FACULTAD DE INGENIERÍAS Y MATEMÁTICAS

ESCUELA PROFESIONAL DE INGENIERÍA DE SOFTWARE

Rechts Schreibtisch

System Design Document (SDD)

Integrantes:

- Miguel Angel Flores Leon
- Abimael Ernesto Frontado Fajardo
- Christofer Alberto Gamio Huaman
- Jhordan Steven Octavio Huamani Huamani
- Jorge Luis Ortiz Castañeda
- Gian Piero Khalil Rodriguez Fadel

2024

INFORME DE CUMPLIMIENTO

Fecha: 25 – 11 - 2024

Integrantes:

Nombre y apellido	**Tareas a realizar**	**Porcentaje de cumplimiento**
Jorge Luis Ortiz Castañeda	Entorno Operacional, Aspectos de Calidad y de Comportamiento y Estructuración de Proyecto	100%
Jhordan Steven Octavio Huamani Huamani	Objetivos y diagramado arquitectura por capas. Elaboración de Ilustraciones.	100%
Miguel Angel Flores León	Objetivos y código ADL de la aplicación a realizar. Desarrollo de la lógica del negocio.	100%
Gian Piero Khalil Rodriguez Fadel	NULL (en c)	0%

Christofer Alberto Gamio Huaman	Nullptr (en c++)	0%
Abimael Ernesto Frontado Fajardo	Desarrollo de gráfico UML y elaboración de gráficos de arquitectura lógica.	100%

TABLA DE CONTENIDO

Tabla de Ilustraciones

Tabla de Tablas

1. Introducción

1.1. Propósito del Documento SDD

El propósito principal del documento de Diseño de Detalle del Sistema (SDD) es proporcionar una guía completa para el desarrollo e implementación del sistema integrado de Consafill. Este documento describirá de manera precisa los requisitos funcionales y no funcionales del sistema, la arquitectura propuesta, los módulos a desarrollar, las interfaces de usuario y las bases de datos que se utilizarán. En resumen, el SDD será la referencia principal para el equipo de desarrollo y garantizará que el sistema final cumpla con las expectativas de los usuarios.

Finalidad de la Elaboración del SDD

La elaboración del SDD tiene como finalidad:

- **Clarificar los requisitos:** Definir de manera clara y concisa los requerimientos del sistema, evitando ambigüedades y malentendidos.
- **Establecer un acuerdo:** Servir como un acuerdo entre el cliente (Consafil) y el equipo de desarrollo sobre las características y funcionalidades del sistema.
- **Guiar el desarrollo:** Proporcionar una guía detallada para el equipo de desarrollo, facilitando la planificación, el diseño y la implementación del sistema.
- **Reducir riesgos:** Identificar y mitigar los riesgos potenciales del proyecto, asegurando un desarrollo exitoso.
- **Facilitar la comunicación:** Servir como un punto de referencia para la comunicación entre todos los involucrados en el proyecto.

Propósito de la Solución

La solución propuesta tiene como objetivo principal automatizar y optimizar los procesos de gestión de documentos, clientes y expedientes de Consafill. Al centralizar la información en un sistema integrado, se busca mejorar la eficiencia, la precisión y la seguridad de las operaciones diarias del despacho de abogados.

Elementos Específicos de la Solución

1. **Gestión de Documentos**:
    - **Almacenamiento Centralizado**: Todos los documentos se almacenarán en una base de datos centralizada, accesible desde cualquier lugar.
    - **Búsqueda Avanzada**: Implementación de un motor de búsqueda que permita localizar documentos rápidamente mediante palabras clave, fechas, y otros criterios.
    - **Control de Versiones**: Seguimiento de las diferentes versiones de un documento para mantener un historial de cambios.
2. **Gestión de Clientes**:
    - **Base de Datos de Clientes**: Registro detallado de la información de cada cliente, incluyendo datos de contacto, historial de casos y comunicaciones.
    - **Seguimiento de Interacciones**: Registro de todas las interacciones con los clientes, facilitando un seguimiento eficiente y personalizado.
3. **Gestión de Expedientes**:
    - **Creación y Seguimiento de Expedientes**: Generación de expedientes electrónicos para cada caso, con la capacidad de adjuntar documentos, notas y actualizaciones.
    - **Alertas y Recordatorios**: Sistema de notificaciones para recordar fechas importantes y plazos de los casos.

4. **Seguridad y Acceso**:
    - **Autenticación y Autorización**: Implementación de un sistema de autenticación robusto para asegurar que solo el personal autorizado tenga acceso a la información sensible.
    - **Cifrado de Datos**: Uso de técnicas de cifrado para proteger los datos almacenados y transmitidos.
5. **Interfaz de Usuario**:
    - **Diseño Intuitivo**: Una interfaz de usuario amigable y fácil de usar que permita a los usuarios navegar y realizar tareas sin complicaciones.
    - **Acceso Multiplataforma**: La aplicación web será accesible desde diferentes dispositivos, incluyendo computadoras, tabletas y teléfonos móviles.
6. **Calendario Integrado**:
    - **Gestión de Citas y Plazos**: Un calendario integrado que permita la programación y seguimiento de citas, plazos y eventos importantes directamente desde la aplicación.
    - **Sincronización Automática**: Capacidad de sincronizar automáticamente con los expedientes y casos, generando recordatorios y alertas para fechas clave.

Puntos Claves del Diseño y Arquitectura del Sistema Propuesto

- **Modularidad:** El sistema se dividirá en módulos bien definidos (documentación, clientes, expedientes, contabilidad, facturación, respaldos) para facilitar el desarrollo, el mantenimiento y la escalabilidad.
- **Centralización de datos:** Toda la información se almacenará en una base de datos centralizada, lo que permitirá acceder a los datos desde cualquier lugar y en cualquier momento.

- **Flexibilidad:** El sistema será diseñado para adaptarse a las necesidades cambiantes de Consafill, permitiendo la adición de nuevas funcionalidades en el futuro.
- **Seguridad:** Se implementarán medidas de seguridad robustas para proteger la información confidencial de los clientes.
- **Usabilidad:** La interfaz de usuario será intuitiva y fácil de usar, permitiendo a los usuarios realizar sus tareas de manera eficiente.

Funcionalidades Claves Para Incluir o Mejorar

- Gestión de documentos: Almacenamiento, búsqueda, clasificación y versión de documentos.
- Gestión de clientes: Creación, actualización y segmentación de clientes.
- Gestión de expedientes: Creación, seguimiento, asignación de estado y cierre de expedientes.
- Gestión de calendarios: Programación de citas y reuniones.

1.2. Audiencia

Este documento ha sido diseñado principalmente para ser comprendido y utilizado por abogados y personal jurídico que, si bien son expertos en derecho, pueden tener un conocimiento técnico limitado. El objetivo es proporcionar una descripción clara y concisa de las funcionalidades del sistema, evitando tecnicismos innecesarios y utilizando un lenguaje accesible.

Consideraciones para la audiencia:

- **Enfoque en los beneficios:** El SDD se centrará en cómo el sistema resolverá problemas específicos y mejorará la eficiencia en las tareas diarias de los abogados.
- **Lenguaje claro y conciso:** Se evitarán términos técnicos y se utilizarán ejemplos prácticos para ilustrar las funcionalidades del sistema.

- **Documentación visual:** Se incluirán diagramas de flujo, maquetas de pantalla y otros elementos visuales para facilitar la comprensión.

Supuestos sobre el Conocimiento Técnico

Se asume que la audiencia:

- **Conoce los procesos legales:** Está familiarizada con los procedimientos legales y la terminología jurídica.
- **Utiliza herramientas informáticas básicas:** Domina el uso de computadoras, procesadores de texto y correo electrónico.

1.3. Resumen Ejecutivo

1.3.1. Descripción General del Sistema

Descripción de Sistemas o Soluciones Similares al de la Propuesta

- **Clio**: Un software de gestión para abogados que ofrece características como administración de casos, gestión de documentos, facturación, y comunicación con clientes. Clio proporciona una plataforma basada en la nube que centraliza toda la información del caso en un solo lugar.
- **PracticePanther**: Otra solución de gestión legal que incluye herramientas para la administración de casos, seguimiento del tiempo, facturación, y gestión de documentos. Ofrece integraciones con diversas aplicaciones y una interfaz amigable para los usuarios.
- **MyCase**: Un software diseñado para gestionar casos legales, que incluye gestión de documentos, comunicación con clientes, calendarios, y tareas. Proporciona un entorno integrado para mejorar la eficiencia y la organización de los abogados.

Objetivos del Proyecto

- **Centralizar la Gestión Documental**: Crear un sistema que permita almacenar, organizar, y gestionar todos los documentos legales y casos en un único lugar, mejorando la accesibilidad y la eficiencia.
- **Automatizar el Proceso de Gestión de Casos**: Reducir la dependencia de herramientas manuales como Excel y Google Drive mediante la automatización de la creación, seguimiento y almacenamiento de documentos y casos.
- **Facilitar la Capacitación y Adaptación del Personal**: Diseñar el sistema de manera que sea intuitivo y fácil de usar, con capacitación adecuada para garantizar una transición fluida para el personal.

Módulos Principales del Sistema

1. **Documentación**:
    - **Gestión de Documentos**: Creación, edición, almacenamiento, y organización de documentos legales.
    - **Etiquetado y Catalogación**: Implementación de etiquetas y categorías para facilitar la búsqueda y organización de documentos por materias y estado del proceso.
2. **Clientes**:
    - **Almacenamiento de Datos de Clientes**: Registro y gestión de información de clientes y casos.
    - **Seguimiento de Casos**: Creación y administración de expedientes y consultas, incluyendo la actualización del estado de los casos.
3. **Expedientes**:

- **Creación y Gestión de Expedientes**: Integración de todos los documentos y datos relevantes en un expediente centralizado para cada caso.
- **Historial de Modificaciones**: Registro de cambios y actualizaciones en los expedientes para mantener un historial completo.

4. **Contabilidad y Facturación (Opcional)**:
   - **Gestión Financiera**: Administración de facturación y contabilidad si se decide incluir esta funcionalidad.
5. **Respaldos**:
   - **Sistema de Respaldo**: Implementación de una solución de respaldo automático y seguro para proteger todos los documentos y datos.

Tipos de Usuario del Sistema

Administradores (Actor 1)

Responsabilidades: Configuración del sistema, gestión de usuarios, configuración de permisos, y supervisión de la integridad de los datos.

Abogados (Actor 2)

Responsabilidades: Creación y gestión de casos, manejo de documentos legales, seguimiento de expedientes, y comunicación con clientes.

1.3.2. Restricciones de Diseño

Restricciones Financieras

Nuestro capital inicial sería de S/10,001. En cuento a los costos operativos, costos de hardware, red o allegados. Esta cantidad limita la complejidad de las características a implementar, así como los recursos asignados para su mantenimiento.

Restricciones Técnicas

El equipo cuenta con acceso a los siguientes recursos computacionales:

- **Hardware:** 3 computadoras de escritorio, 6 laptops.

Restricciones Organizativas

El equipo de desarrollo está compuesto por estudiantes de ingeniería de software con una sólida formación en los fundamentos de la programación y el desarrollo de software. Contamos con experiencia en:

- **Lenguajes de programación:** C++, Python, Kotlin, JavaScript.
- **Bases de datos:** SQL (MongoDB), NoSQL (FireBase).
- **Frameworks y herramientas:** React, Angular, Git.

Toda la información contenida en este documento, así como los detalles del proyecto, se consideran confidenciales. Se tomarán las medidas necesarias para proteger la información y evitar su divulgación a terceros no autorizados.

1.3.3. Dependencias y Limitaciones

Dependencias Funcionales

A1. Gestión de Archivos

- **Organización de Archivos:** Capacidad para crear, mover y organizar archivos en carpetas jerárquicas.
- **Carga y Descarga de Archivos:** Soporte para subir, descargar y eliminar archivos desde la interfaz web.
- **Etiquetado de Archivos:** Permitir la asignación de etiquetas a los archivos para facilitar la búsqueda y organización.

A2. Búsqueda y Filtrado

- **Búsqueda por Etiquetas:** Implementación de una función de búsqueda que permita encontrar archivos mediante etiquetas asignadas.
- **Filtrado por Estado:** Opciones para filtrar archivos según su estado (Culminado, Stand By, Urgente, En Plazo).

A3. **Calendario y Programación**

- **Gestión de Eventos:** Funcionalidad para crear, editar y eliminar eventos en un calendario.
- **Recordatorios y Notificaciones:** Envío de recordatorios y notificaciones sobre eventos y reuniones.

A4. Accesibilidad y Seguridad

- **Autenticación de Usuarios:** Implementación de un sistema de autenticación para asegurar que solo los usuarios autorizados accedan a la aplicación.
- **Control de Accesos y Permisos:** Gestión de roles y permisos para diferentes usuarios (administradores, abogados, asistentes).
- **Encriptación de Datos:** Protección de datos sensibles mediante encriptación en reposo y en tránsito.

A5. Interfaz y Experiencia de Usuario

- **Diseño Adaptativo:** Interfaz que se adapta a diferentes tamaños de pantalla y dispositivos (móviles, tablets, escritorios).
- **Experiencia Táctil:** Optimización para una navegación intuitiva en dispositivos móviles con pantallas táctiles.

Dependencias Técnicas

1. **Dependencias de almacenamiento de archivos:**

Servicio de almacenamiento en la nube o en el servidor local: Para guardar, organizar y administrar archivos. Un sistema de archivos distribuido o un servicio de almacenamiento en la nube (como Amazon S3, Google Cloud Storage, o una solución interna).

Sistemas de archivos organizados: Se necesitará un esquema de carpetas o directorio para organizar y guardar los archivos.

Clasificación y búsqueda: Necesitará una base de datos o un servicio especializado para indexar los archivos y las etiquetas asociadas para usar el etiquetado y la búsqueda por etiquetas. Para búsquedas rápidas y filtradas, pueden ser útiles tecnologías como Elasticsearch.

2. **Dependencias de autenticación y control de acceso:**

- **Autenticación y autorización de usuarios:** Para controlar quién puede acceder a la aplicación, se requiere un sistema de autenticación robusto, como OAuth, SSO o un sistema de autenticación propio, ya que no está disponible al público en general.
- **Roles de usuario y permisos:** Los usuarios deben tener permisos distintos según su posición (abogados, administrativos, etc.) para acceder a carpetas específicas.
- **Seguridad de los datos y encriptación:** Los archivos y la información confidencial deben estar encriptados tanto cuando están en reposo como cuando están en tránsito. Será crucial utilizar HTTPS para las conexiones y mecanismos como la encriptación AES para el almacenamiento de archivos.

3. **Dependencias de chat y mensajería interna:**
    - **Bases de datos para almacenar conversaciones:** Las conversaciones entre los trabajadores deben ser almacenadas de manera segura y organizada, lo que requerirá una base de datos que soporte almacenamiento y búsqueda rápida (SQL, NoSQL).
4. **Dependencias del calendario y webmail:**
    - **Calendario:** Podrías usar bibliotecas o APIs como Google Calendar API, o implementar una solución personalizada para gestionar eventos, citas y recordatorios.
    - **Webmail:** Dependiendo del sistema de correo utilizado por el despacho, podrías integrar un sistema de webmail (por ejemplo, IMAP/SMTP) o crear una interfaz personalizada para gestionar correos electrónicos internos.
5. **Dependencias del sistema de etiquetas y filtrado de archivos:**
    - **Base de datos para etiquetas y estado de archivos:** Se deberá implementar un sistema para almacenar y gestionar las etiquetas y los estados de los archivos (culminado, urgente, etc.). Bases de datos relacionales o NoSQL pueden ser útiles para esto.

- **Motor de búsqueda:** Para que los usuarios puedan buscar archivos por etiquetas, se necesitará un motor de búsqueda rápido y eficiente.

6. **Dependencias de infraestructura:**
   - **Backend:** El servidor necesitará un backend robusto que pueda gestionar la autenticación, la gestión de archivos, videollamadas, mensajería y demás funcionalidades. Tecnologías como Node.js, Django, Ruby on Rails o Spring podrían ser candidatas.
   - **Frontend:** El desarrollo de la interfaz web debe considerar bibliotecas y frameworks modernos (React, Angular, Vue.js) para facilitar la interacción con los usuarios.
   - **Base de datos:** Necesitarás una base de datos para almacenar usuarios, permisos, conversaciones, archivos, etiquetas, estados y otros metadatos. Puedes utilizar bases de datos relacionales (MySQL, PostgreSQL) o NoSQL (MongoDB, Firebase).
7. **Dependencias de seguridad:**
   - **Seguridad de la información:** La aplicación debe cumplir con regulaciones legales y estándares de seguridad, especialmente cuando se maneja información sensible de casos legales. Esto implica encriptación de datos, auditorías de seguridad y uso de firewalls.
8. **Dependencias de comunicación con otros sistemas:**
   - Si el despacho utiliza otros sistemas (por ejemplo, gestión de casos o bases de datos de clientes), puede ser necesario integrarlos a la aplicación a través de APIs o servicios web.

