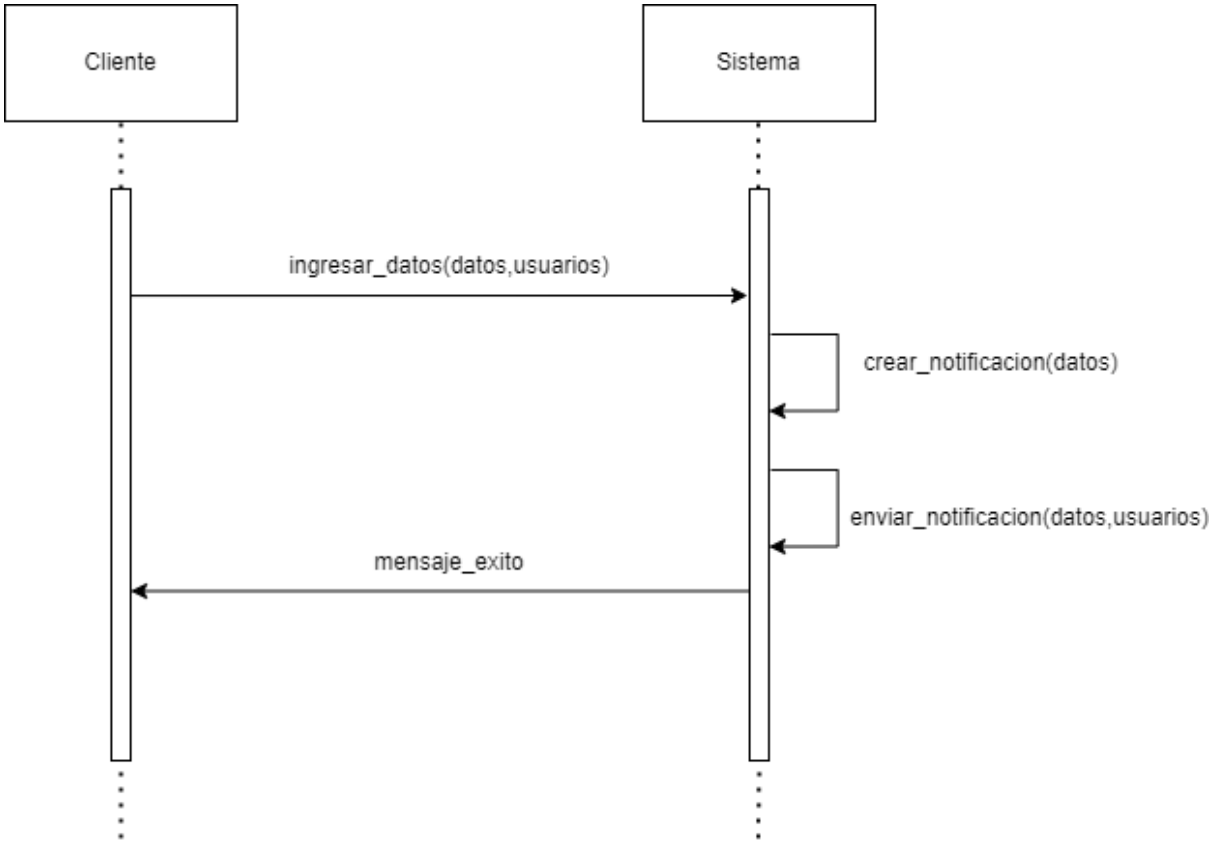

Figura 32. Diagrama de secuencia: Notificar usuarios

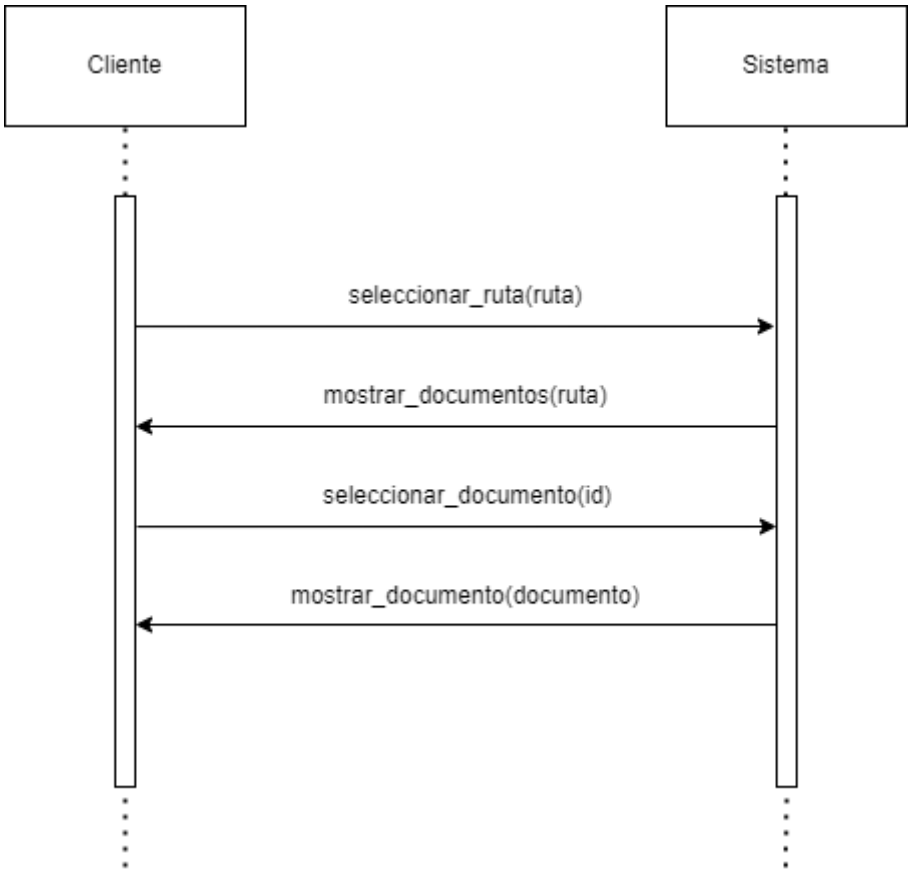

Figura 33. Diagrama de secuencia: Visualizar documentos

6. Implementación del sistema

6.1 Frontend

Dado que se utilizó react como framework para el frontend, partes importantes que se utilizan son los componentes y contextos de estos, se explicara detalladamente estos en sus respectivas secciones.

6.1.1 Componentes

Los componentes en React son funciones que devuelven elementos React, estos elementos utilizan el lenguaje de programación JSX, que es una combinación de código javascript y html, esto permite poder manipular código html usando javascript. Los componentes son porciones de código que se utilizan para la creación de páginas.
A continuación se muestran los componentes utilizados para el proyecto:

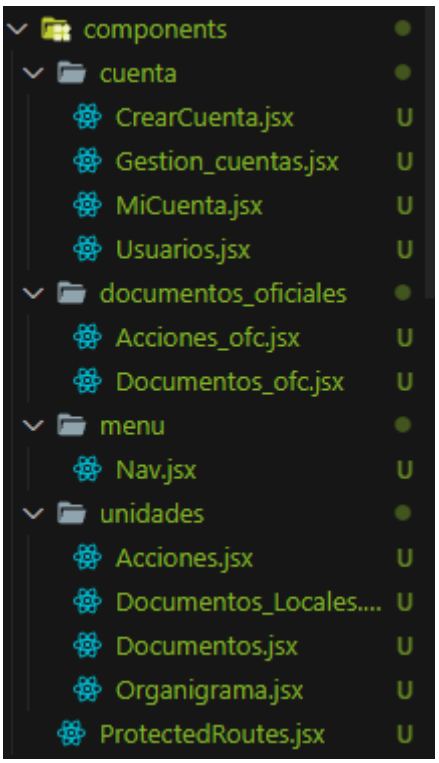

Figura 34. Componentes

- Los componentes dentro de la carpeta cuenta están relacionados a la cuenta de usuarios.
- Los componentes dentro de la carpeta documentos_oficiales están relacionados a los documentos oficiales.
- Los componentes dentro de la carpeta menú están relacionados con la barra de navegación del sistema.
- Los componentes dentro de la carpeta unidades están relacionados con la sección de unidades del sistema.
- El componente ProtectedRoutes.jsx es el encargado de proteger la ruta y que estas solo se muestran si el usuario ha sido autenticado.

6.1.2 Contextos

El contexto es un elemento que podemos crear en React para establecer una comunicación directa entre un componente en un nivel muy alto y uno en un nivel mucho más bajo.
A continuación se muestran los contextos creados para el proyecto:

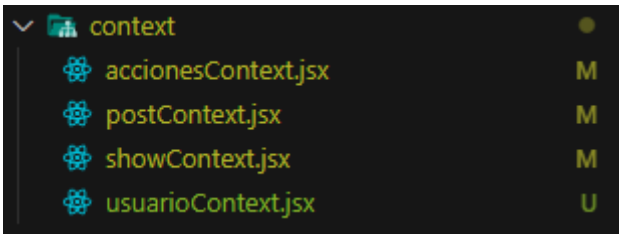

Figura 35. Contextos

- accionesContext.jsx: Está encargado de las funcionalidades de las acciones que pueden realizar los usuarios.