Aplicación web responsiva para móvil:

A1. Diseño y Desarrollo Web Responsivo

- **Frameworks CSS:** Uso de Bootstrap y Tailwind CSS para un diseño adaptable a diferentes tamaños de pantalla.

- **Media Queries:** Ajustes en CSS para optimizar la visualización en móviles, tablets y escritorios.
- **Diseño para Pantallas Táctiles:** Elementos interactivos diseñados para una fácil navegación en dispositivos táctiles.

A2. Optimización de la Experiencia Móvil

- **Minimización de Recursos:** Reducción del tamaño de imágenes, scripts y hojas de estilo.
- **Lazy Loading:** Carga diferida de imágenes y recursos para mejorar el rendimiento.
- **Compresión GZIP y CDN:** Técnicas para acelerar el tiempo de carga.
- **Almacenamiento en Caché:** Implementación de Service Workers para acceso sin conexión y mejoras en la velocidad.

A3. Videollamadas y Chat en Móviles

- **WebRTC:** Optimización para videollamadas en dispositivos móviles.
- **WebSockets/Firebase:** Para chat en tiempo real, garantizando comunicación rápida y eficiente.

A4. Notificaciones Push

- **Push Notifications API:** Integración para enviar notificaciones a través del navegador móvil.

A5. Acceso Offline

- **Service Workers:** Permiten que la aplicación funcione parcialmente sin conexión a Internet.

A6. Compatibilidad con Navegadores Móviles

- **Pruebas Cross-Browser:** Verificación de funcionalidad en navegadores móviles principales como Chrome, Safari y Firefox.

Limitaciones Técnicas

Hardware: A nivel de hardware la empresa cuenta con laptops a su disposición, solo el gerente general cuenta con una computadora de escritorio, por lo cual el aplicativo web debe de correr en laptops para que admita a cualquier trabajador de la empresa.

Software: Las laptops que maneja la empresa trabajan con Windows 10 y 11, el aplicativo web debe de adaptarse al software que maneje cada máquina, por lo cual tiene que tener una versión compatible con Windows 10 y otra con Windows 11, además de las laptops se requiere que el aplicativo pueda ejecutarse en celulares, los cuales trabajan en su mayoría con Android 10.

Conectividad: Se pretende que todos los trabajadores puedan ver documentos y tareas simultáneamente, por lo cual debe tener una alta cohesión, además que permita hacer modificaciones en tiempo real.

Infraestructura: Las instalaciones de la empresa cuentan con un espacio pequeño, cuentan con un organigrama de alrededor de 15 personas excluyendo a los becarios.

Gestión documental: Todos los documentos deben de ser accesibles en todo momento y se deben de compartir entre todos los trabajadores, además deben de poder agregarse luego de que el gerente general los apruebe.

Limitaciones de Recursos

1. Presupuesto Limitado

Tienen un presupuesto total de S/10,000 para el desarrollo del software. Esto puede limitar la complejidad y las características del software que se pueden implementar de inmediato, así como los recursos asignados para la implementación y el mantenimiento inicial.

2. Acceso a Recursos Tecnológicos

Solo tienen 15 GB de espacio gratuito en Google Drive. Esto es insuficiente para el volumen creciente de documentos, y la empresa necesitará una solución de almacenamiento más robusta para manejar y respaldar adecuadamente todos los archivos.

Actualmente utilizan herramientas básicas como Excel y Google Drive para manejar documentación y comunicación. No cuentan con servidores dedicados ni software especializado adicional, lo cual puede limitar las opciones para una solución integrada y eficiente.

3. Personal Calificado

No tienen personal dedicado a TI, lo que representa una limitación significativa. La empresa necesitará contratar personal especializado o externalizar el desarrollo y mantenimiento del software, así como la capacitación del personal existente en el uso del nuevo sistema.

4. Gestión de Respaldos

Usan Google Drive gratuito para respaldos, lo cual puede ser insuficiente para garantizar la integridad y seguridad de los datos. Además, la dependencia de métodos de respaldo físicos (discos externos) puede no ser suficiente para un sistema que requiere alta disponibilidad y seguridad.

5. Procesos Manuales Actuales

Enfrentan dificultades con la lectura, accesibilidad y visualización de documentos utilizando Google Drive y Excel. Estos problemas indican la necesidad de un sistema que mejore la organización, la búsqueda y la gestión documental para facilitar el acceso y la colaboración.

2. Vista General

2.1 Situación actual

La organización actualmente lleva a cabo sus procesos de manera semi-automatizada, utilizando herramientas básicas como Google Drive y Excel para la gestión documental y la comunicación. No cuentan con un sistema integral que permita gestionar todas las operaciones de manera eficiente, lo que genera dificultades en la accesibilidad y organización de la información.

- **Características técnicas:** Las laptops disponibles en la empresa utilizan Windows 10 y 11, con 15 GB de espacio gratuito en Google Drive para el almacenamiento de documentos. El sistema debe ser compatible con estos entornos operativos y permitir la visualización y edición de documentos en tiempo real. También se requiere que el aplicativo funcione en dispositivos móviles, principalmente con Android 10.
- **Desarrollo y subsistemas:** Actualmente, los sistemas de gestión documental, comunicación interna y organización de tareas son gestionados manualmente mediante Google Drive, Excel y Famcal. No existe una infraestructura TI robusta que soporte una automatización integral de los procesos.

2.1.1 Necesidades y Requisitos

Necesidades para satisfacer:

El nuevo sistema es necesario para mejorar la gestión documental, la comunicación entre trabajadores y la visualización de tareas en tiempo real. El actual sistema manual basado en Google Drive y Excel es insuficiente, ya que genera problemas de accesibilidad, retrasos en la aprobación de documentos y limitaciones en la colaboración simultánea entre trabajadores.

Requerimientos funcionales:

- Autenticación y autorización de usuarios mediante un sistema seguro (OAuth, SSO).

- Roles de usuario con permisos diferenciados según su posición (administrativos, abogados).
- Gestión documental eficiente con acceso en tiempo real.
- Visualización y modificación de documentos y tareas en tiempo real.
- Calendarización, el sistema debe permitir cronogramas y notificaciones de los procesos pertinentes (Costum).

Desarrollo Requerimientos Funcionales

No.	Requerimiento Funcional	Descripción
1	Autenticación y autorización de usuarios	Implementar un sistema de autenticación robusta (OAuth o SSO) para verificar la identidad de los usuarios y un sistema de autorización basado en roles para definir los permisos de acceso a las funcionalidades del sistema.
2	Roles de usuario con permisos diferenciados	Definir roles específicos (administradores, abogados, etc.) y asignarles permisos diferenciados para acceder a determinadas funcionalidades y datos.
3	Gestión documental eficiente con acceso en tiempo real	Implementar un sistema de gestión documental que permita a los usuarios subir, descargar, editar y compartir documentos de forma rápida y segura. La información debe estar disponible en tiempo real para todos los usuarios autorizados.

4	Visualización y modificación de documentos y tareas en tiempo real	Permitir a los usuarios visualizar y modificar documentos y tareas de manera simultánea y en tiempo real, facilitando la colaboración y la toma de decisiones.
5	Calendarización y notificaciones	Implementar un sistema de calendarización que permita a los usuarios crear eventos, establecer recordatorios y recibir notificaciones sobre las tareas y procesos pendientes.

Tabla 1: Requerimientos Funcionales

Requerimientos no funcionales:

- Seguridad de datos mediante encriptación (AES para almacenamiento, HTTPS para conexiones).
- Alta disponibilidad del sistema para todos los trabajadores, evitando interrupciones.
- Compatibilidad con Windows 10, Windows 11 y Android 10 en adelante.
- Escalabilidad y modularización.
- Usabilidad: El sistema se caracterizará por ser amigable con el usuario y responsivo.

Desarrollo Requerimientos no Funcionales

No.	Requerimiento No Funcional	Descripción
1	Seguridad de datos	Implementar medidas de seguridad robustas para proteger la información confidencial, incluyendo el cifrado de datos en reposo (AES) y en tránsito (HTTPS).

2	Alta disponibilidad	Garantizar que el sistema esté disponible en todo momento para todos los usuarios, minimizando las interrupciones en el servicio.
3	Compatibilidad	Asegurar que el sistema funcione correctamente en los siguientes sistemas operativos: Windows 10, Windows 11 y Android 10 en adelante.
4	Escalabilidad y modularización	Diseñar el sistema de manera que pueda adaptarse a un aumento en el número de usuarios y funcionalidades, permitiendo una fácil ampliación y mantenimiento.
5	Usabilidad	Garantizar que la interfaz de usuario sea intuitiva y fácil de utilizar, facilitando la adopción del sistema por parte de los usuarios.

Tabla 2: Requerimientos no Funcionales

2.2 Roles y Responsabilidades de los Stakeholders

2.2.1 Roles

Roles de Consafil:

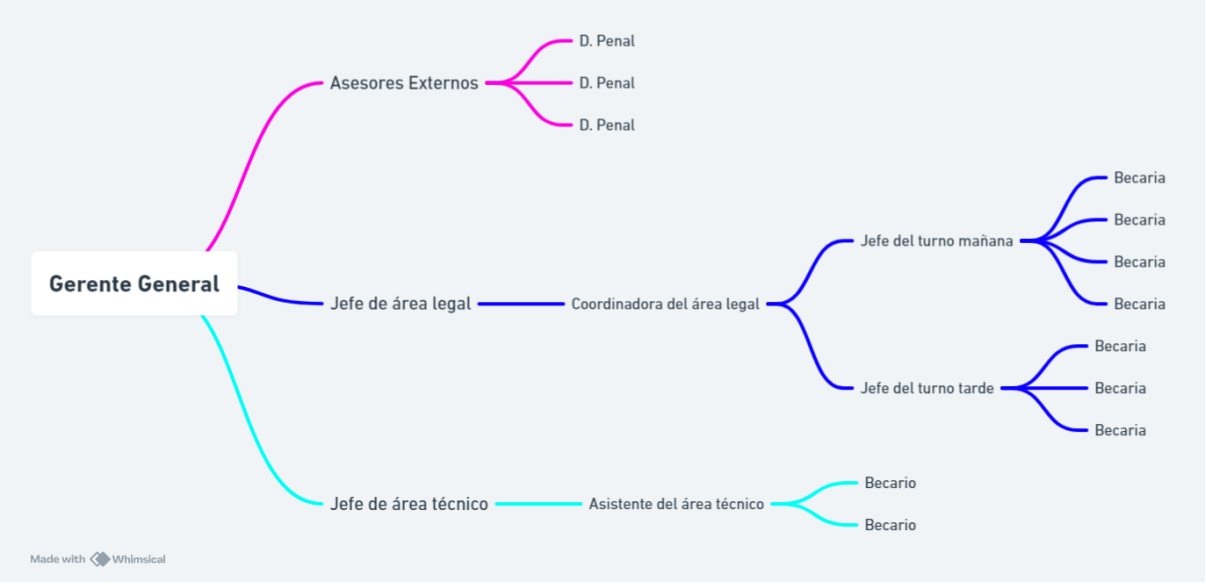

Ilustración 1: Roles de Consafil

Roles de Rechts Schreibtisch:

Nombre	Rol
Jorge Ortiz Castañeda	Tech Lead
Jhordan Huamani Huamani	Arquitecto de Software
Gian Piero Rodriguez Fadel	Desarrollador Back-end
Miguel Flores León	Desarrollador Front-end
Christofer Gamio Huaman	Desarrollador Front-end
Abimael Frontado Fajardo	Aseguramiento de la Calidad (QA)

Tabla 3: Roles de Rechts Schreibtisch

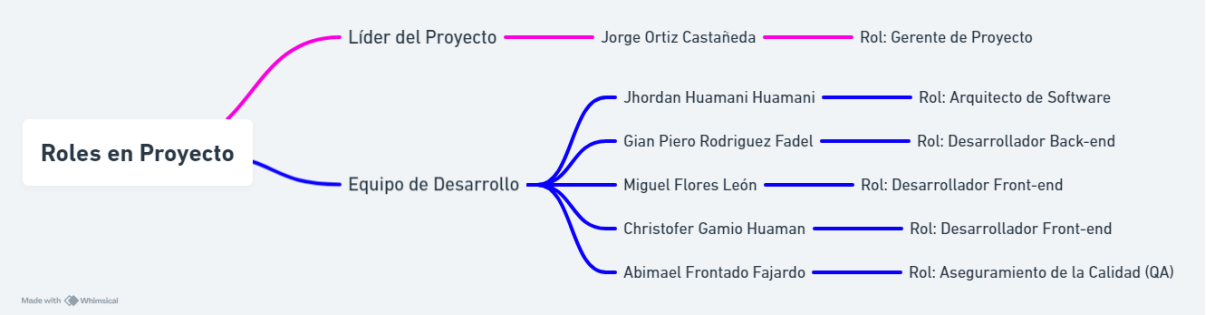

Ilustración 2: Roles de Rechts Schreibtisch

2.2.2 Responsabilidades

Responsabilidades de Consafil:

Los Becarios podrán:

- Buscar información: Realizar búsquedas detalladas dentro de los documentos para encontrar la información necesaria.
- Visualizar documentos: Acceder y revisar los documentos existentes, pero sin la posibilidad de realizar modificaciones.

- Colaborar en tareas asignadas: Participar activamente en las tareas que les sean encomendadas, siguiendo las instrucciones de sus superiores.

Rol	Responsabilidades	Permisos
Becario	Investigación de casos, apoyo en la recopilación de documentos, asistencia en tareas administrativas, aprendizaje de los procesos internos.	Búsqueda dentro de documentos, visualización de documentos, acceso a módulos específicos de capacitación.
Abogado Junior	Análisis de documentos legales, elaboración de informes preliminares, asistencia en la preparación de escritos, participación en reuniones.	Edición de documentos propios, creación de nuevos documentos, acceso a bases de datos de jurisprudencia.
Abogado Senior	Gestión de casos complejos, representación legal, supervisión de equipos, desarrollo de estrategias legales.	Todos los permisos, administración de usuarios, configuración de la plataforma.

Tabla 4: Roles y Responsabilidades Consafil

Responsabilidades Rechts Schreibtisch:

Líder del Proyecto:

Rol: Tech Lead

Responsabilidades:

- Definición del alcance del proyecto.
- Creación del plan de proyecto.

- Gestión de los recursos (equipo, presupuesto).
- Comunicación con el cliente (la firma de abogados).
- Gestión de riesgos.
- Supervisión del progreso del proyecto.

Equipo de Desarrollo:

Rol: Arquitecto de Software

Responsabilidades:

- Diseño de la arquitectura de la aplicación.
- Selección de tecnologías.
- Definición de las interfaces entre los componentes.
- Asegurar la escalabilidad y mantenibilidad de la aplicación.

Rol: Desarrollador Back-end

Responsabilidades:

- Desarrollo de la lógica del negocio de la aplicación.
- Integración con bases de datos.
- Creación de APIs.
- Optimización del rendimiento del servidor.

Rol: Desarrollador Front-end

Responsabilidades:

- Desarrollo de la interfaz de usuario.
- Creación de una experiencia de usuario intuitiva.
- Asegurar la compatibilidad con diferentes dispositivos.

Rol: Desarrollador Front-end

Responsabilidades:

- Desarrollo tanto del front-end como del back-end.
- Colaboración en todas las fases del desarrollo.
- Soporte técnico.

Rol: Aseguramiento de la Calidad (QA)

Responsabilidades:

- Diseño y ejecución de pruebas.
- Identificación y reporte de defectos.
- Verificación de la calidad del software.

2.3 Asunciones y Riesgos del Sistema

2.3.1 Asunciones

Técnicas: Se asume que el sistema será desarrollado sobre una infraestructura sencilla que utilice tecnologías estándar (como Node.js, SQL) compatibles con las herramientas disponibles en la empresa.

Operativas: Se espera que todos los trabajadores tengan acceso constante a internet y que las laptops con Windows 10 y 11 puedan ejecutar el sistema sin inconvenientes.

Regulatorias: La aplicación deberá cumplir con normativas de seguridad de datos, incluyendo la protección de información legal sensible.

Financieras: Se asume que el proyecto podrá realizarse en su totalidad con el presupuesto dado.

2.3.2 Riesgos

Técnicos: Riesgo de incompatibilidad con los sistemas operativos (Windows 10, 11 y Android 10). Además, la limitada capacidad de almacenamiento en Google Drive (15 GB) podría no ser suficiente para gestionar todos los documentos necesarios.

Operativos: La falta de personal especializado en TI en la empresa representa un riesgo significativo para el desarrollo y mantenimiento del sistema.

Externos: Cambios en las regulaciones sobre la protección de datos podrían requerir ajustes adicionales en la implementación.

Recursos Humanos: La capacitación del personal en el uso del nuevo sistema podría demorar la adopción plena del mismo.

Financieros: El presupuesto limitado podría restringir las funcionalidades o comprometer el mantenimiento y actualización del sistema a largo plazo.

3. Diseño y Desarrollo de la Solución

3.1 Objetivos

El objetivo principal de esta solución es desarrollar una plataforma de gestión documental robusta y escalable que permita a la empresa de asesoría legal optimizar sus procesos internos, mejorar la eficiencia y garantizar la seguridad de la información. Específicamente, se busca:

Optimización de procesos:

- Automatizar la captura, clasificación y almacenamiento de documentos.
- Establecer flujos de trabajo personalizados para cada tipo de documento (contratos, informes, correspondencia, etc.).
- Implementar un sistema de notificaciones para recordar fechas de vencimiento, revisiones y otros eventos relevantes.
- Reducir el tiempo de búsqueda y recuperación de documentos.

Mejora de la colaboración:

- Facilitar la colaboración en tiempo real entre múltiples usuarios sobre un mismo documento.

- Implementar un sistema de control de versiones para rastrear los cambios realizados en los documentos.
- Permitir la creación de espacios de trabajo colaborativos para proyectos específicos.

Aumento de la seguridad:

- Garantizar la confidencialidad de la información mediante el cifrado de datos en reposo y en tránsito.
- Implementar un sistema de control de acceso basado en roles y perfiles de usuario.
- Realizar auditorías de seguridad periódicas para identificar y mitigar posibles vulnerabilidades.
- Cumplir con las normativas legales y sectoriales en materia de protección de datos.

Mejora de la toma de decisiones:

- Generar informes y análisis sobre el estado de la documentación.
- Facilitar la identificación de tendencias y patrones en los datos.
- Proporcionar herramientas de visualización de datos para una mejor comprensión de la información.

Escalabilidad y flexibilidad:

- Diseñar una arquitectura escalable para adaptarse al crecimiento de la empresa y al aumento del volumen de documentos.
- Permitir la personalización de la plataforma para ajustarla a las necesidades específicas de cada usuario.

3.2 Entorno Operacional

Para garantizar el funcionamiento óptimo de la plataforma, se ha diseñado un entorno operacional robusto y escalable, compuesto por:

Herramientas y tecnologías centrales:

- **Android Studio:** IDE principal para el desarrollo de aplicaciones Android y como entorno de desarrollo unificado para Kotlin Multiplatform.
- **Kotlin Multiplatform:** Lenguaje de programación que permite compartir código entre diferentes plataformas, incluyendo Android, iOS, web y escritorio.
- **Gradle:** Sistema de construcción utilizado por Android Studio para gestionar las dependencias y configurar los proyectos.
- **Compose Multiplatform:** Toolkit de interfaz de usuario declarativo para crear interfaces de usuario nativas para Android, iOS, web y escritorio.

Estructura del proyecto:

- **Módulo compartido:** Contiene el código Kotlin que se comparte entre todas las plataformas, como lógica de negocio, modelos de datos y utilidades comunes.
- **Módulo Android:** Contiene el código específico para la plataforma Android, incluyendo la interfaz de usuario con Compose, la interacción con los servicios de Android y las dependencias específicas de Android.
- **Módulo Desktop:** Contiene el código específico para la plataforma de escritorio, incluyendo la interfaz de usuario con Compose para Desktop, la interacción con el sistema operativo y las dependencias específicas de escritorio.

Flujo de trabajo:

1. **Desarrollo del módulo compartido:** Se escribe el código Kotlin compartido en el módulo compartido, enfocándose en la lógica de negocio y las entidades de datos.

2. **Desarrollo de los módulos específicos:** Se desarrolla la interfaz de usuario y la lógica específica para cada plataforma en los módulos Android y Desktop, utilizando Compose Multiplatform.
3. **Compilación y ejecución:** Se utiliza Gradle para compilar el proyecto y generar los artefactos para cada plataforma. Android Studio proporciona herramientas para ejecutar y depurar las aplicaciones en los emuladores o dispositivos físicos.

Consideraciones:

- **Plataformas de escritorio:** Kotlin Multiplatform permite generar aplicaciones de escritorio para diversas plataformas, como Windows, macOS y Linux. Es importante seleccionar las plataformas de destino y configurar las dependencias correspondientes.
- **Gestores de paquetes:** Se utilizarán los gestores de paquetes específicos de cada plataforma (por ejemplo, Maven o Gradle) para gestionar las dependencias de las bibliotecas de terceros.
- **Distribución de aplicaciones:** Se utilizarán los mecanismos de distribución estándar para cada plataforma (Google Play Store para Android, App Store para iOS, tiendas de aplicaciones de escritorio, etc.).

Ejemplo de estructura de un proyecto Kotlin Multiplatform:

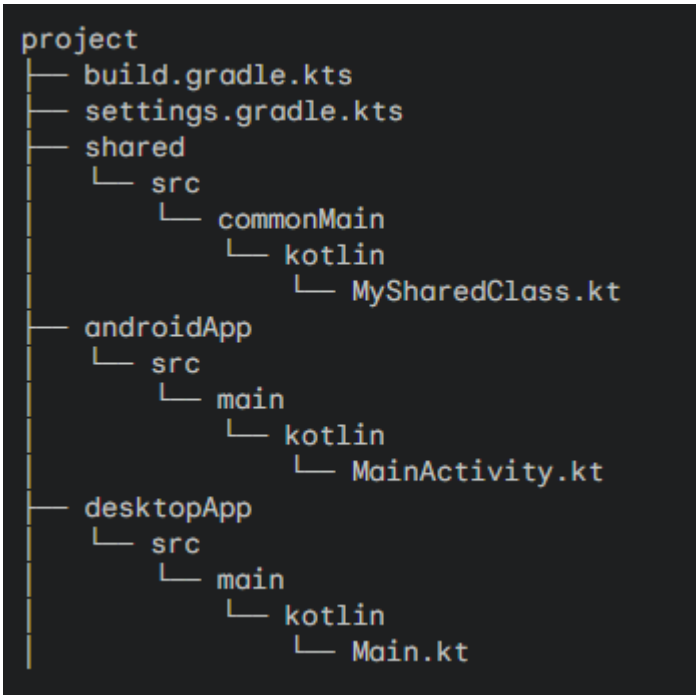

Ilustración 3: Estructura Proyecto Multiplataforma

3.3 Aspectos de Calidad y de Comportamiento

La calidad y el comportamiento de la aplicación son fundamentales para garantizar una buena experiencia de usuario y el éxito del proyecto. A continuación, se detallan los aspectos clave a considerar:

3.3.1 Desempeño

- **Optimización de código:** Evitar operaciones costosas, para mejorar el rendimiento de la aplicación.
- **Gestión de memoria:** Implementar una gestión de memoria eficiente para evitar fugas de memoria y garantizar un funcionamiento fluido de la aplicación.
- **Renderizado eficiente:** Optimizar el proceso de renderizado de la interfaz de usuario para lograr una experiencia visual fluida, especialmente en dispositivos móviles.

3.3.2 Seguridad

- **Seguridad de datos:** Implementar mecanismos de cifrado para proteger los datos sensibles del usuario, tanto en tránsito como en reposo.
- **Autenticación y autorización:** Implementar mecanismos de autenticación robustos y sistemas de autorización basados en roles para proteger el acceso a los recursos de la aplicación.
- **Protección contra vulnerabilidades:** Mantener actualizadas las bibliotecas y dependencias utilizadas para evitar la explotación de vulnerabilidades conocidas.
- **Pruebas de seguridad:** Realizar pruebas de seguridad de forma regular para identificar y corregir posibles vulnerabilidades.

3.3.3 Confiabilidad

- **Manejo de errores:** Implementar un manejo de errores robusto para evitar que la aplicación se bloquee en caso de errores inesperados.
- **Pruebas unitarias:** Escribir pruebas unitarias exhaustivas para garantizar la corrección del código y detectar errores tempranamente.
- **Pruebas de integración:** Realizar pruebas de integración para verificar que los diferentes componentes de la aplicación funcionan correctamente juntos.
- **Pruebas de UI:** Automatizar las pruebas de la interfaz de usuario para garantizar que funcione correctamente en diferentes dispositivos y configuraciones.

3.3.4 Usabilidad

- **Diseño de interfaz de usuario intuitiva:** Utilizar principios de diseño de interfaces de usuario para crear una experiencia de usuario agradable y fácil de usar.

- **Accesibilidad:** Asegurar que la aplicación sea accesible para usuarios con discapacidades, cumpliendo con las pautas de accesibilidad.
- **Localización:** Permitir que la aplicación se adapte a diferentes idiomas y regiones.

3.3.5 Concurrencia

- **Corrutinas:** Utilizar corrutinas de Kotlin para manejar tareas asíncronas de forma eficiente y evitar bloqueos de la interfaz de usuario.
- **Sincronización:** Implementar mecanismos de sincronización para evitar condiciones de carrera y garantizar la consistencia de los datos.

3.3.6 Persistencia de datos

- **Selección de la base de datos:** Para este proyecto, hemos seleccionado **PostgreSQL** como base de datos principal debido a su robustez, escalabilidad y amplio conjunto de características. PostgreSQL ofrece un excelente soporte para transacciones ACID, relaciones complejas y funcionalidades avanzadas como JSONB para almacenar datos semiestructurados.

  Transacciones ACID: Esto garantiza que las operaciones de la base de datos sean atómicas, consistentes, aisladas y duraderas, evitando problemas de corrupción de datos.

  Relaciones complejas: Permite establecer relaciones entre diferentes tipos de datos, como por ejemplo, una tabla de clientes y una tabla de pedidos.

  JSONB: Esta característica es útil para almacenar datos con una estructura flexible, como datos de configuración o información proveniente de APIs externas.
- **Optimización de consultas:** Utilizaremos índices, vistas materializadas y consultas parametrizadas para mejorar el rendimiento de las consultas SQL. Además, se emplearán herramientas de análisis de consultas para identificar y solucionar cuellos de botella.

Índices: Son como los índices de un libro, te permiten encontrar rápidamente la información que necesitas.

Vistas materializadas: Son como resúmenes precalculados de datos, lo que acelera las consultas que se ejecutan con frecuencia.

Consultas parametrizadas: Evitan que la base de datos tenga que analizar la misma consulta varias veces, lo que mejora el rendimiento.

- **Migraciones de datos:** Implementaremos un sistema de migraciones de datos basado en **Flyway** o **Liquibase** para gestionar de forma segura y eficiente los cambios en el esquema de la base de datos a lo largo del tiempo.

  Flyway y Liquibase son herramientas que automatizan el proceso de migración de datos. Te permiten definir los cambios que quieres hacer en la base de datos en archivos de configuración y ejecutarlos de forma ordenada y segura.

3.3.7 Manipulación de errores y excepciones

- **Manejo de excepciones personalizado:** Crear un sistema de manejo de excepciones personalizado para proporcionar mensajes de error claros y útiles.
- **Registro de errores:** Registrar los errores en un log para facilitar la depuración y el análisis.

3.3.8 Tolerancia a fallos

- **Recuperación de errores:** Implementar mecanismos de recuperación de errores para permitir que la aplicación se recupere de fallos y continúe funcionando.

3.4 Estrategia Arquitectónica

La arquitectura de la plataforma se basa en los siguientes principios:

- **Modularidad:** La plataforma se ha dividido en módulos bien definidos, lo que facilita el mantenimiento y la escalabilidad.
- **Escalabilidad:** La arquitectura permite escalar la plataforma horizontalmente y verticalmente para adaptarse a las necesidades cambiantes del negocio.
- **Flexibilidad:** La plataforma es altamente configurable, permitiendo adaptar la funcionalidad a las necesidades específicas de cada cliente.
- **Seguridad:** La seguridad se ha integrado en todos los niveles de la arquitectura, desde la infraestructura hasta el código.
- **Usabilidad:** La interfaz de usuario se ha diseñado siguiendo los principios de usabilidad, con el objetivo de facilitar la interacción de los usuarios.

Detallado de elementos a usar

Uso de UML: El uso de UML (Unified Modeling Language) facilita la abstracción de los módulos clave en una organización. Estas representaciones sirven como una base sólida para el desarrollo de arquitecturas futuras, permitiendo una comprensión clara y estructurada de los componentes esenciales del sistema.

Uso de Layered Architecture: La Arquitectura en Capas permite visualizar el sistema de manera estructurada, separando las diferentes capas funcionales y delimitando claramente los niveles de acceso. Esta organización resulta fundamental para diferenciar los permisos entre los usuarios finales y el personal de TI, asegurando así un control robusto y preciso de las interacciones dentro del sistema.

Uso de ADL(Acme): El uso de lenguajes de descripción arquitectónica como ADL, específicamente Acme, proporciona un enfoque preciso para modelar las conexiones entre componentes de software. Esta herramienta describe de manera explícita las relaciones físicas y

lógicas dentro de la infraestructura, facilitando un diseño arquitectónico detallado y eficiente que optimiza la interacción entre los elementos del sistema.