- postContex.jsx: Está encargado de las funcionalidades de los documentos y carpetas.
- showContext.jsc: Está encargado de la activación y desactivación de componentes, mostrando los componentes cuando estos están activos.
- usuarioContext: Está encargado de las funcionalidades de los usuarios.

6.1.3 Pages

Las páginas son elementos que se mostrarán dependiendo de la ruta en que se encuentre, estos están creados a partir de componentes.
A continuación se muestran las páginas y las rutas en cual estan estas paginas:

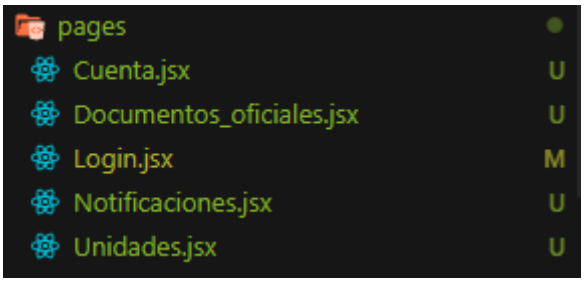

Figura 36. Pages

```jsx
function App() {
  return (
    <UsuarioProvider>
      <PostProvider>
        <ShowProvider>
          <AccionesProvider>
            <Routes>
              <Route path="" element={<Login />}></Route>
              <Route element={<ProtectedRoutes />}>
                <Route path="/menu" element={<Nav />}></Route>
                <Route path="/unidades" element={<Unidades />}></Route>
                <Route
                  path="/documentos_oficiales"
                  element={<Documentos_oficiales />}
                ></Route>
                <Route
                  path="/notificaciones"
                  element={<Notificaciones />}
                ></Route>
                <Route path="/cuenta" element={<Cuenta />}></Route>
              </Route>

              <Route
                path="*"
                element={<div>No se encontro la pagina</div>}
              ></Route>
            </Routes>
            <Toaster />
          </AccionesProvider>
        </ShowProvider>
      </PostProvider>
    </UsuarioProvider>
  );
}
```

Figura 37. App.jsx: Rutas de páginas

Como se puede observar el componente ProtectedRoutes está protegiendo la ruta de todas las páginas excepto la del Login, esto porque esa es la página que se usa para autenticar el usuario y permitirá al usuario acceder a las demás páginas del sistema.

6.1.4 Login

Como se puede observar en la figura 38, esta es la vista del sistema para que el usuario inicie su sesión, ingresando su usuario y contraseña.

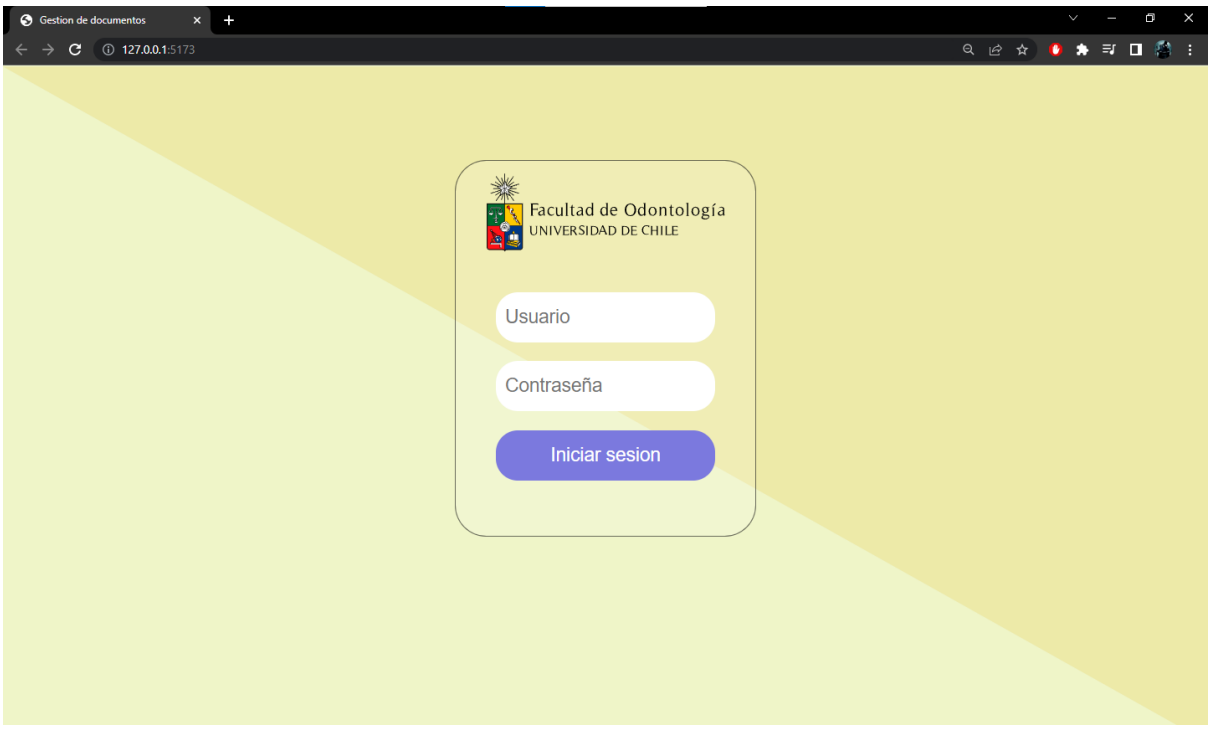

Figura 38. Login

6.1.5 Home

Como se puede observar en la figura 39, esta es la vista del sistema una vez el usuario ya haya sido autenticado, tendrá una barra de navegación en el lateral izquierdo que podrá usar para dirigirse a las secciones.

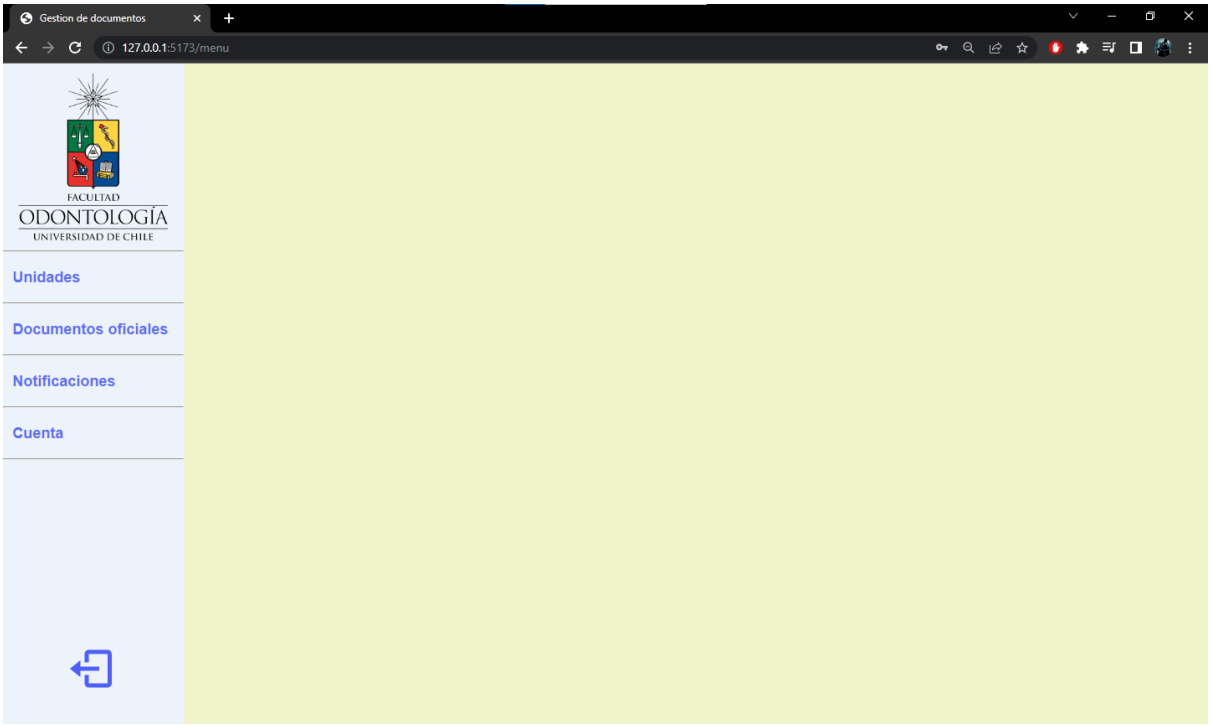

Figura 39. Home

6.1.6 Unidades

Como se puede observar en la figura 40, esta es la vista del sistema para la sección de las unidades, en específico “Mis unidades”, se muestran las carpetas y archivos dentro de la unidad 1(Podremos retroceder carpetas seleccionando el historial de rutas de la parte superior), las acciones que se puede realizar y las opciones de subir un documento o crear una carpeta en la parte inferior, al momento de crear una carpeta podremos seleccionar los usuarios permitidos a la carpeta a crear.