4. Diseño Arquitectónico

4.1 Arquitectura Lógica

4.1.1 Arquitectura por Capas - Esquema

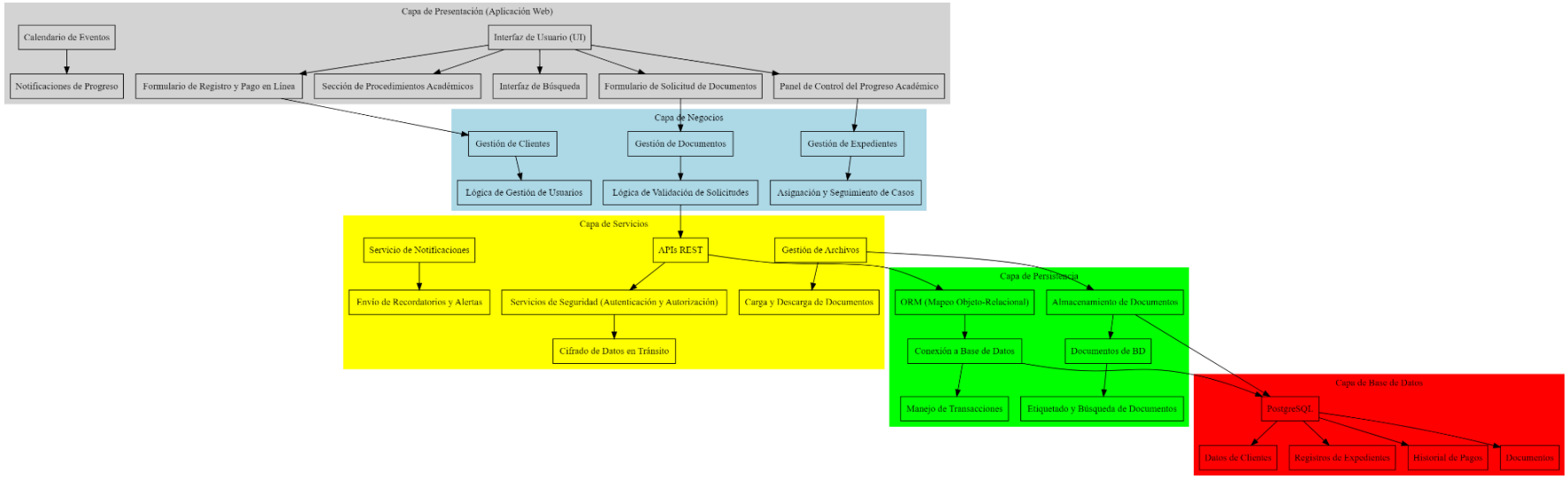

Ilustración 4: Arquitectura por Capas – Presentación Esquemática

4.1.2 Arquitectura por Capas

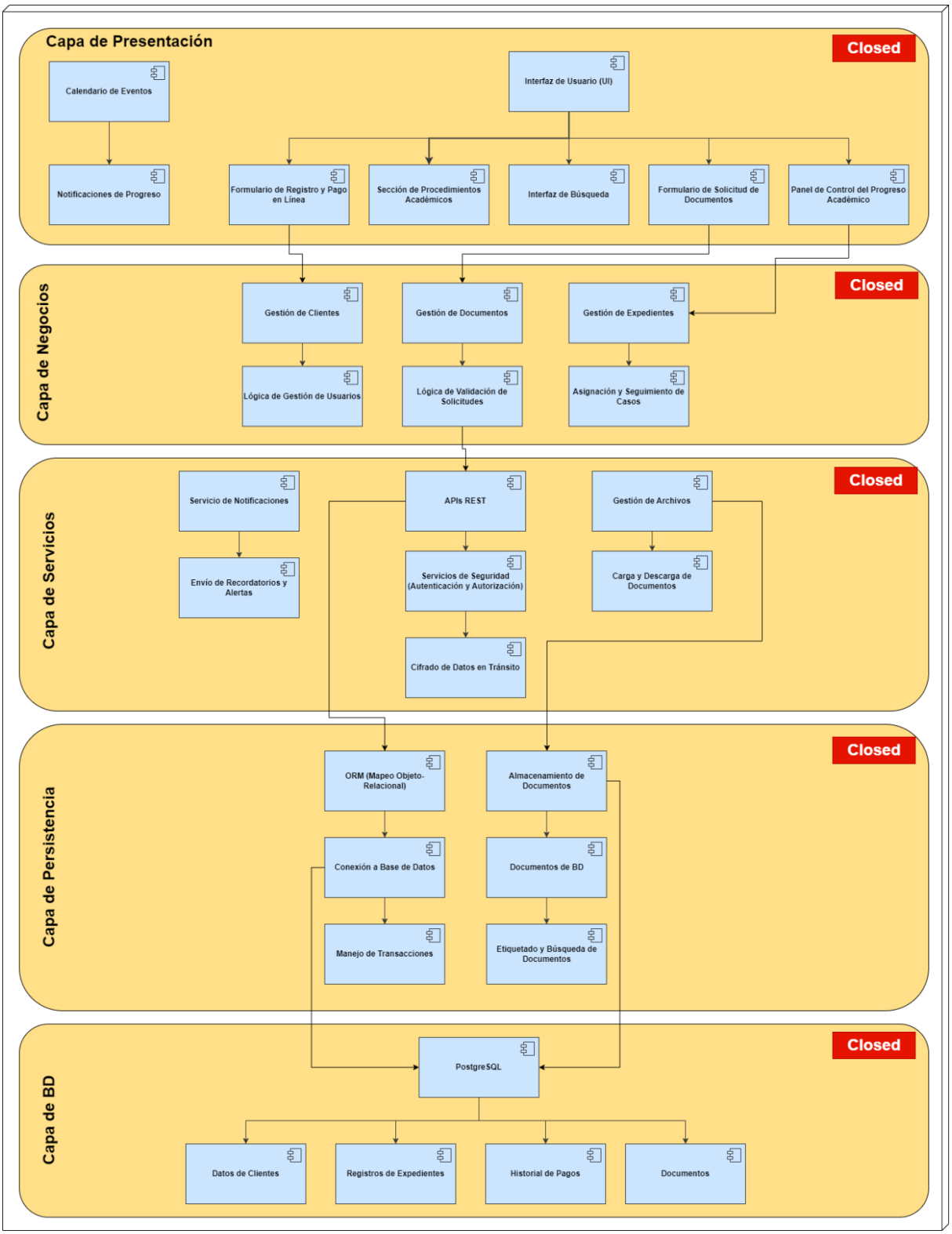

Ilustración 5: Arquitectura por Capas

4.1.3 Diagrama UML

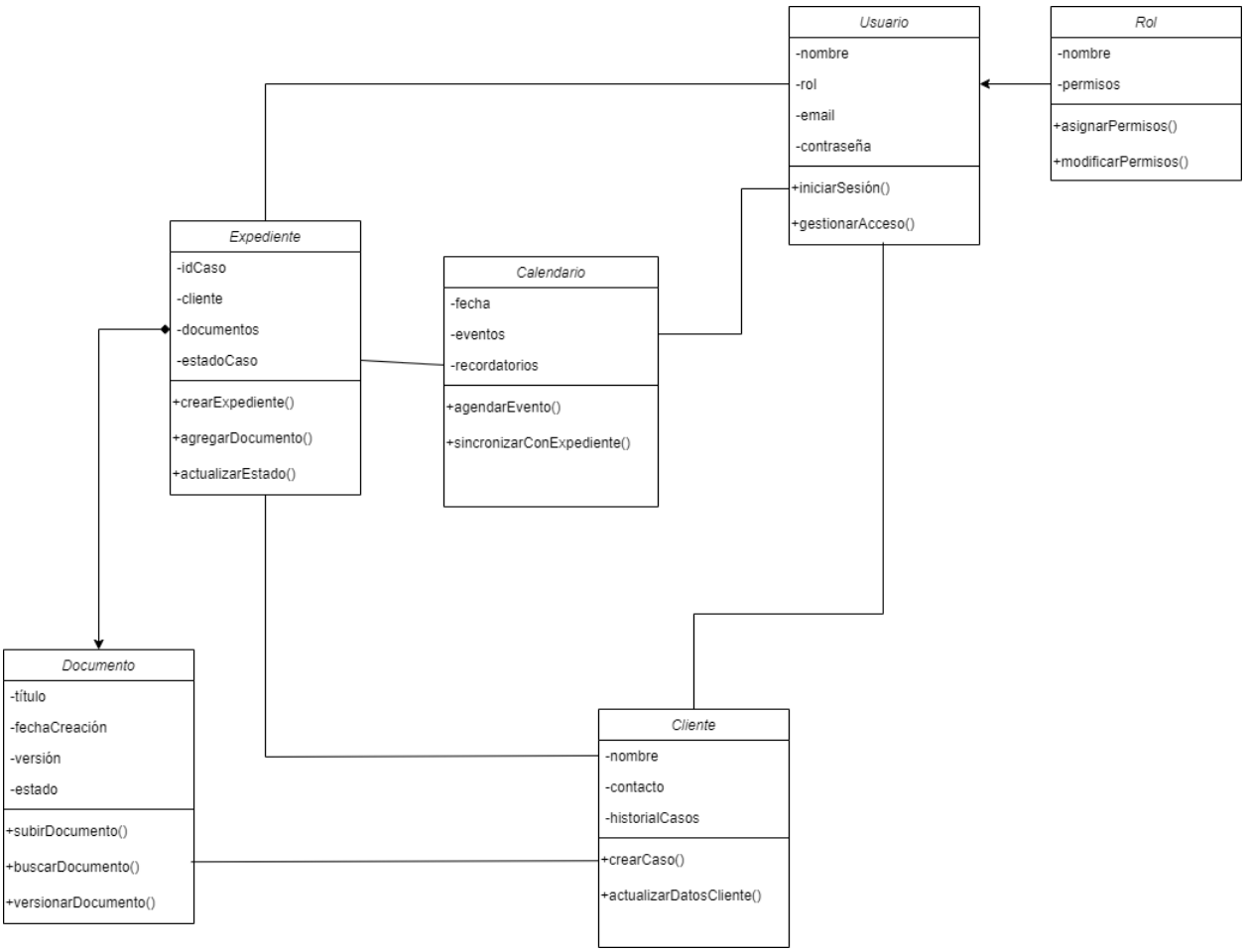

Ilustración 6: Diagrama UML

4.2 Arquitectura Física

4.2.1 Gráfico de Componentes Internos

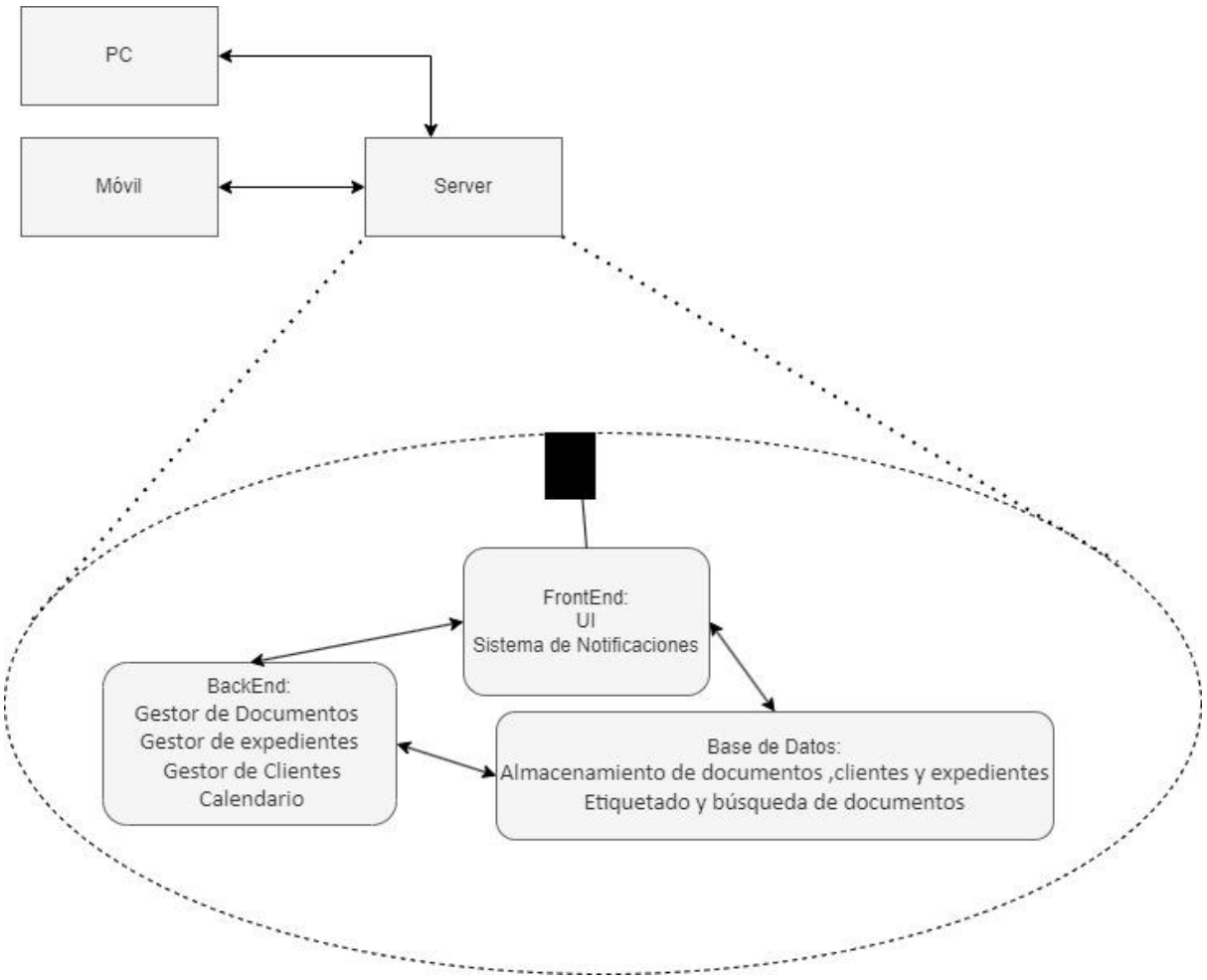

Ilustración 7: Gráfico de Componentes internos

4.2.2 Código ADL del Gráfico de Componentes Internos

```
System CONSAFILCS = {
        Component Computer = {
                Ports { ServerSocketDesktop; }
        }
        Component Phone = {
                Ports { ServerSocketMobile; }
        }
        Component ServerNAS = {
                Ports { ManagementDesktop; ManagementMobile; InternalSocket }
                Representation ServerNASDetails = {
                        System ServerNASDetailsSys = {
                                Component Frontend = {
                                        Ports { FrontendDataConnection; BackInteraction; ExternalSocket }
                                }
                                Component Backend = {
                                        Ports { BackendDataConnection; FrontInteraction; }
                                }
                                Component Base_de_Datos = {
                                        Ports { FrontendDataReceive; BackendDataReceive; }
                                }
                                Connector FrontCentralDataBase = { Roles { FCDB_ONE; FCDB_TWO } }
                                Connector BackCentralDataBase = { Roles { BCDB_ONE; BCDB_TWO } }
                                Connector AppInteraction = { Roles { AppFrontInteraction; AppBackInteraction } }
                                Attachments {
                                        Frontend.FrontendDataConnection to FrontCentralDataBase.FCDB_ONE;
                                        Base_de_Datos.FrontendDataReceive to FrontCentralDataBase.FCDB_TWO;
                                        Backend.BackendDataConnection to BackCentralDataBase.BCDB_ONE;
                                        Base_de_Datos.BackendDataReceive to BackCentralDataBase.BCDB_TWO;
                                        Frontend.BackInteraction to AppInteraction.AppFrontInteraction;
                                        Backend.FrontInteraction to AppInteraction.AppBackInteraction;
                                }
                        }
                        Bindings { Frontend.ExternalSocket to ServerNAS.InternalSocket }
                }
        }
        Connector SQLDesktop = { Roles { Desktopcaller; DesktopNAScallee } }
        Connector SQLPhone = { Roles { Mobilecaller; MobileNAScallee } }
        Attachments {
                Computer.ServerSocketDesktop to SQLDesktop.Desktopcaller;
                ServerNAS.ManagementDesktop to SQLDesktop.DesktopNAScallee;
                Phone.ServerSocketMobile to SQLPhone.Mobilecaller;
                ServerNAS.ManagementMobile to SQLPhone.MobileNAScallee;
        }
}
```

Ilustración 8: Código ADL

4.2.3 Gráfico ADL de Componentes

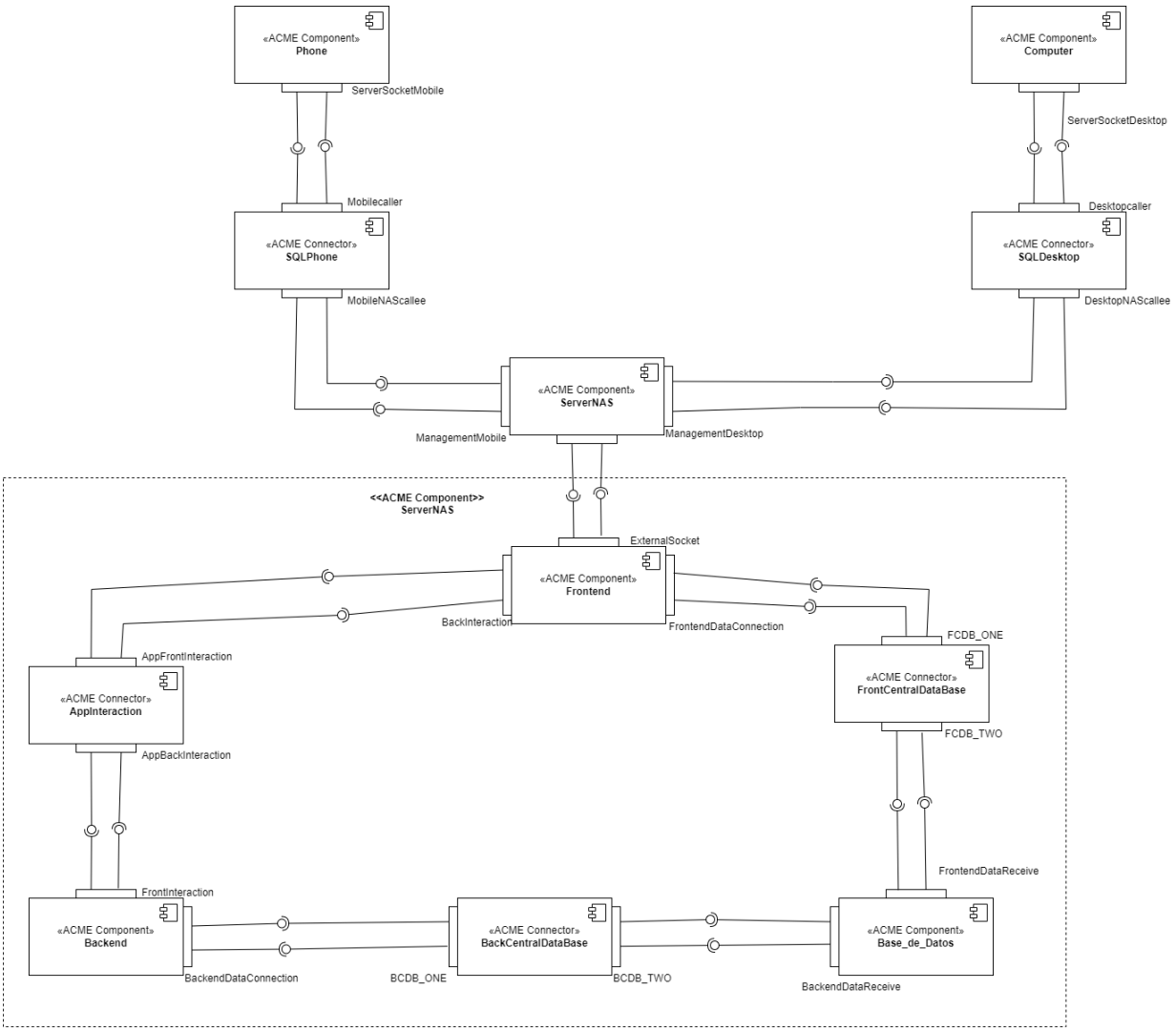

Ilustración 9: Gráfico ADL de Componentes Internos

5. Diseño Detallado

5.1 Diagramas Estructurales

5.1.1 Diagrama de Clases

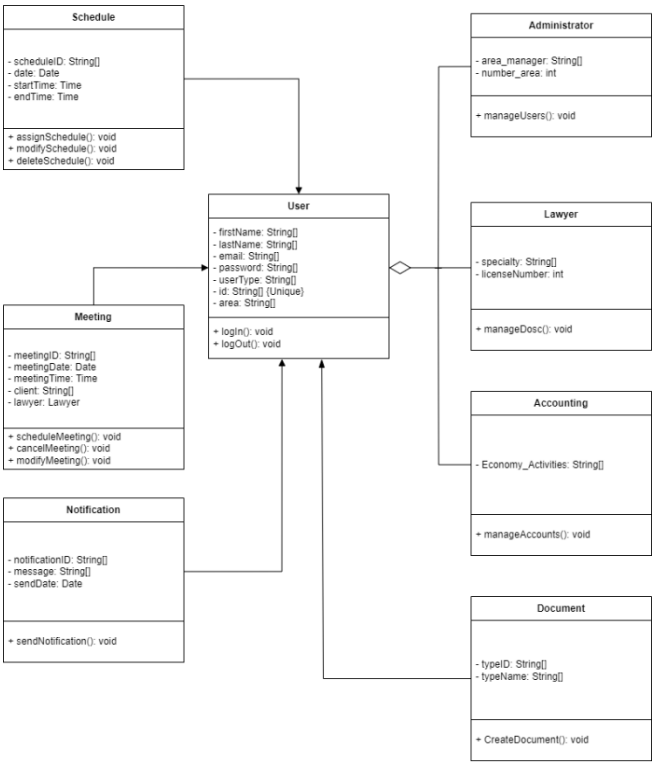

Ilustración 10: Diagrama de Clases

5.1.2 Diagrama de objetos

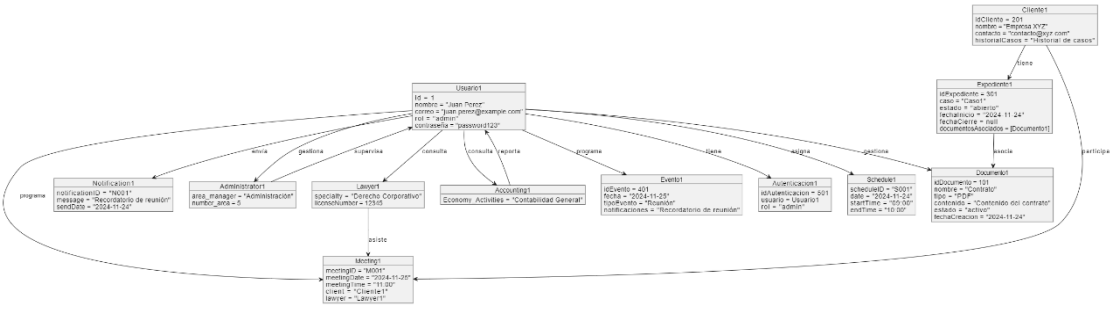

Ilustración 11: Diagrama de Objetos

5.1.3 Diagrama de componentes

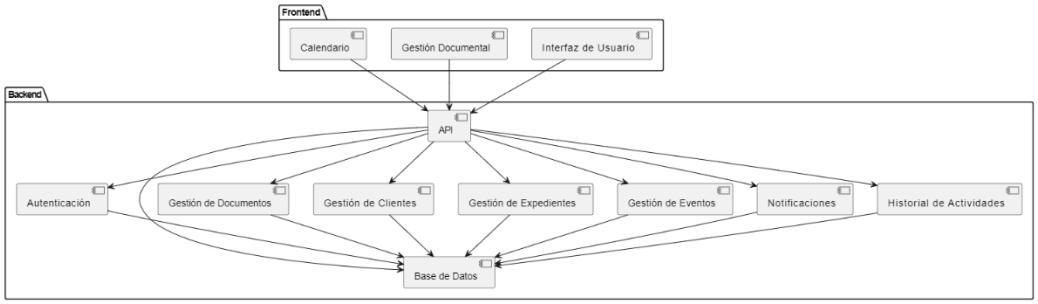

Ilustración 12: Diagrama de Componentes

5.1.4 Diagrama de despliegue

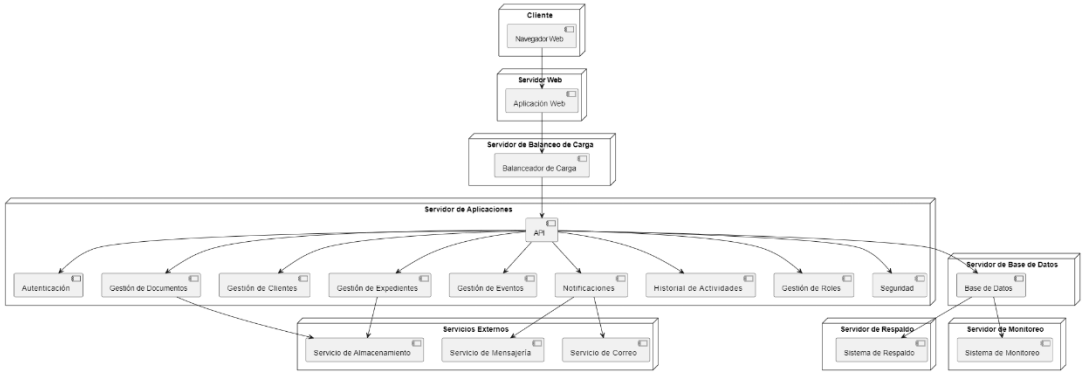

Ilustración 13: Diagrama de despliegue

5.1.5 Diagrama de paquetes

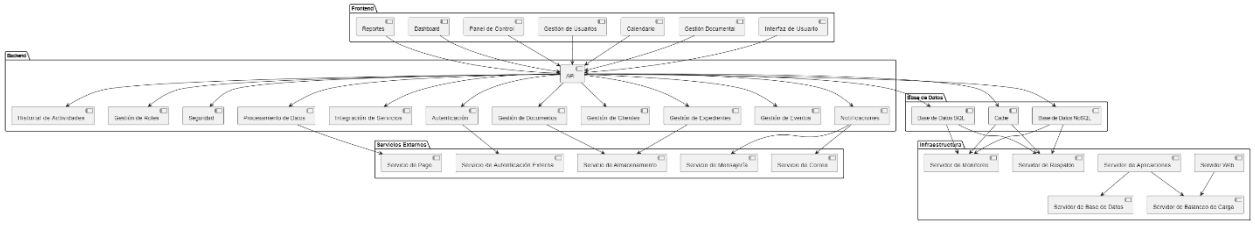

Ilustración 14: Diagrama de Paquetes

5.2 Diagramas de Comportamiento

5.2.1 Diagrama de Flujo de Datos

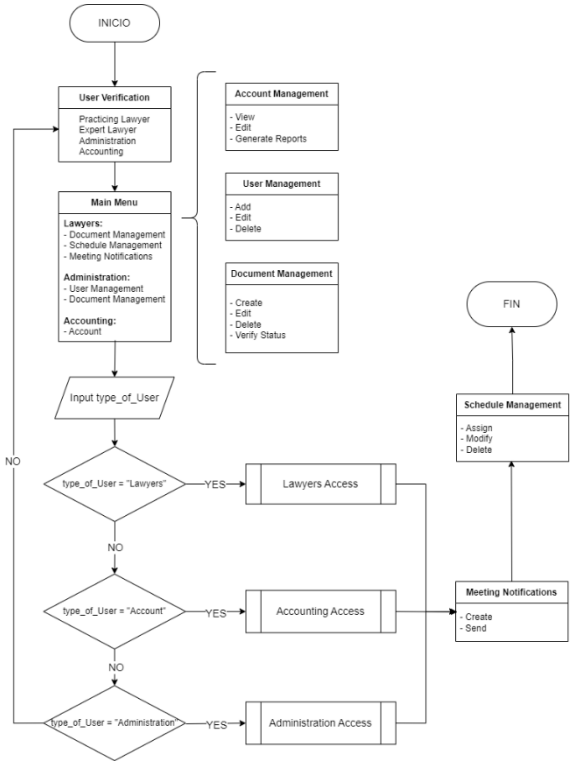

Ilustración 15: Diagrama de Flujo de Datos

5.2.2 Diagrama de Actividad

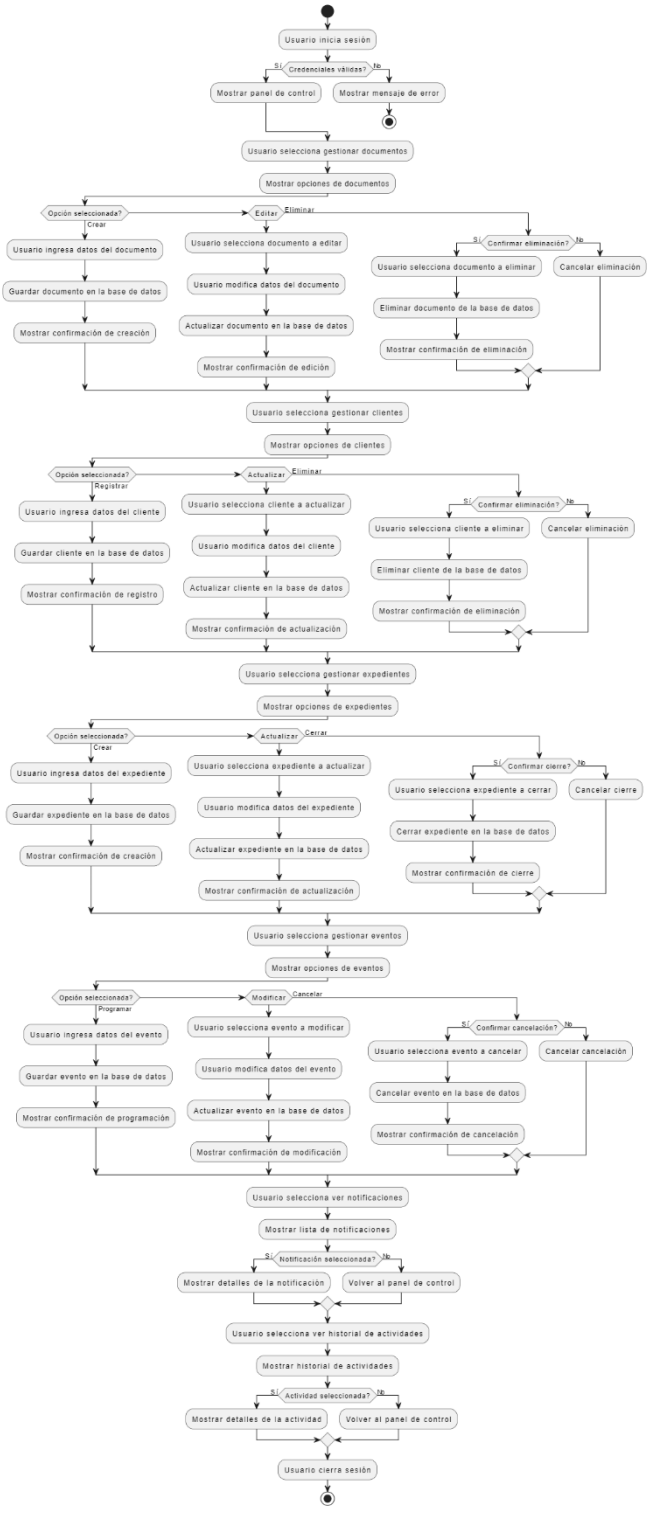

Ilustración 16: Diagrama de Actividades

5.2.3 Diagrama de Estados

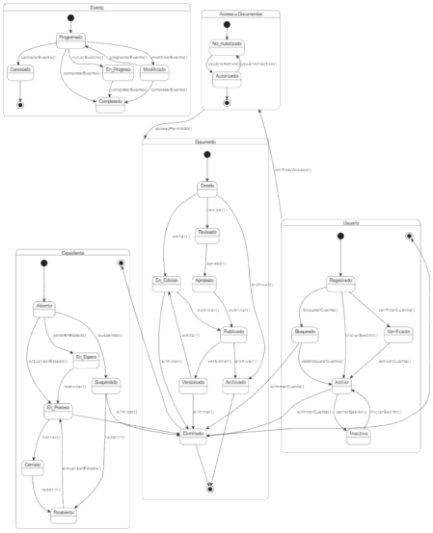

Ilustración 17: Diagrama de Estados

5.2.4 Diagrama de Casos de Uso

Ilustración 18: Diagrama de Casos de Uso

5.2.5 Diagrama de interacción

5.2.5.1 Inicio de Sesión

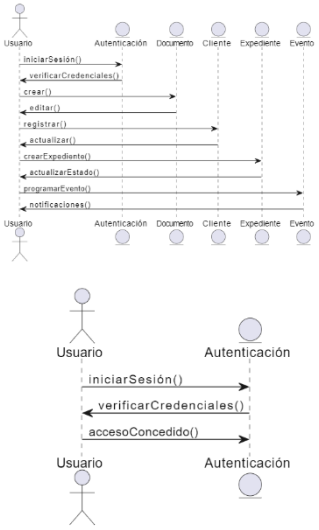

Ilustración 19: Inicio de Sesión

5.2.5.2 Gestión de Documentos

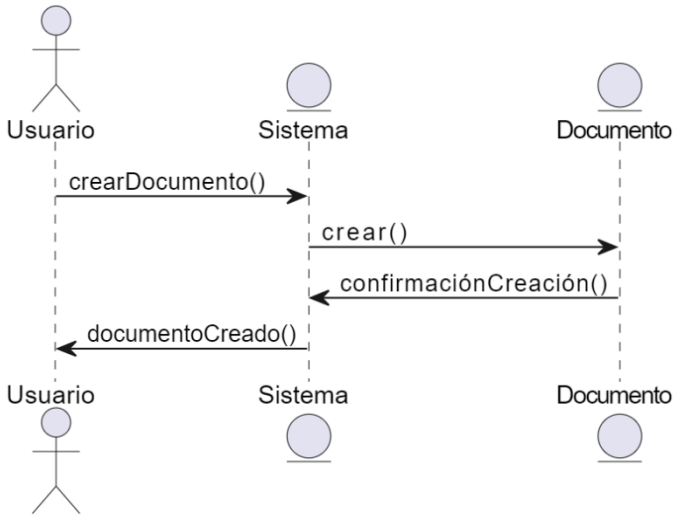

Ilustración 20: Gestión de Documentos

5.2.5.3 Clientes y Expedientes

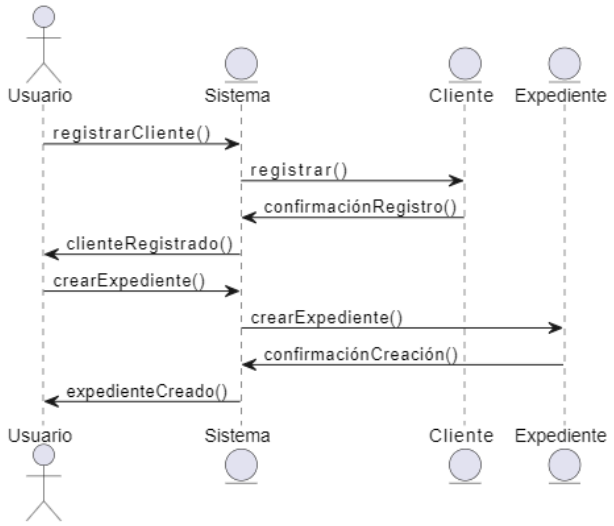

Ilustración 21: Cliente y Expedientes

5.2.5.4 Programación de Eventos

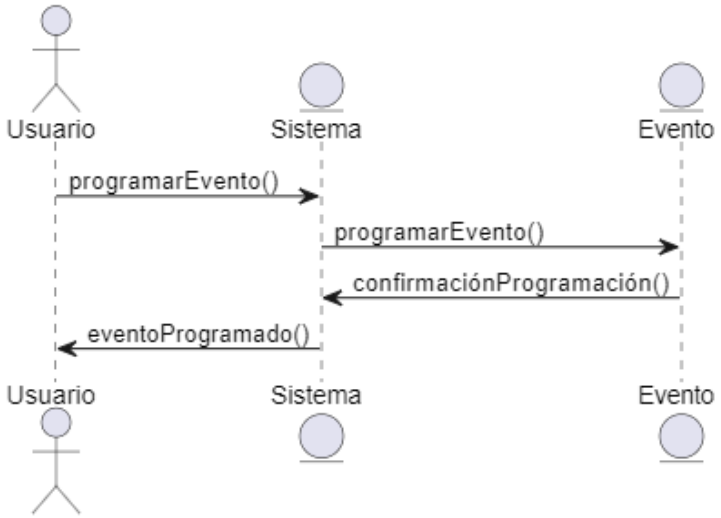

Ilustración 22: Actualización de Clientes

5.2.5.5 Programación de Eventos

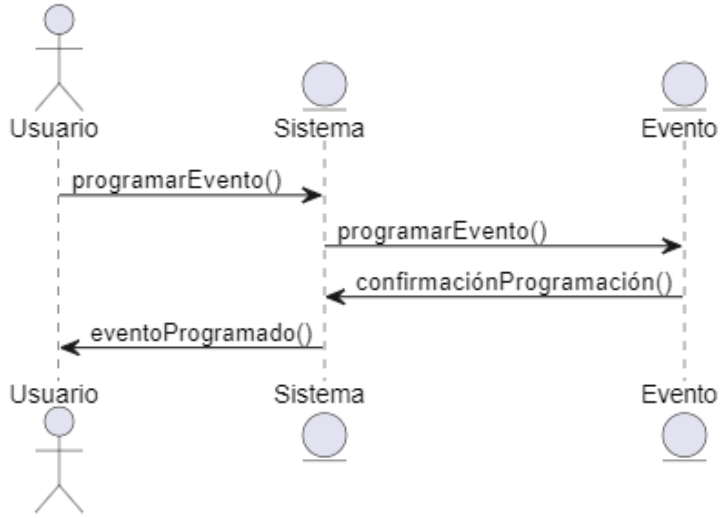

Ilustración 23: Programación de Eventos

5.2.5.6 Eliminación de Documentos

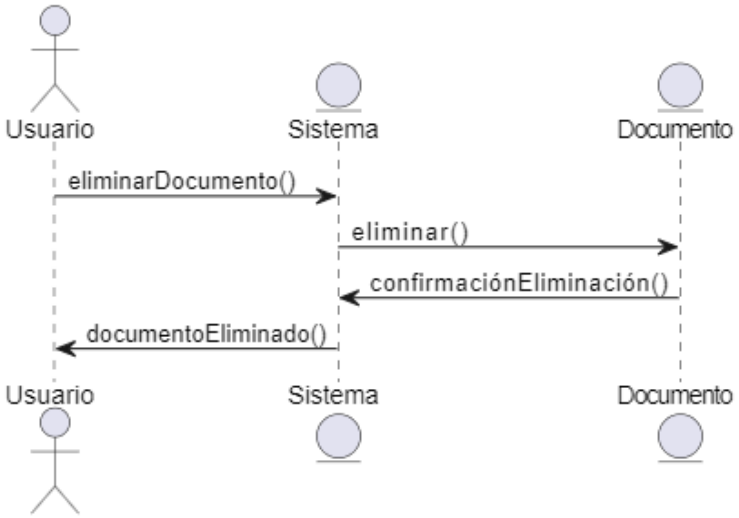

Ilustración 24: Eliminación de Documentos

5.2.5.7 Actualización de Clientes

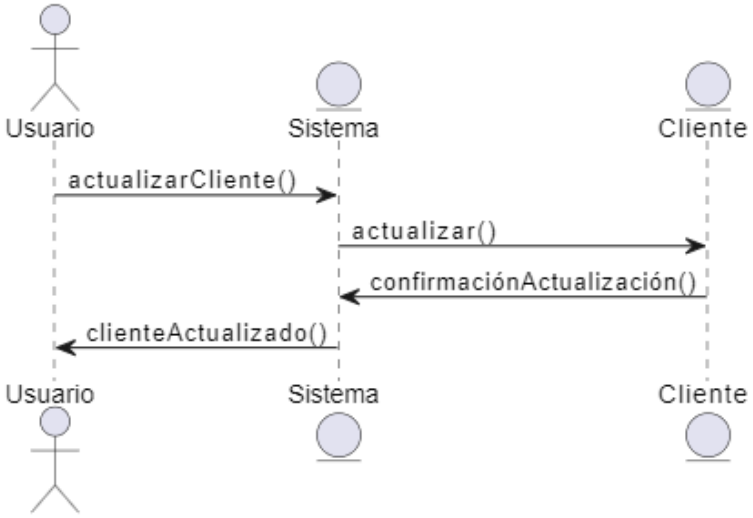

Ilustración 25: Actualización de Clientes

5.2.5.8 Cierre de Expedientes

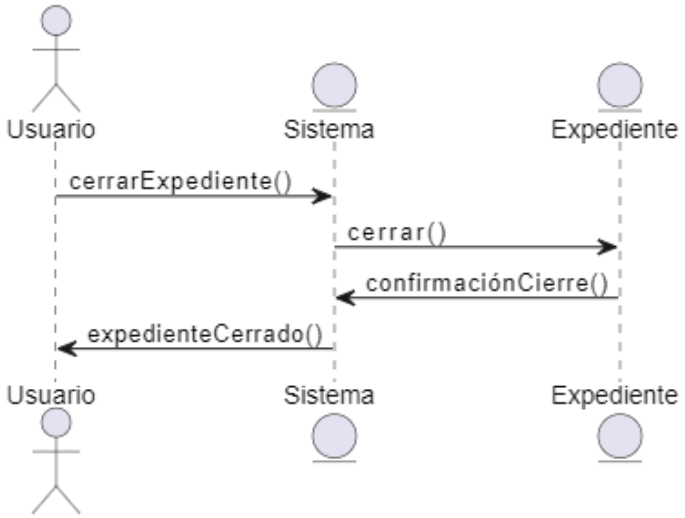

Ilustración 26: Cierre de Expedientes

5.2.5.9 Cancelación de eventos

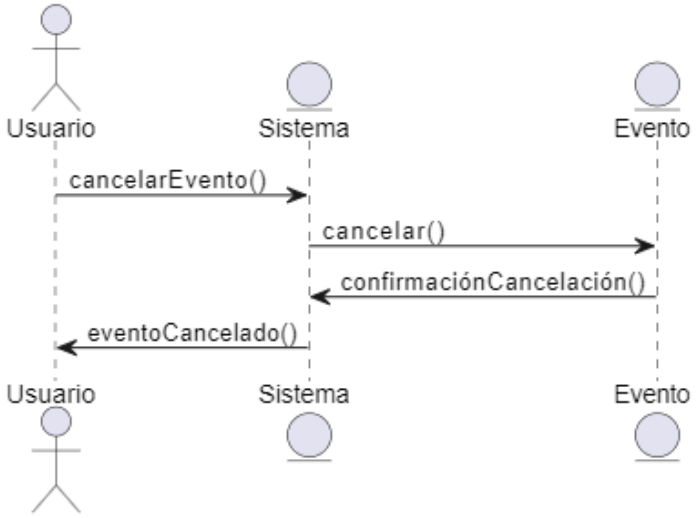

Ilustración 27: Cancelación de Eventos

5.2.5.10 Gestión de contraseñas

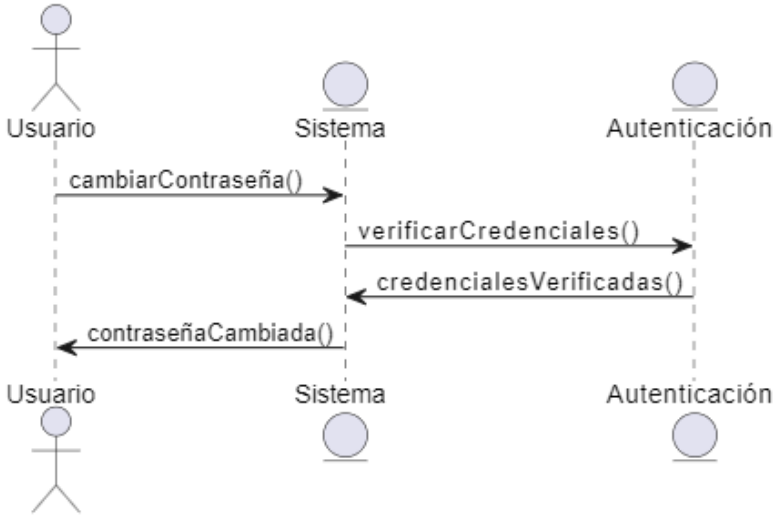

Ilustración 28: Gestión de Contraseñas

5.2.5.11 Recuperación de Contraseñas

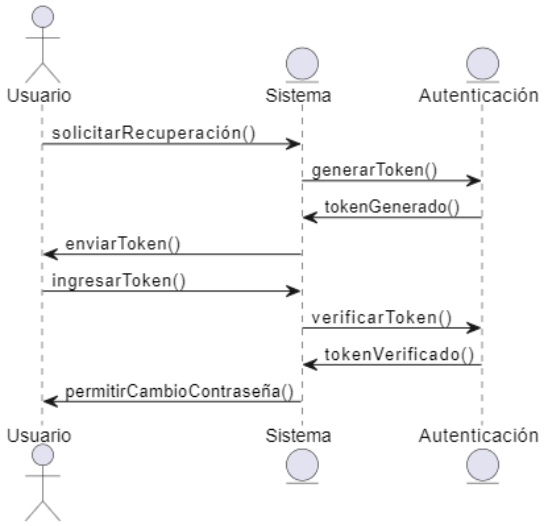

Ilustración 29: Recuperación de Contraseñas

5.2.5.12 Asignación de Permisos

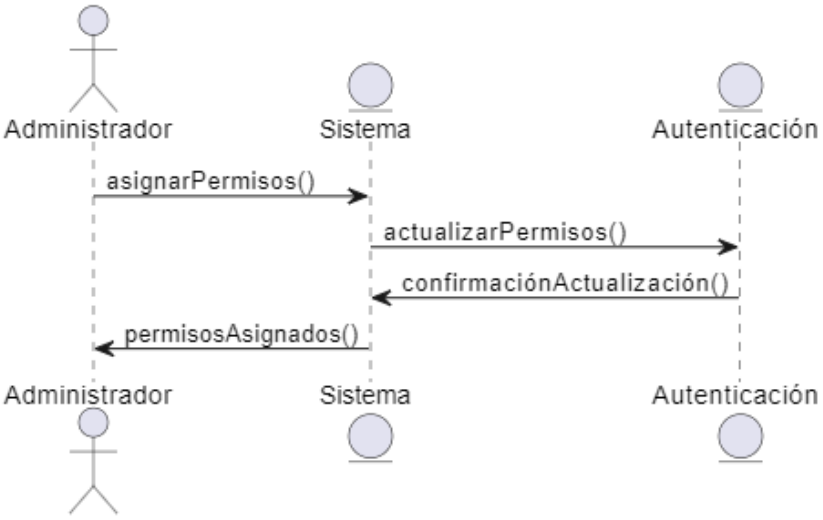

Ilustración 30: Asignación de Permisos

5.2.5.13 Notificación de Eventos

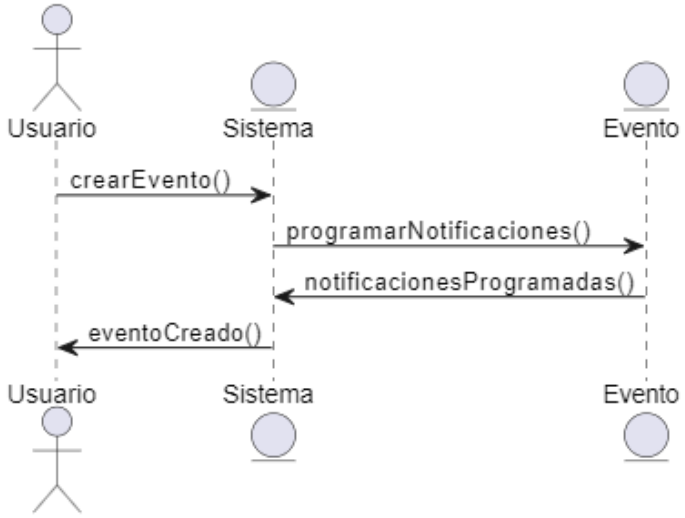

Ilustración 31: Notificación de Eventos

5.2.5.14 Consulta de documentos asociados a un expediente

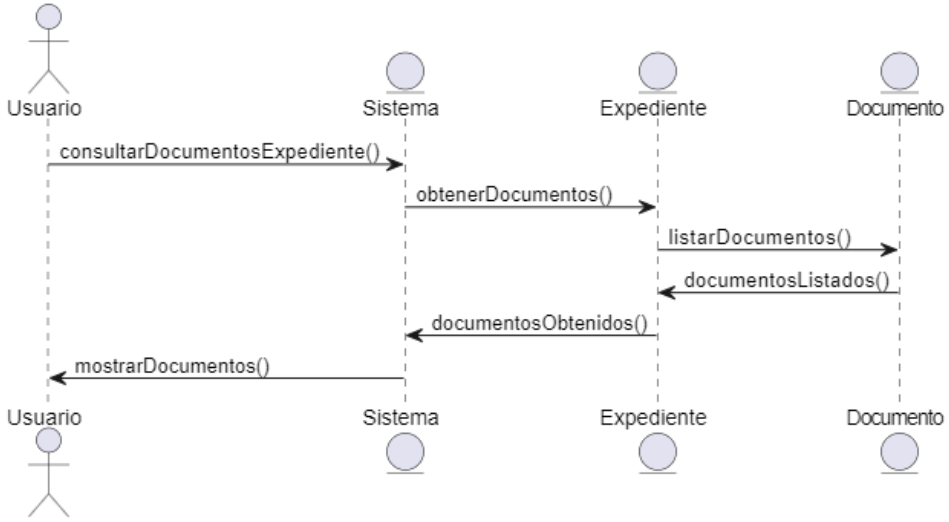

Ilustración 32: Consulta de documentos asociados a un expediente

5.2.6 Diagrama de secuencia

5.2.6.1 Inicio de Sesión

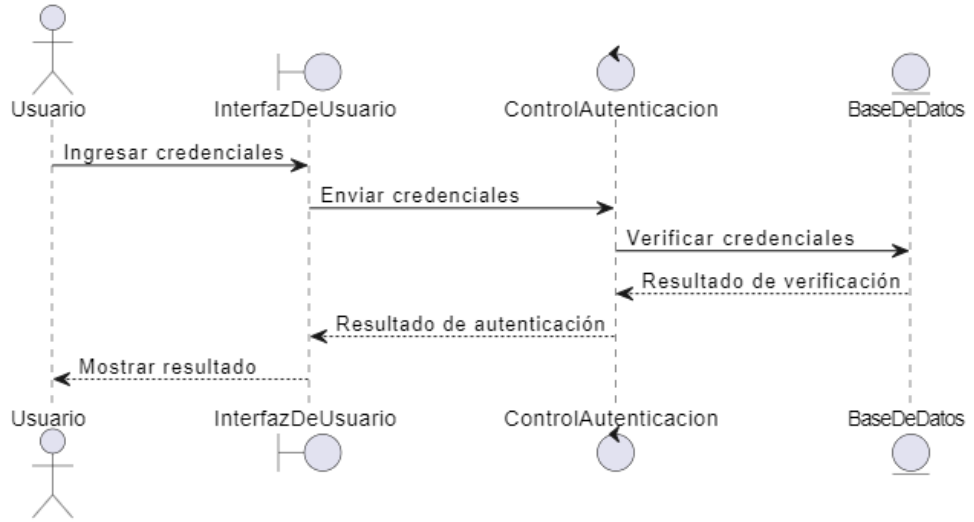

Ilustración 33: Inicio de Sesión

5.2.6.2 Gestión de documentos

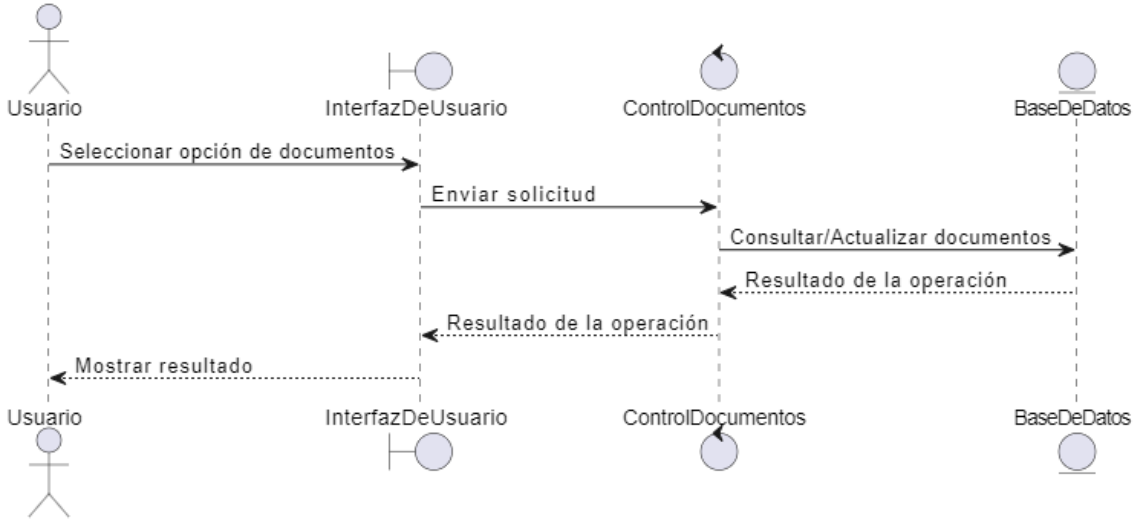

Ilustración 34: Gestión de Documentos

5.2.6.3 Gestión de Clientes

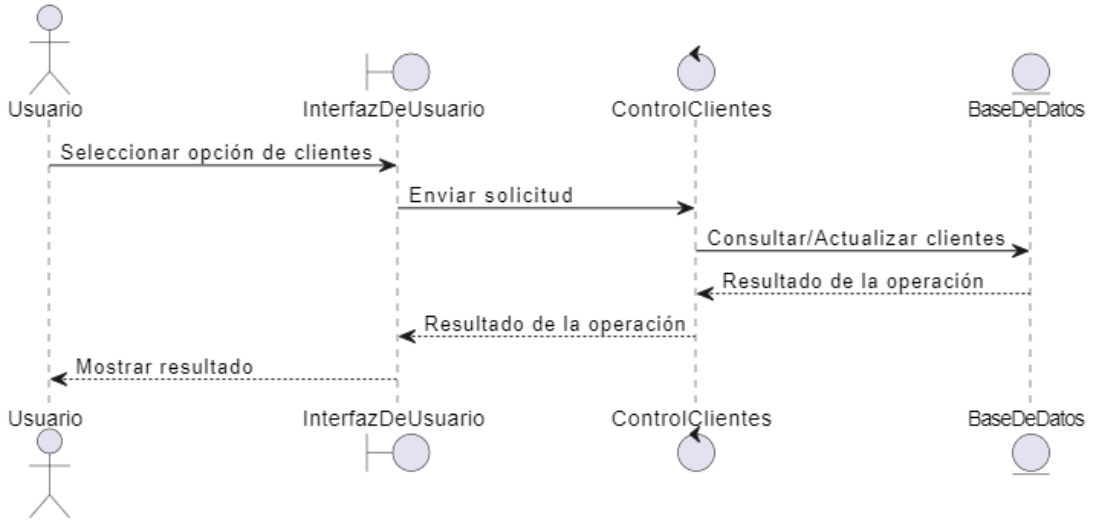

Ilustración 35: Gestión de Clientes

5.2.6.4 Gestión de Expedientes

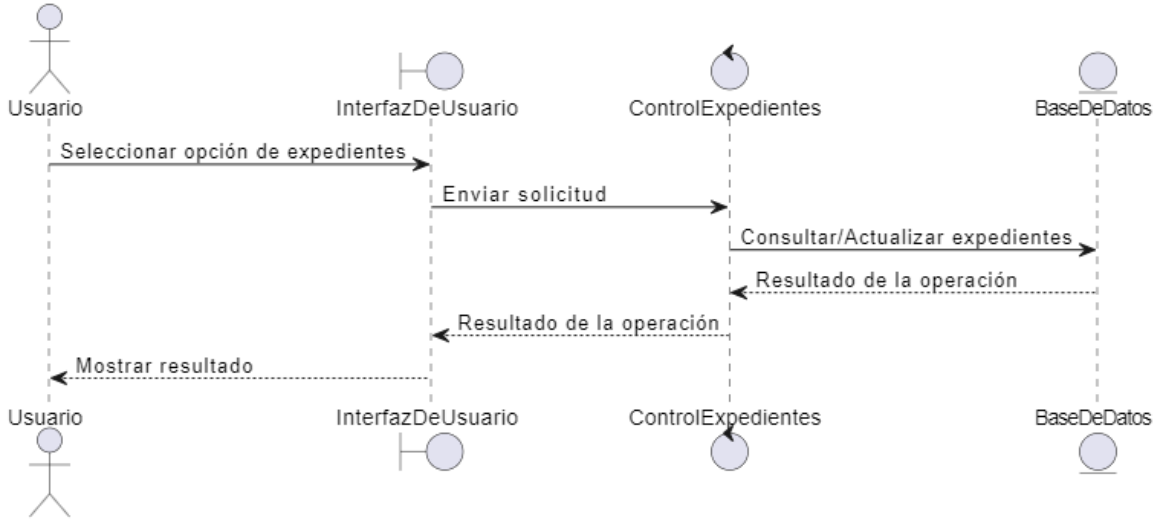

Ilustración 36: Gestión de Expedientes

5.2.6.5 Gestión de Eventos

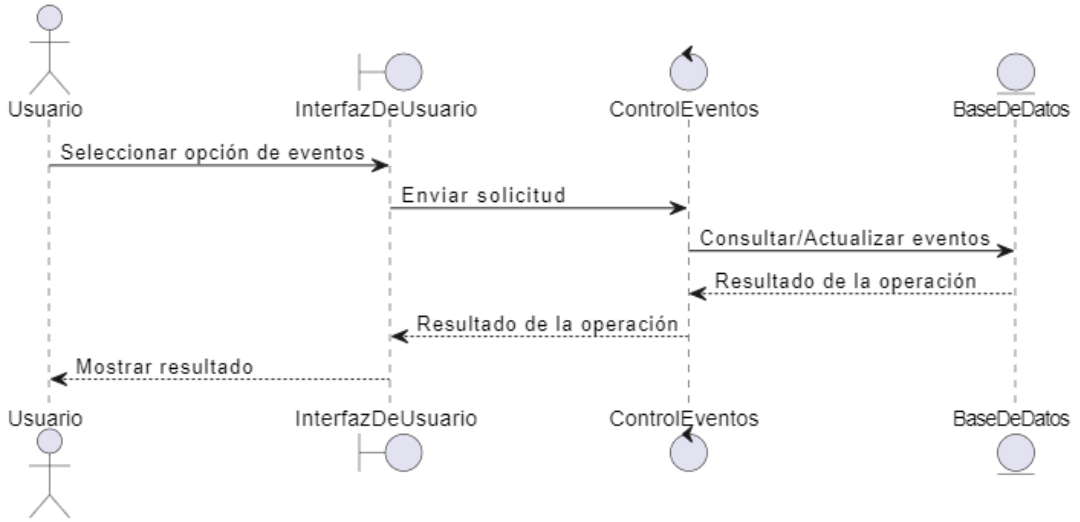

Ilustración 37: Gestión de Eventos

5.2.6.6 Envío de Notificaciones

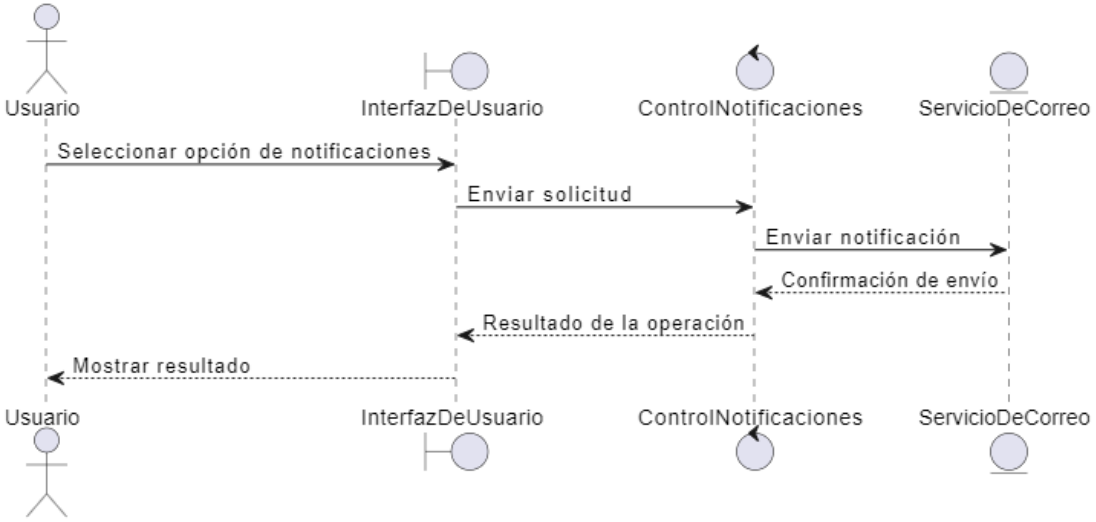

Ilustración 38: Gestión de Notificaciones

5.2.6.7 Registro de Usuario

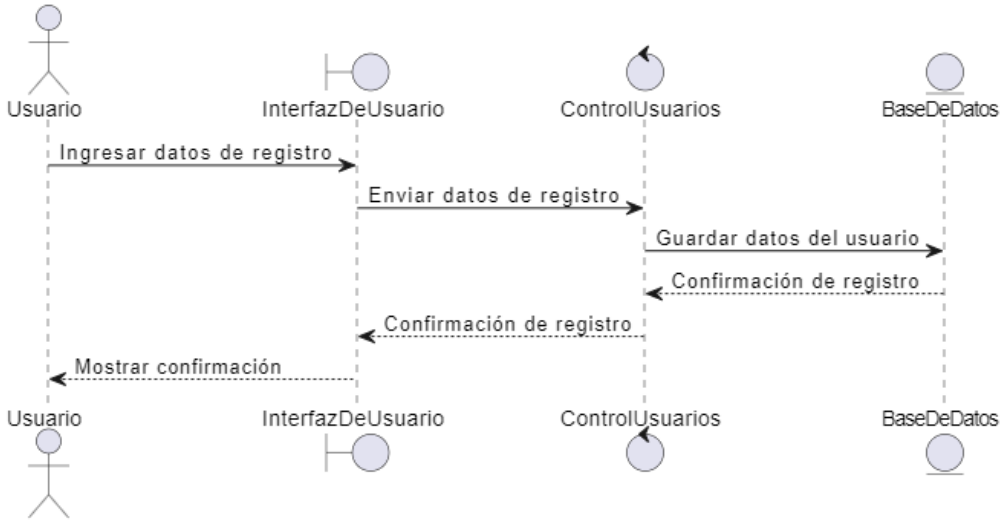

Ilustración 39: Registro de Usuario

5.2.6.8 Actualización de Perfil de Usuario

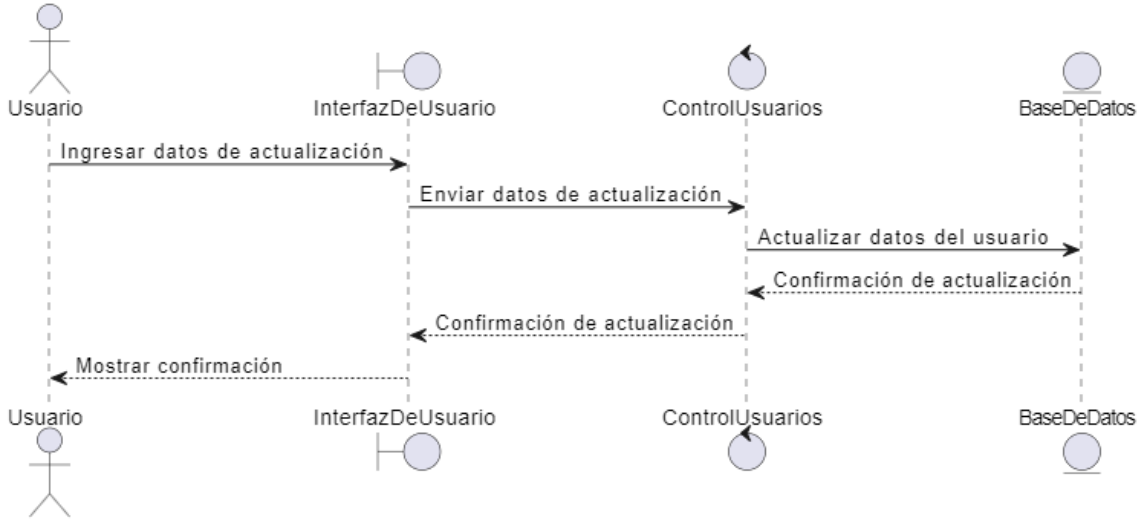

Ilustración 40: Actualización de Perfil de Usuario

5.2.6.9 Generación de Reportes

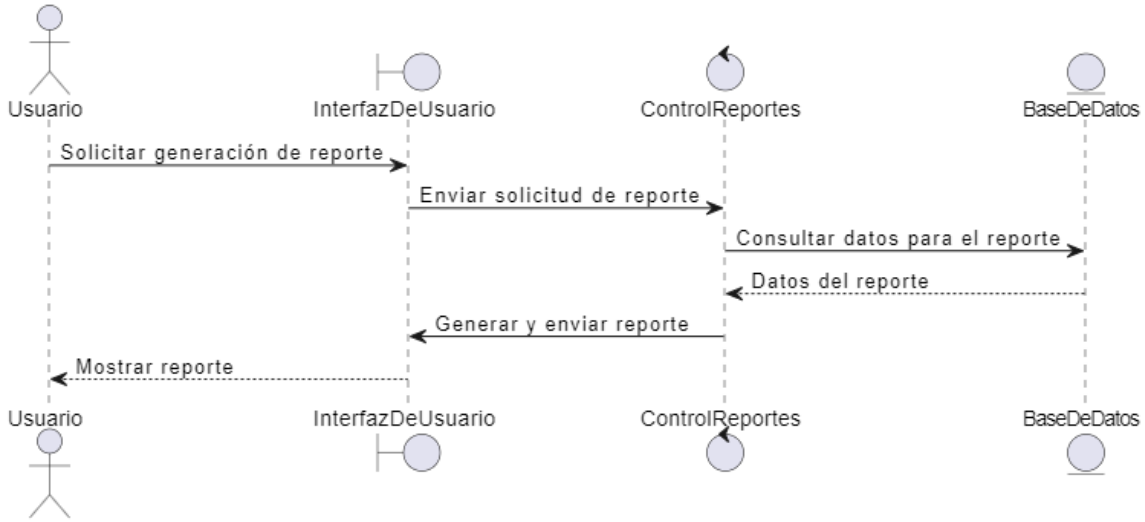

Ilustración 41: Generación de Reportes