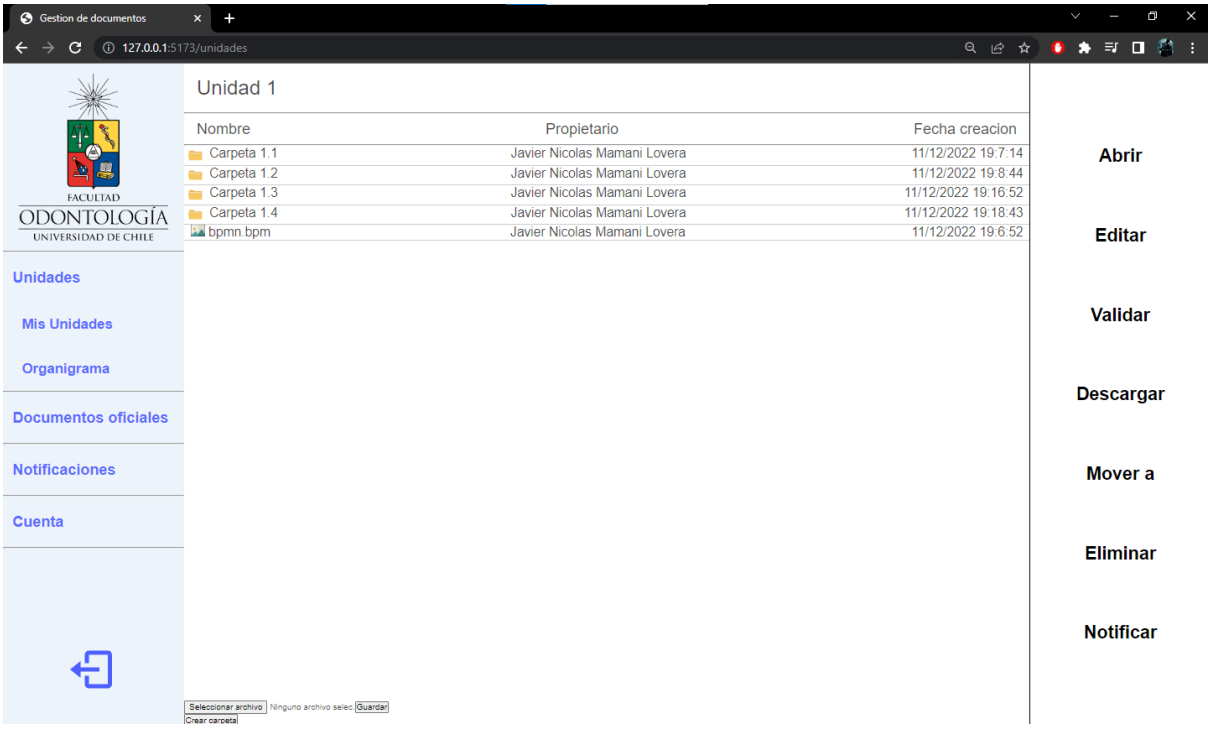

Figura 40. Mis unidades

6.1.7 Documentos oficiales

Como se puede observar en la figura 41, esta es la vista del sistema para la sección de los documentos oficiales, esta vista es bastante parecida a la de Mis unidades con la diferencia de que aca solo se encontraran documentos validados por la decana (usuario de tipo admin).

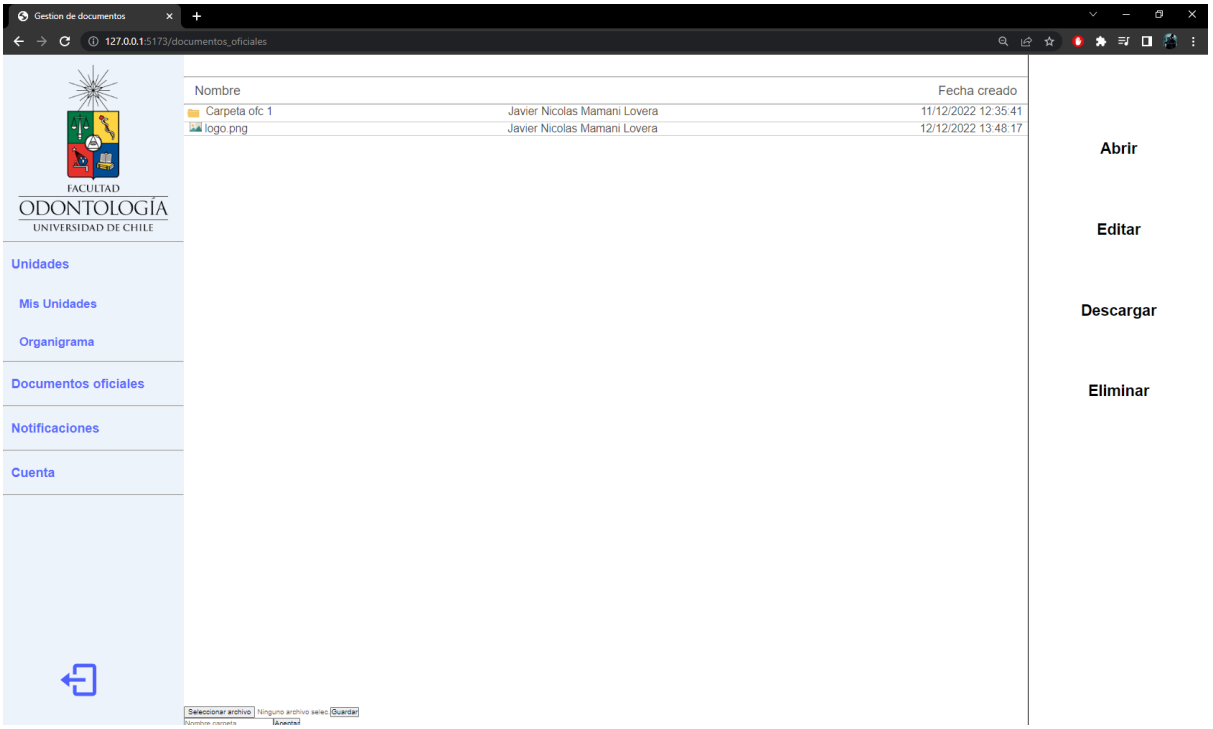

Figura 41. Documentos oficiales

6.1.8 Cuenta

6.1.8.1 Mi cuenta

Como se puede observar en la figura 42, esta es la vista de los datos de la cuenta en la que se encuentra el usuario.

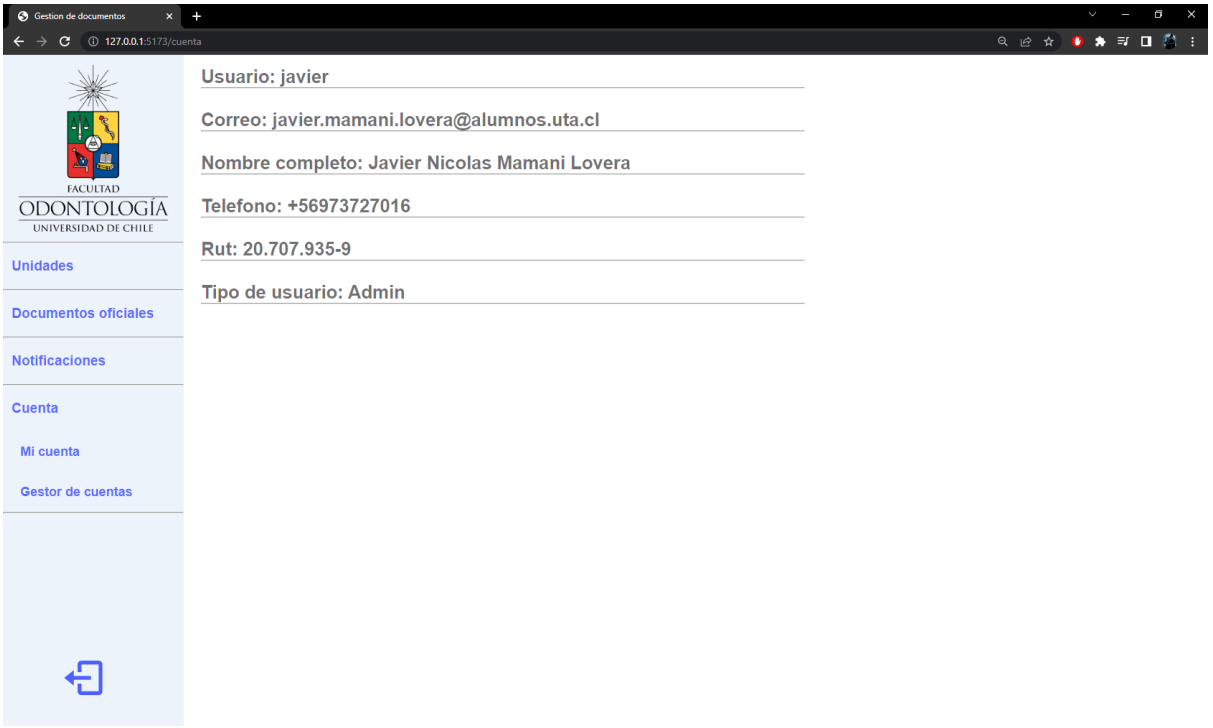

Figura 42. Mi cuenta

6.1.8.2 Gestor de cuentas

A continuación se muestran las vistas del sistema para el gestor de cuentas, cabe destacar que estas vistas solo pueden ser accedidas por usuarios de tipo admin, se tienen dos apartados, uno de crear cuenta figura 43 y otra de edición o eliminación de cuentas figura 44.