5.2.6.10 Programación de Reuniones

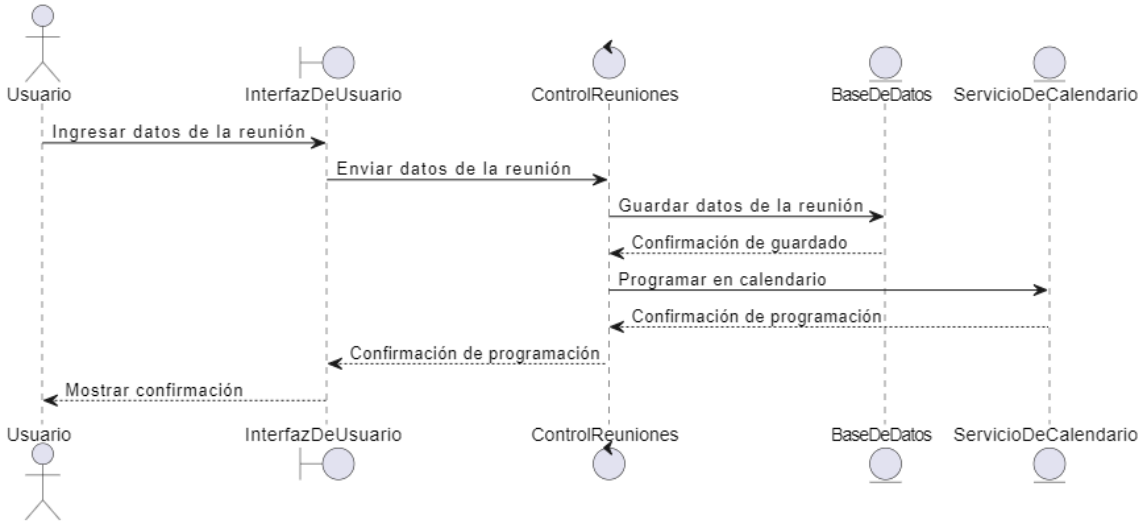

Ilustración 42: Programación de Reuniones

5.2.6.11 Envío de Recordatorios

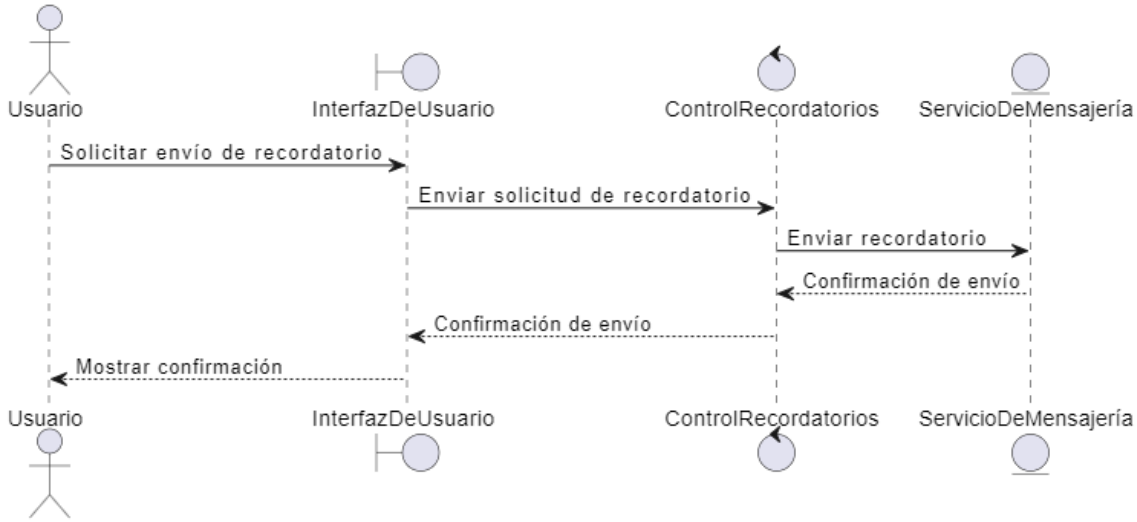

Ilustración 43: Envío de Recordatorios

5.3 Pseudocódigo

5.3.1 Módulos:

5.3.1.1 Módulo de Gestión de Usuarios

```
// Método para iniciar sesión
Método iniciarSesion(correo, contraseña) {
  autenticado = Autenticacion.verificarCredenciales(correo, contraseña)
  Si (autenticado) {
    usuario = obtenerUsuarioPorCorreo(correo)
    Autenticacion.asignarPermisos(usuario, usuario.rol)
    registrarActividad(usuario, "Inicio de sesión")
    mostrarPanelControl(usuario)
  } sino {
    mostrarMensajeError("Credenciales inválidas")
  }
}
```

```
// Método para cerrar sesión
Método cerrarSesion(usuario) {
  registrarActividad(usuario, "Cierre de sesión")
  limpiarSesion(usuario)
  mostrarPaginaInicio()
}

// Método para gestionar la cuenta del usuario
Método gestionarCuenta(usuario) {
  opciones = obtenerOpcionesCuenta(usuario)
  mostrarOpcionesCuenta(opciones)
}
```

5.3.1.2 Módulo de Gestión de Documentos

```
// Método para crear un documento
Método crearDocumento(datosDocumento) {
  validarDatos(datosDocumento)
  guardarEnBaseDeDatos(datosDocumento)
  registrarActividad(usuario, "Documento creado: " + datosDocumento.nombre)
  mostrarConfirmacion("Documento creado")
}

// Método para editar un documento
Método editarDocumento(idDocumento, nuevosDatos) {
  validarDatos(nuevosDatos)
  documento = obtenerDocumentoPorId(idDocumento)
```

```
  actualizarEnBaseDeDatos(idDocumento, nuevosDatos)
  registrarActividad(usuario, "Documento editado: " + documento.nombre)
  mostrarConfirmacion("Documento editado")
}

// Método para eliminar un documento
Método eliminarDocumento(idDocumento) {
  documento = obtenerDocumentoPorId(idDocumento)
  eliminarDeBaseDeDatos(idDocumento)
  registrarActividad(usuario, "Documento eliminado: " + documento.nombre)
  mostrarConfirmacion("Documento eliminado")
}

// Método para versionar un documento
Método versionarDocumento(idDocumento) {
  documento = obtenerDocumentoPorId(idDocumento)
  nuevaVersion = crearNuevaVersion(documento)
  guardarEnBaseDeDatos(nuevaVersion)
  registrarActividad(usuario, "Documento versionado: " + documento.nombre)
  mostrarConfirmacion("Documento versionado")
}
```

5.3.1.3 Módulo de Gestión de Clientes

```
// Método para registrar un cliente
Método registrarCliente(datosCliente) {
  validarDatos(datosCliente)
  guardarEnBaseDeDatos(datosCliente)
```

```
  registrarActividad(usuario, "Cliente registrado: " + datosCliente.nombre)
  mostrarConfirmacion("Cliente registrado")
}

// Método para actualizar un cliente
Método actualizarCliente(idCliente, nuevosDatos) {
  validarDatos(nuevosDatos)
  cliente = obtenerClientePorId(idCliente)
  actualizarEnBaseDeDatos(idCliente, nuevosDatos)
  registrarActividad(usuario, "Cliente actualizado: " + cliente.nombre)
  mostrarConfirmacion("Cliente actualizado")
}

// Método para eliminar un cliente
Método eliminarCliente(idCliente) {
  cliente = obtenerClientePorId(idCliente)
  eliminarDeBaseDeDatos(idCliente)
  registrarActividad(usuario, "Cliente eliminado: " + cliente.nombre)
  mostrarConfirmacion("Cliente eliminado")
}
```

5.3.1.4 Módulo de Gestión de Expedientes

```
// Método para crear un expediente
Método crearExpediente(datosExpediente) {
  validarDatos(datosExpediente)
  guardarEnBaseDeDatos(datosExpediente)
  registrarActividad(usuario, "Expediente creado: " + datosExpediente.caso)
```

```
  mostrarConfirmacion("Expediente creado")
}

// Método para actualizar el estado de un expediente
Método actualizarEstadoExpediente(idExpediente, nuevoEstado) {
  expediente = obtenerExpedientePorId(idExpediente)
  actualizarEnBaseDeDatos(idExpediente, nuevoEstado)
  registrarActividad(usuario, "Estado del expediente actualizado: " + expediente.caso)