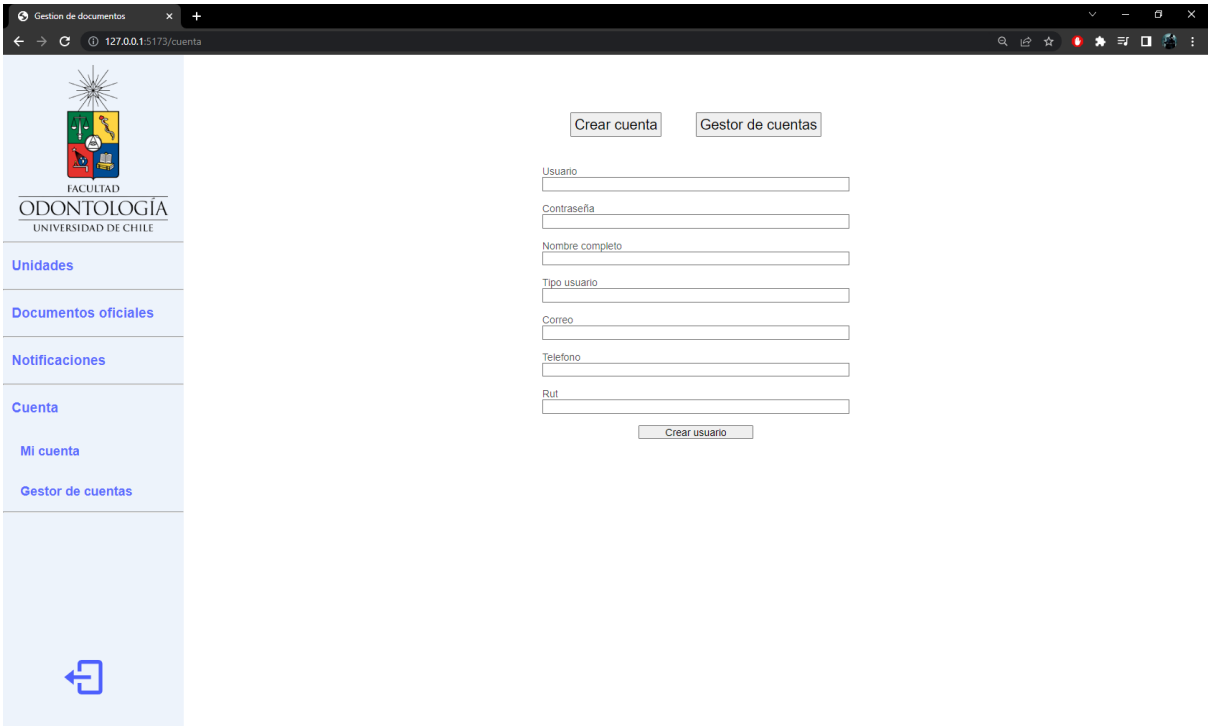

Figura 43. Crear cuentas

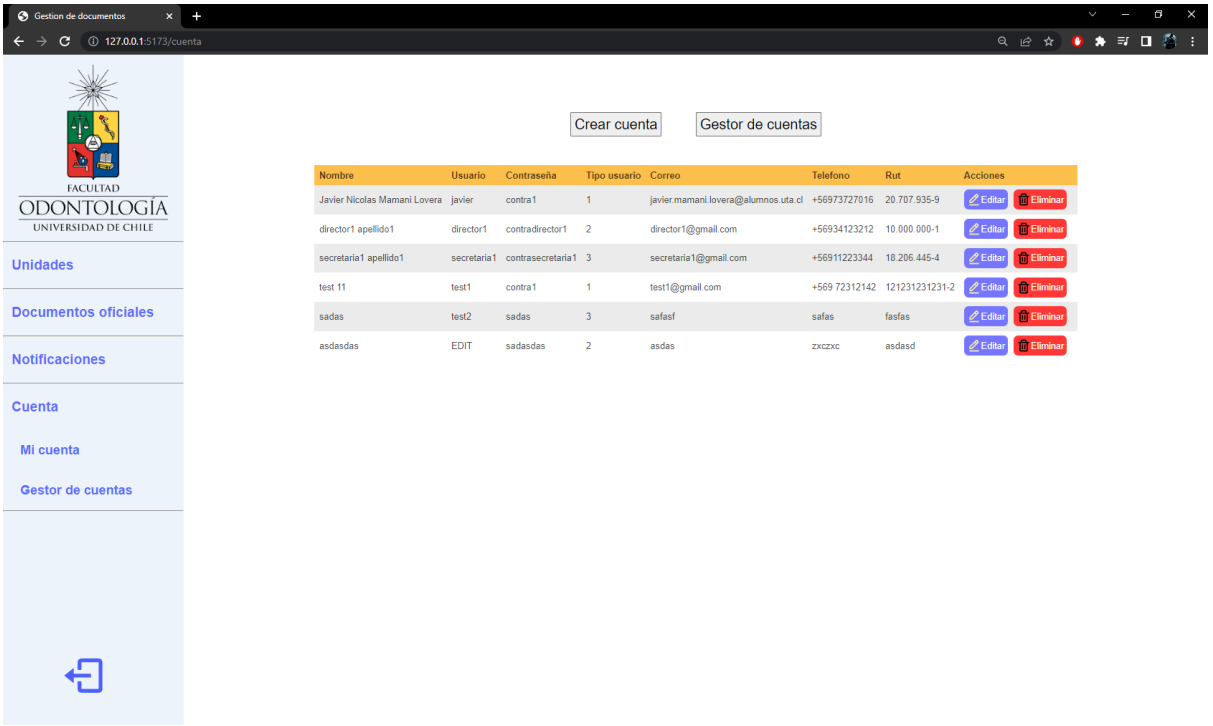

Figura 44.Gestor de cuentas

6.2 Backend

Dado que se utilizó node como framework para el backend, partes importantes que se utilizan son las rutas y controladores, se explicara detalladamente estos en sus respectivas secciones.

6.2.1 Rutas

A continuación se muestran todas las rutas del servidor, las funciones que ejecutan las rutas se mostrarán en la sección de controladores.

```
router.post("/login", login); //Verificar que exista la cuenta de usuario y la devuelve
router.post("/createUser", createUser); //Crear usuario
router.get("/users", getUsers); //Obtener usuarios
router.delete("/users/:id", deleteUser); //Eliminar un usuario
router.post("/users/edit", editUser); //Editar un usuario

router.get("/posts", getPosts); //Obtener documentos
router.post("/posts", createPosts); //Crear documento
router.put("/posts/:id", uptdatePosts); //Actualizar documento
router.delete("/posts/:id", deletePosts); //Elimina el documento mediante su id
router.get("/download/:id", getPost); //Descargar documento
router.post("/posts/send", sendPost); //Mover documento
router.post("/posts/validate", validatePost); //Validar documento
router.post("/posts/oficial", oficialPost); //Validar documento volviendolo oficial

router.post("/folders", createFolders); //Crear carpeta
router.get("/folders", getFolders); //Obtener carpetas
router.delete("/folders/:id", deleteFolder); //Eliminar carpeta mediante su id
router.post("/folders/permisos", createPermisosFolders); //Crear permisos de usuario y carpetas
router.get("/folders/permisos", getPermisosFolders); //Obtener permisos de usuario y carpetas
```

Figura 45. Rutas del servidor

6.2.2 Controladores

Los controladores son las funciones que se ejecutan al hacer una petición a alguna ruta del servidor, a continuación se muestra la carpeta de controladores del proyecto.

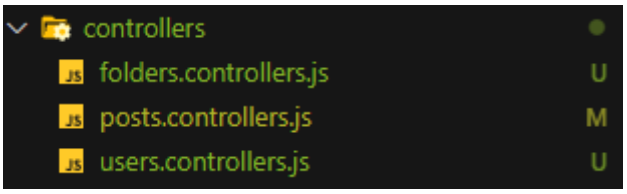

Figura 46. Controladores

- folders.controllers.js: Es la encargada de los controladores relacionados a las carpetas.
- posts.controllers.js: Es la encargada de los controladores relacionados a los documentos.
- users.controllers.js: Es la encargada de los controladores relacionados a los usuarios.