  mostrarConfirmacion("Estado del expediente actualizado")
}

// Método para cerrar un expediente
Método cerrarExpediente(idExpediente) {
  expediente = obtenerExpedientePorId(idExpediente)
  cerrarEnBaseDeDatos(idExpediente)
  registrarActividad(usuario, "Expediente cerrado: " + expediente.caso)
  mostrarConfirmacion("Expediente cerrado")
}
```

5.3.1.5 Módulo de Gestión de Eventos

```
// Método para programar un evento
Método programarEvento(datosEvento) {
  validarDatos(datosEvento)
  guardarEnBaseDeDatos(datosEvento)
  registrarActividad(usuario, "Evento programado: " + datosEvento.tipoEvento)
  mostrarConfirmacion("Evento programado")
}
```

```
// Método para modificar un evento
Método modificarEvento(idEvento, nuevosDatos) {
  validarDatos(nuevosDatos)
  evento = obtenerEventoPorId(idEvento)
  actualizarEnBaseDeDatos(idEvento, nuevosDatos)
  registrarActividad(usuario, "Evento modificado: " + evento.tipoEvento)
  mostrarConfirmacion("Evento modificado")
}

// Método para cancelar un evento
Método cancelarEvento(idEvento) {
  evento = obtenerEventoPorId(idEvento)
  cancelarEnBaseDeDatos(idEvento)
  registrarActividad(usuario, "Evento cancelado: " + evento.tipoEvento)
  mostrarConfirmacion("Evento cancelado")
}
```

5.3.1.6 Módulo de Notificaciones

```
// Método para agregar un observador
Método agregarObservador(observador) {
  listaObservadores.agregar(observador)
}

// Método para eliminar un observador
Método eliminarObservador(observador) {
  listaObservadores.eliminar(observador)
```

```
}

// Método para notificar a los observadores
Método notificarObservadores() {
  Para cada observador en listaObservadores {
    observador.actualizar()
  }
}
```

5.3.1.7 Módulo de Historial de Actividades

```
// Método para registrar una actividad
Método registrarActividad(usuario, actividad) {
  datosActividad = {
    usuario: usuario.id,
    actividad: actividad,
    fecha: obtenerFechaActual()
  }
  guardarEnBaseDeDatos(datosActividad)
}

// Método para consultar el historial de actividades
Método consultarHistorial(usuario) {
  actividades = obtenerDeBaseDeDatos(usuario)
  mostrarHistorial(actividades)
}
```

5.3.1.8 Módulo de Roles

```
// Método para asignar un rol
Método asignarRol(usuario, rol) {
  usuario.rol = rol
  registrarActividad(usuario, "Rol asignado: " + rol)
  mostrarConfirmacion("Rol asignado")
}

// Método para revocar un rol
Método revocarRol(usuario) {
  rolAnterior = usuario.rol
  usuario.rol = null
  registrarActividad(usuario, "Rol revocado: " + rolAnterior)
  mostrarConfirmacion("Rol revocado")
}
```

5.3.2 Funciones:

5.3.2.1 Método: verificarCredenciales

```
Método verificarCredenciales(usuario, contraseña) {
  credenciales = obtenerCredencialesDeBaseDeDatos(usuario)
  Si (credenciales != null) {
    Si (credenciales.contraseña == contraseña) {
      retornar verdadero
    } sino {
      retornar falso
    }
```

```
  } sino {
    retornar falso
  }
}
```

5.3.2.2 Método: asignarPermisos

```
Método asignarPermisos(usuario, rol) {
  permisos = obtenerPermisosDeRol(rol)
  usuario.permisos = permisos
}
```

5.3.2.3 Método: iniciarSesion

```
Método iniciarSesion(correo, contraseña) {
  autenticado = Autenticacion.verificarCredenciales(correo, contraseña)
  Si (autenticado) {
    usuario = obtenerUsuarioPorCorreo(correo)
    Autenticacion.asignarPermisos(usuario, usuario.rol)
    registrarActividad(usuario, "Inicio de sesión")
    mostrarPanelControl(usuario)
  } sino {
    mostrarMensajeError("Credenciales inválidas")
  }
}
```

5.3.2.4 Método: cerrarSesion

```
Método cerrarSesion(usuario) {
```

```
  registrarActividad(usuario, "Cierre de sesión")
  limpiarSesion(usuario)
  mostrarPaginaInicio()
}
```

5.3.2.5 Método: gestionarCuenta

```
Método gestionarCuenta(usuario) {
  opciones = obtenerOpcionesCuenta(usuario)
  mostrarOpcionesCuenta(opciones)
}
```

5.3.2.6 Método: crearDocumento

```
Método crearDocumento(datosDocumento) {
  validarDatos(datosDocumento)