A continuación se muestra un ejemplo de un controlador, esta función está encargada de eliminar una carpeta y todas las subcarpetas y documentos que tenga dentro.

```
export const deleteFolder = async (req, res) => {
  //Llega el id de la carpeta como params.id
  try {
    let eliminarCarpetas = async (fk_id) => {
      const [carpetas] = await pool.query(
        "SELECT * FROM carpetas WHERE fk_carpeta= (?)",
        [fk_id]
      );
      //console.log(carpetas.length);
      if (carpetas.length > 0) {
        for (const carpeta of carpetas) {
          await eliminarCarpetas(carpeta.id);
        }
      }
      //console.log("Se esta elminando la carpeta", fk_id);
      const [result2] = await pool.query(
        "SELECT * FROM documentos WHERE fk_carpeta = (?)",
        [fk_id]
      );
      await pool.query("DELETE FROM documentos WHERE fk_carpeta=?", [fk_id]);
      for (const documento of result2) {
        //console.log(documento.path_documento);
        await fs.remove(documento.path_documento);
      }
      const [result] = await pool.query("DELETE FROM carpetas WHERE id=?", [
        fk_id,
      ]);
      //console.log("Se elimino la carpeta: ", fk_id);
      if (result.affectedRows === 0) {
        return res
          .status(404)
          .json({ message: "Carpeta no encontrada para eliminar" });
      }
    };
    await eliminarCarpetas(req.params.id);
    return res.sendStatus(204);
  } catch (error) {
    return res.status(500).json({ message: error.message });
  }
};
```

Figura 47. Ejemplo controlador deleteFolder

6.2.3 Repositorio de documentos

Los documentos subidos al sistema se almacenan en la carpeta “subido” que se encuentra en el servidor, para la realización de esto se usó el paquete multer.

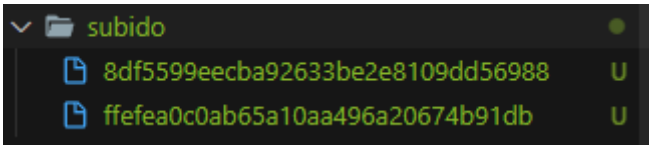

Figura 48. Carpeta subido

server > index.js > ...

```
import express from "express";
import postRoutes from "./routes/posts.routes.js";
import cors from "cors";
import multer from "multer";
import { Server as SocketServer } from "socket.io";
import http from "http";

const app = express(); //aplicacion servidor de express
const server = http.createServer(app); //Convertirlo a servidor http
const io = new SocketServer(server, {
  cors: {
    origin: "*",
  },
}); //Se crea el server socket pasandole el servidor http antes creado y configurando las entreadas origin: "http://127.0.0.1:5173"
const port = 4000;
//middlewares
app.use(express.json()); //procesar los datos del cliente (si es json)
app.use(cors()); //areggla el error de seguridad cors, se puede especificar que paginas queremos darle permisos
app.use(multer({ dest: "./subido" }).single("archivo"));
//rutas
app.use(postRoutes);
//io
io.on("connection", (socket) => {
  console.log("Se conecto socket.io con el usuario", socket.id);
  socket.on("message", (message) => {
    console.log(message);
  });
});

server.listen(port);
console.log("Server in running port:", port);

```

Figura 49. index.js del servidor

6.2.4 Conexión con el frontend

Para la conexión del frontend con el backend se utilizó el paquete axios para realizar las peticiones al servidor, esto se muestra en la siguiente imagen.

Figura 50. Carpeta api

client > src > api > posts.js > getFoldersRequests

```
import axios from "axios";

export const getPostsRequests = async () =>
  await axios.get("http://localhost:4000/posts"); //esto es asi por el proxy, sino tendria que ser http://localhost:4000/
  posts
export const createPostRequests = async (post) => {
  const form = new FormData();
  for (let key in post) {
    form.append(key, post[key]);
  }
  return await axios.post("http://localhost:4000/posts", form, {
    headers: { "Content-Type": "multipart/form-data" },
  });
};
export const deletePostRequests = async (id) =>
  await axios.delete("http://localhost:4000/posts/" + id);
export const downloadPostRequests = async (id) =>
  await axios.get("http://localhost:4000/download/" + id);
export const sendPostRequests = async (datos) =>
  await axios.post("http://localhost:4000/posts/send", datos); //datos tiene id y fk_carpeta
export const validatePostRequests = async (id) =>
  await axios.post("http://localhost:4000/posts/validate", id);
export const oficialPostRequests = async (id) =>
  await axios.post("http://localhost:4000/posts/oficial", id);

//CARPETAS
export const getFoldersRequests = async () =>
  await axios.get("http://localhost:4000/folders");
export const createFoldersRequests = async (folder) =>
  await axios.post("http://localhost:4000/folders", folder);
export const deleteFolderRequests = async (id) =>
  await axios.delete("http://localhost:4000/folders/" + id);
export const getPermisosFoldersRequests = async () =>
  await axios.get("http://localhost:4000/folders/permisos");
export const createPermisosFoldersRequests = async (datos) =>
```

Figura 51. api/posts.js

7. Pruebas

En esta sección se mostrarán algunas pruebas del sistema, cabe recalcar que el sistema es funcional y se está utilizando de manera local.

7.1 Documentos

7.1.1 Subir documento

Para subir un documento debemos de entrar a alguna carpeta del sistema como se muestra en la siguiente imagen:

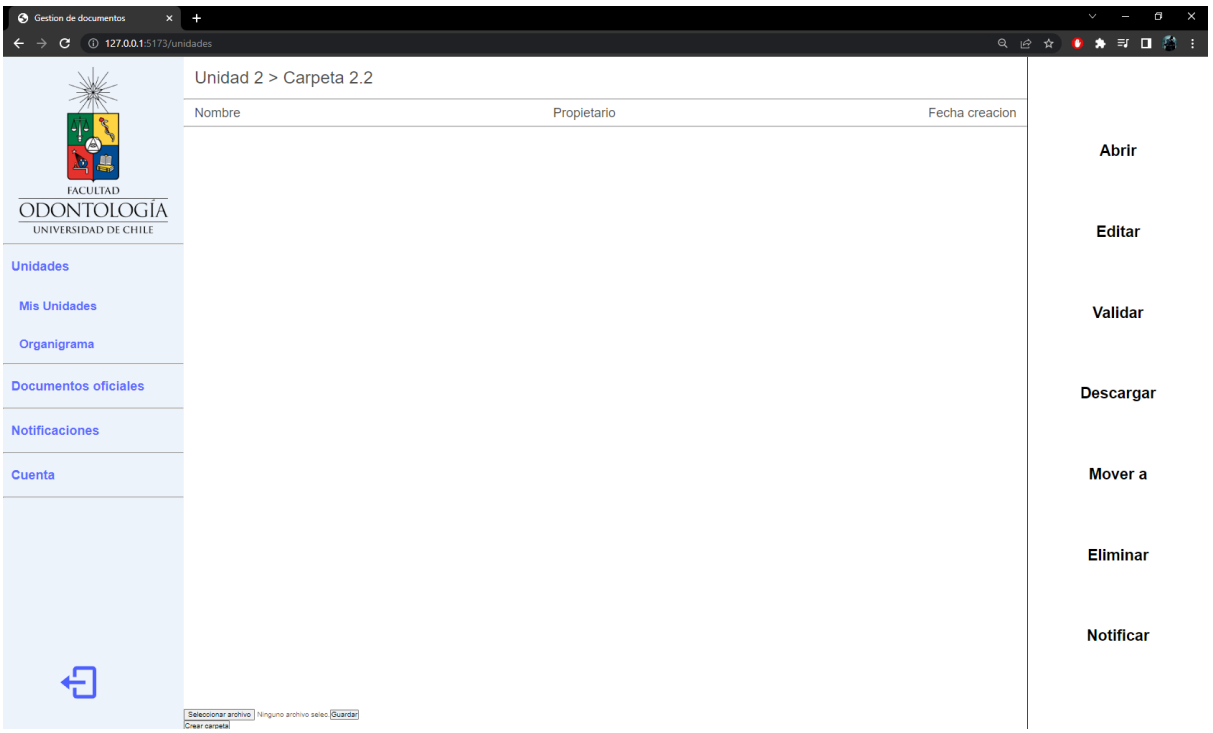

Figura 52. Subir documento: 1

Luego se debe de presionar “Seleccionar archivo” de la parte inferior, esto abrirá el explorador de archivos del computador, debemos de seleccionar el archivo a subir como se muestra en la siguiente imagen:

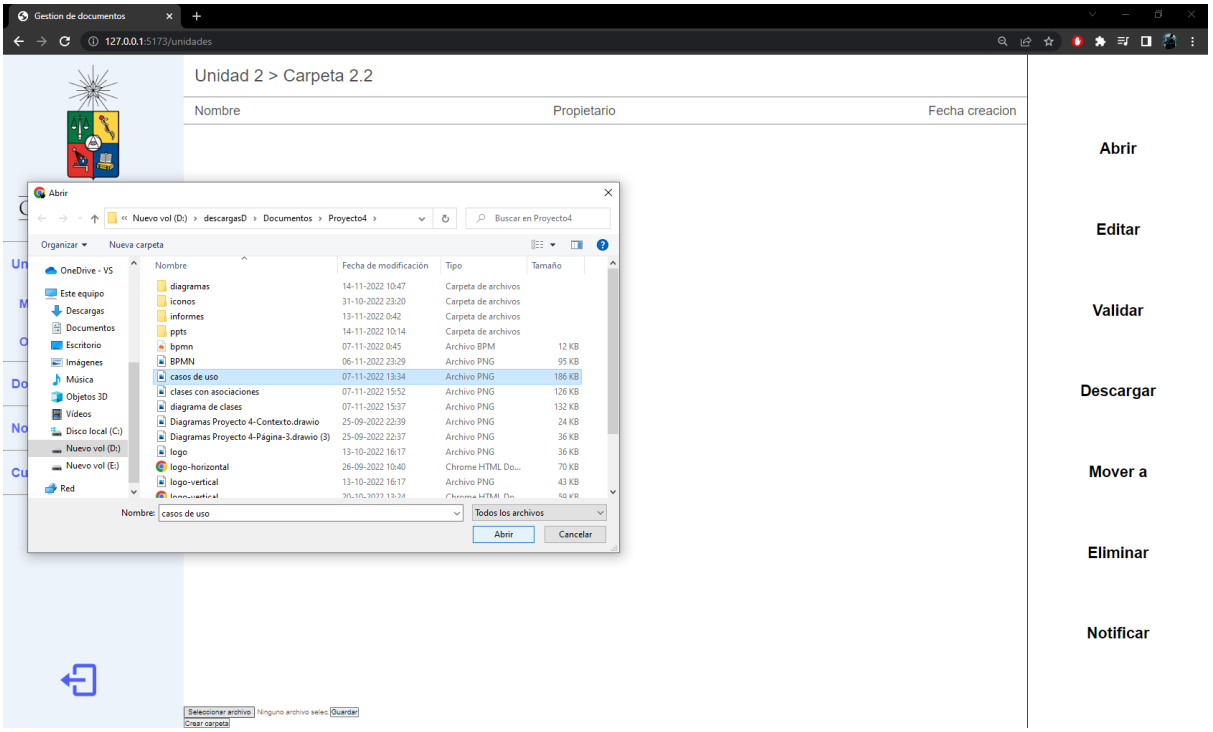

Figura 53. Subir documento: 2

Presionamos Abrir en el explorador de archivos y Guardar en el sistema, esto guardará en el sistema los datos del documento (en la carpeta actual) y en el servidor el documento en sí, esto se ve en la siguiente imagen:

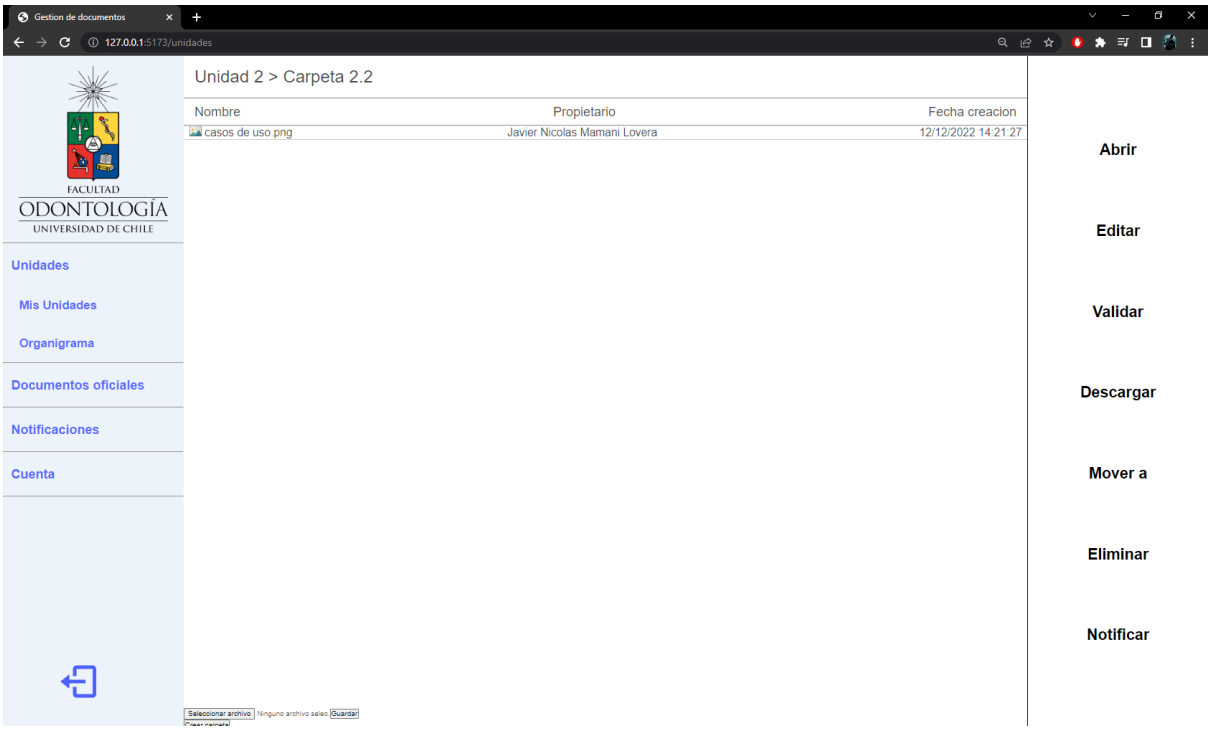

Figura 54. Subir documento: 3

7.1.2 Bajar documento

Para bajar un documento debemos de seleccionar un documento y presionar “Descargar” de la barra de acciones que se encuentra en el lateral derecho, luego de hacer esto el sistema buscará el documento y empezará la descarga, esto se muestra en la siguiente imagen:

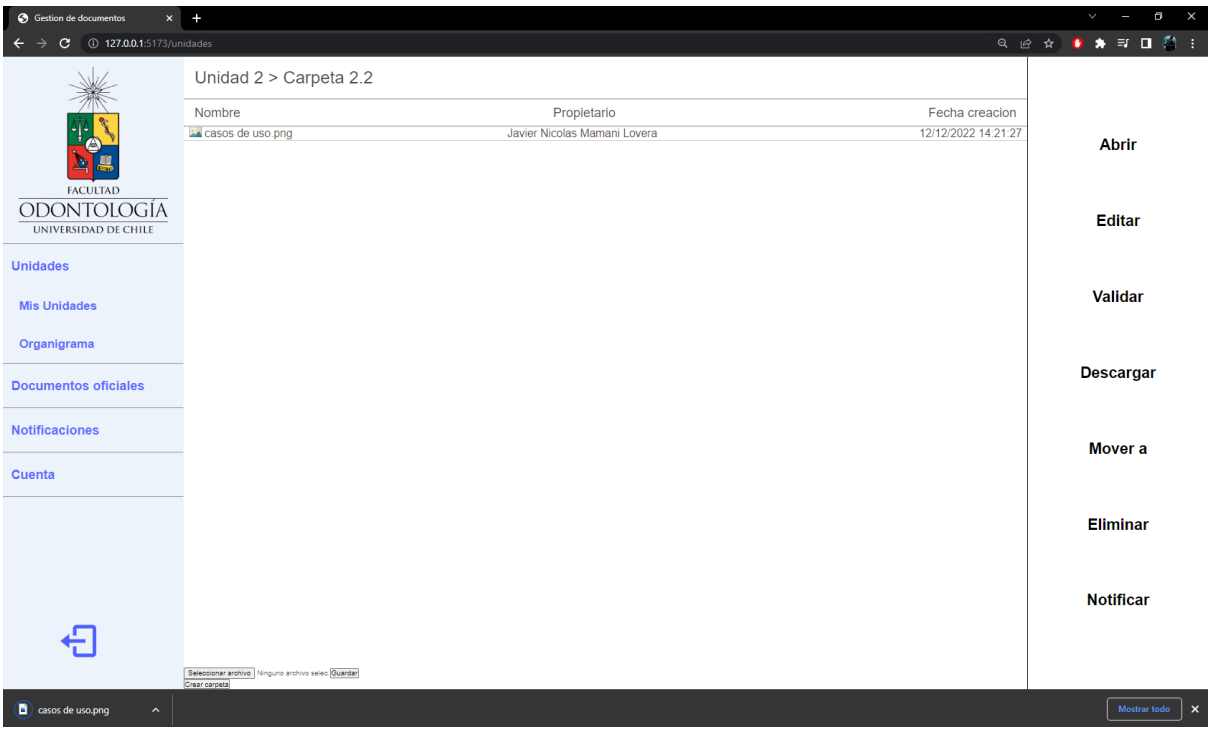

Figura 55. Bajar documento

7.1.3 Mover documento

Para mover un documentos debemos de seleccionar el documento y presionar “Mover a” de la barra de acciones, esto nos mostrará una notificación como se muestra en la siguiente imagen:

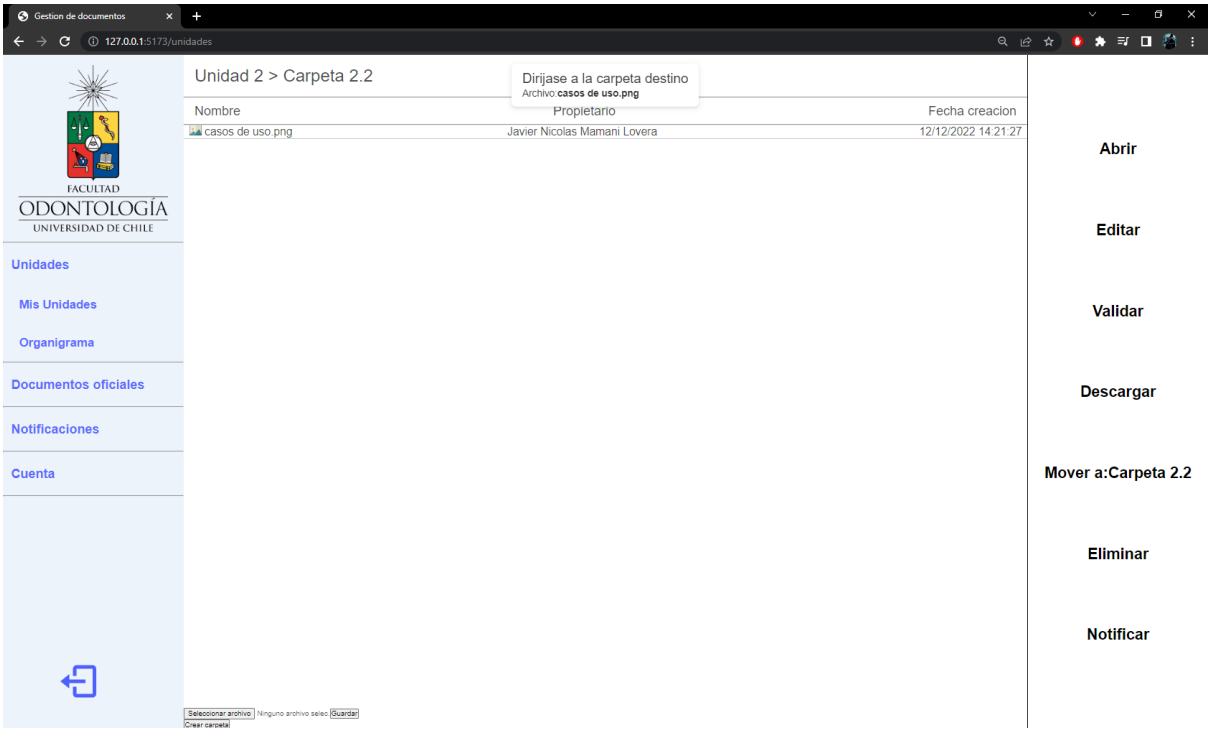

Figura 56. Mover documento: 1

Luego si cambiamos a otra carpeta y volvemos a presionar el mismo botón nos aparecerá lo que se muestra en la figura 57 y se habrá movido el documento.

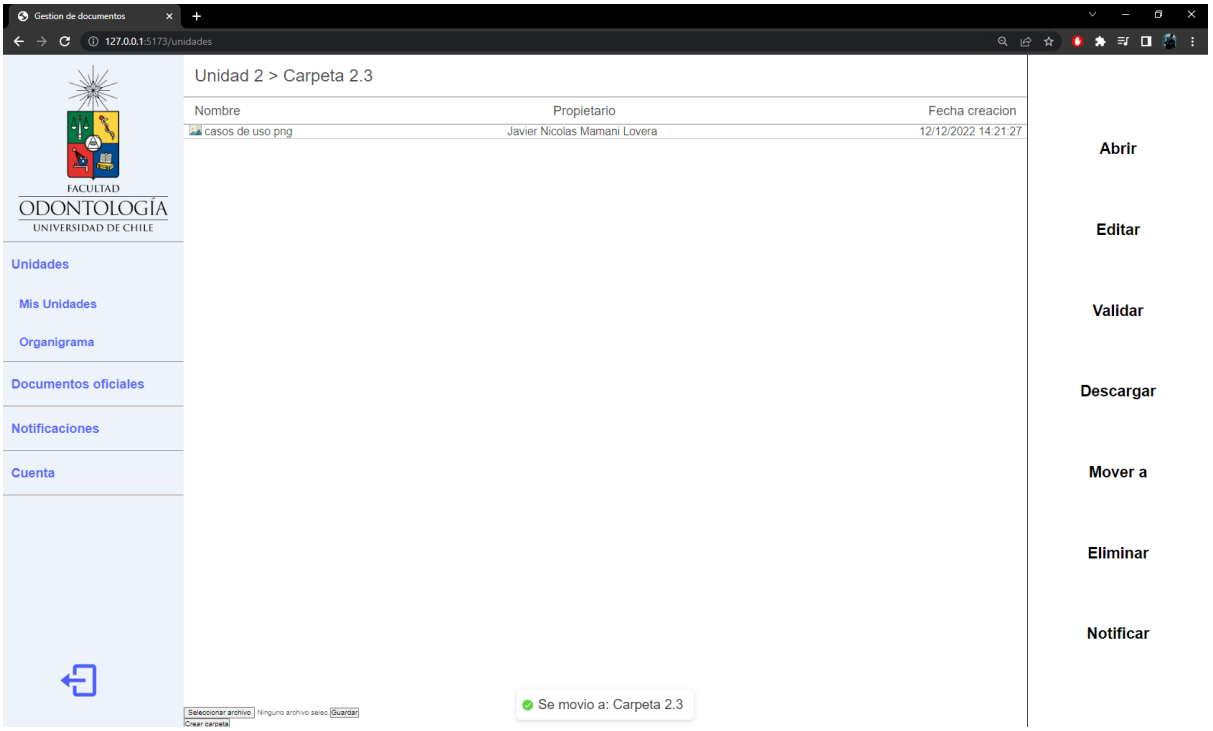

Figura 57. Mover documento: 2

7.2 Carpetas

7.2.1 Crear carpeta

Para crear una carpeta nos debemos de dirigir hacia alguna carpeta y presionar “Crear carpeta” de la parte inferior, esto nos mostrará lo siguiente:

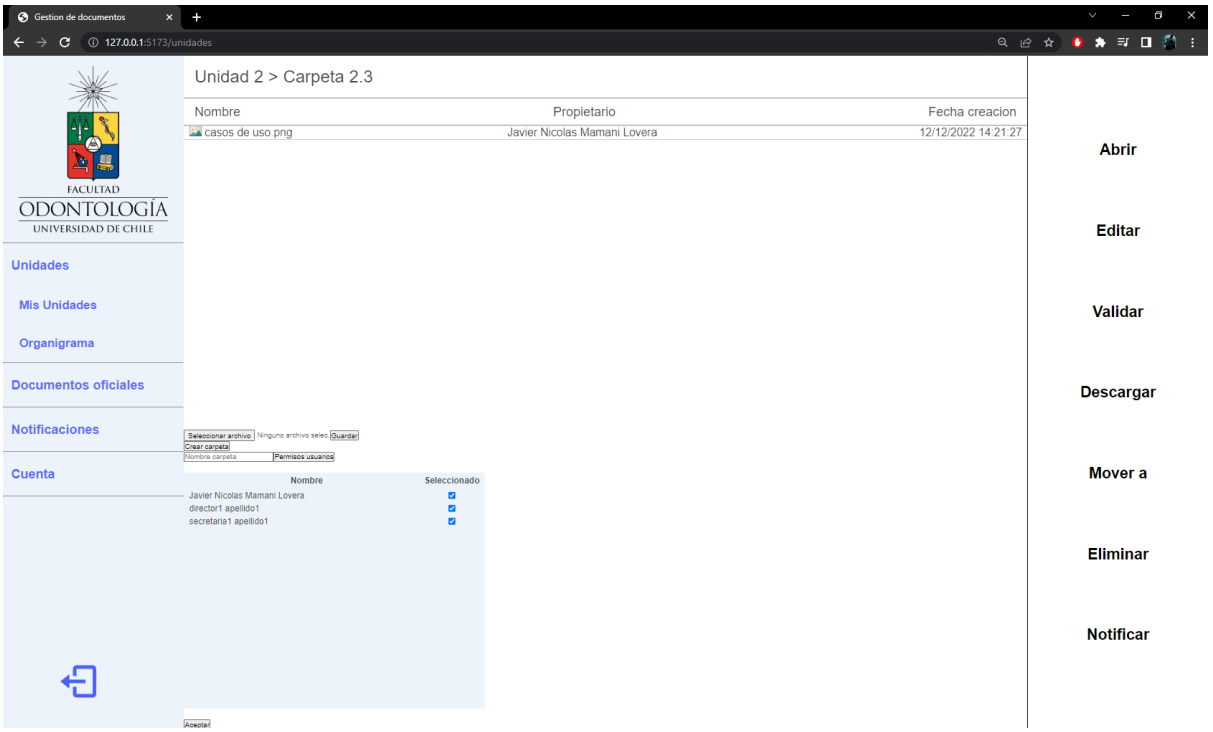

Figura 58. Crear carpeta

Donde podremos ingresar el nombre de la carpeta y los usuarios permitidos, para crear la carpeta presionamos aceptar.

7.2.2 Eliminar carpeta

Para eliminar una carpeta debemos de seleccionar la carpeta a eliminar y presionar el botón “Eliminar”, esto nos mostrará una notificación de confirmación como se muestra en la figura 59.

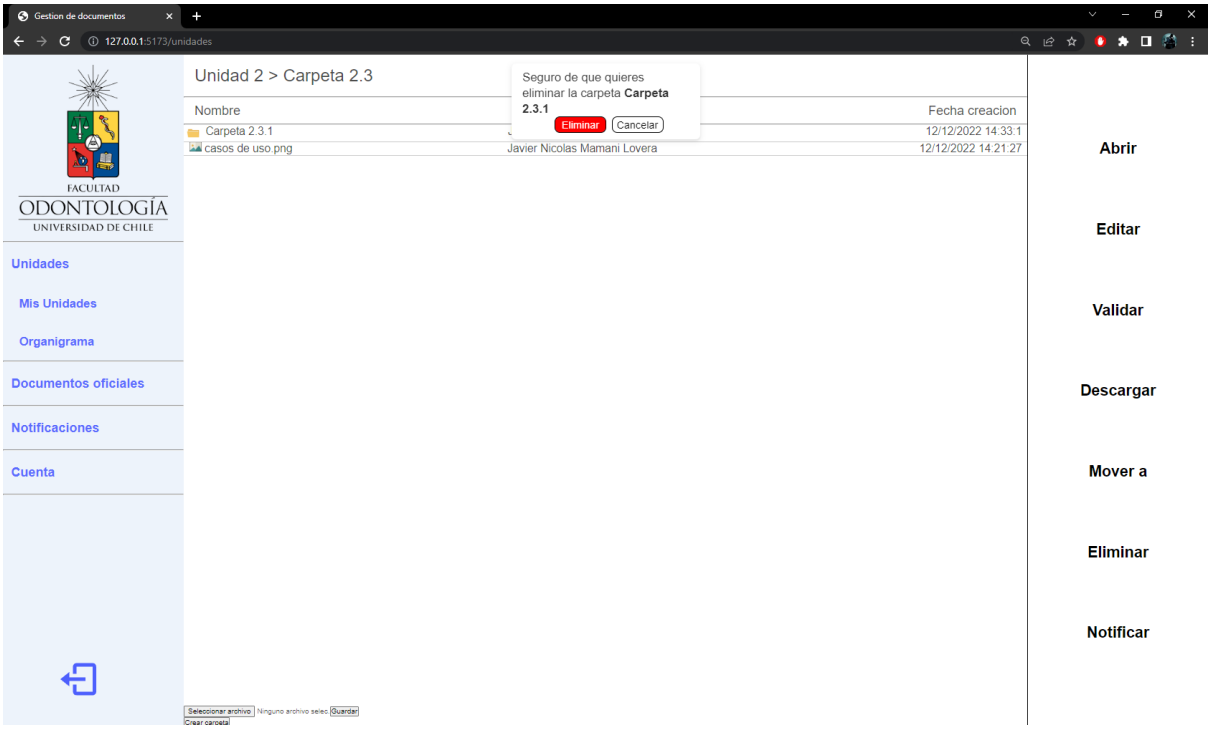

Figura 59. Eliminar carpeta

7.3 Usuarios

7.3.1 Crear usuario

Para crear usuarios se debe de estar en una cuenta de tipo admin, se debe de ir a la sección “Mi cuenta” y rellenar el formulario con los datos del usuario como se muestra en la figura 60.

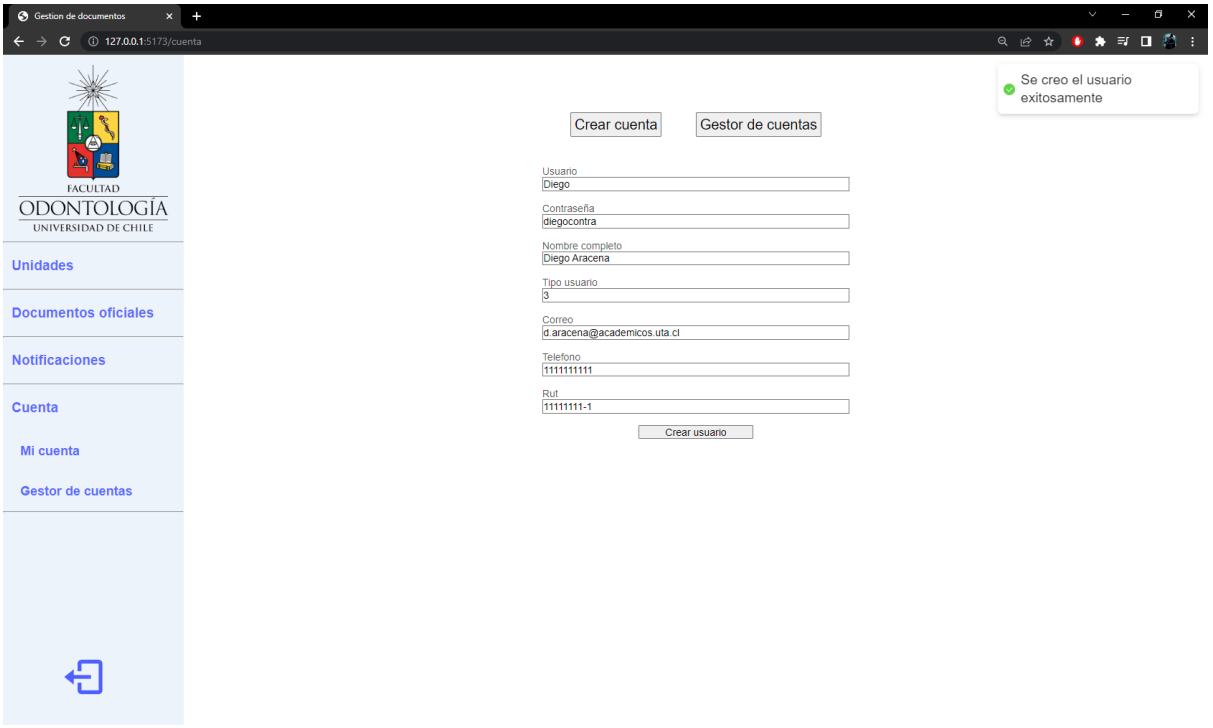

Figura 60. Crear usuario

8. Conclusión

En esta última etapa de avance del proyecto se continuó con el proyecto, metiéndose más a la programación del frontend y backend, realizando una extensa investigación acerca de los tópicos necesarios para el desarrollo exitoso del sistema, realizando pruebas y resolviendo errores que se ocasionaron.
Mediante el desarrollo del proyecto se obtuvo extenso conocimiento en el stack de tecnologías MERN, el cual está conformado de Mysql, Express, React y Node, profundizando en estas tecnologías y cómo funcionan en conjunto. Cabe recalcar que se tenía conocimiento muy pobre de estas tecnologías y se acabó con una base sólida de estas.

Por otro lado, la comunicación con el cliente fue buena, aunque con algunos problemas en el intercambio de información respecto al proyecto, esto afectó en la planificación del proyecto.
Por último, se logró un sistema funcional estable, pero no completando todos los requisitos del proyecto y diseño del mismo, estos por falta de tiempo y organización, además de que el proyecto en sí es bastante extenso teniendo en cuenta las demás actividades estudiantiles que se deben de cumplir y el trabajo individual, sin un equipo que apoye en el desarrollo del sistema, se planea completar el proyecto en su totalidad en posteriores periodos.

9. Referencias

[1] Facultad de Odontología Universidad de Chile: https://odontologia.uchile.cl/
[2] Patrón de diseño MVC: https://developer.mozilla.org/es/docs/Glossary/MVC
[3] Patrón de diseño MVVM: https://learn.microsoft.com/es-es/xamarin/xamarin-forms/enterprise-application-patterns/mvvm
[4] Patrón de diseño publicador suscriptor: https://learn.microsoft.com/es-es/azure/architecture/patterns/publisher-subscriber
[5] MongoDB: https://www.mongodb.com/
[6] ExpressJS: https://expressjs.com/es/
[7] Vue.js: https://vuejs.org/
[8] NodeJS: https://nodejs.org/es/