  guardarEnBaseDeDatos(datosDocumento)
  registrarActividad(usuario, "Documento creado: " + datosDocumento.nombre)
  mostrarConfirmacion("Documento creado")
}
```

5.3.2.7 Método: editarDocumento

```
Método editarDocumento(idDocumento, nuevosDatos) {
  validarDatos(nuevosDatos)
  documento = obtenerDocumentoPorId(idDocumento)
  actualizarEnBaseDeDatos(idDocumento, nuevosDatos)
  registrarActividad(usuario, "Documento editado: " + documento.nombre)
  mostrarConfirmacion("Documento editado")
```

```
}
```

5.3.2.8 Método: eliminarDocumento

```
Método eliminarDocumento(idDocumento) {
  documento = obtenerDocumentoPorId(idDocumento)
  eliminarDeBaseDeDatos(idDocumento)
  registrarActividad(usuario, "Documento eliminado: " + documento.nombre)
  mostrarConfirmacion("Documento eliminado")
}
```

5.3.2.9 Método: registrarCliente

```
Método registrarCliente(datosCliente) {
  validarDatos(datosCliente)
  guardarEnBaseDeDatos(datosCliente)
  registrarActividad(usuario, "Cliente registrado: " + datosCliente.nombre)
  mostrarConfirmacion("Cliente registrado")
}
```

5.3.2.10 Método: actualizarCliente

```
Método actualizarCliente(idCliente, nuevosDatos) {
  validarDatos(nuevosDatos)
  cliente = obtenerClientePorId(idCliente)
  actualizarEnBaseDeDatos(idCliente, nuevosDatos)
  registrarActividad(usuario, "Cliente actualizado: " + cliente.nombre)
  mostrarConfirmacion("Cliente actualizado")
}
```

5.3.2.11 Método: eliminarCliente

```
Método eliminarCliente(idCliente) {
  cliente = obtenerClientePorId(idCliente)
  eliminarDeBaseDeDatos(idCliente)
  registrarActividad(usuario, "Cliente eliminado: " + cliente.nombre)
  mostrarConfirmacion("Cliente eliminado")
}
```

5.3.2.12 Método: crearExpediente

```
Método crearExpediente(datosExpediente) {
  validarDatos(datosExpediente)
  guardarEnBaseDeDatos(datosExpediente)
  registrarActividad(usuario, "Expediente creado: " + datosExpediente.caso)
  mostrarConfirmacion("Expediente creado")
}
```

5.3.2.13 Método: actualizarEstadoExpediente

```
Método actualizarEstadoExpediente(idExpediente, nuevoEstado) {
  expediente = obtenerExpedientePorId(idExpediente)
  actualizarEnBaseDeDatos(idExpediente, nuevoEstado)
  registrarActividad(usuario, "Estado del expediente actualizado: " + expediente.caso)
  mostrarConfirmacion("Estado del expediente actualizado")
}
```

5.3.2.14 Método: cerrarExpediente

```
Método cerrarExpediente(idExpediente) {
```

```
  expediente = obtenerExpedientePorId(idExpediente)
  cerrarEnBaseDeDatos(idExpediente)
  registrarActividad(usuario, "Expediente cerrado: " + expediente.caso)
  mostrarConfirmacion("Expediente cerrado")
}
```

5.3.2.15 Método: programarEvento

```
Método programarEvento(datosEvento) {
  validarDatos(datosEvento)
  guardarEnBaseDeDatos(datosEvento)
  registrarActividad(usuario, "Evento programado: " + datosEvento.tipoEvento)
  mostrarConfirmacion("Evento programado")
}
```

5.3.2.16 Método: cancelarEvento

```
Método cancelarEvento(idEvento) {
  evento = obtenerEventoPorId(idEvento)
  cancelarEnBaseDeDatos(idEvento)
  registrarActividad(usuario, "Evento cancelado: " + evento.tipoEvento)
  mostrarConfirmacion("Evento cancelado")
}
```

5.3.2.17 Método: agregarObservador

```
Método agregarObservador(observador) {
  listaObservadores.agregar(observador)
}
```

5.3.2.18 Método: eliminarObservador

```
Método eliminarObservador(observador) {
  listaObservadores.eliminar(observador)
}
```

5.3.2.19 Método: notificarObservadores

```
Método notificarObservadores() {
  Para cada observador en listaObservadores {
    observador.actualizar()
  }
}
```

5.3.2.20 Método: registrarActividad

```
Método registrarActividad(usuario, actividad) {
  datosActividad = {
    usuario: usuario.id,
    actividad: actividad,
    fecha: obtenerFechaActual()
  }
  guardarEnBaseDeDatos(datosActividad)
}
```

5.3.2.21 Método: consultarHistorial

```
Método consultarHistorial(usuario) {
  actividades = obtenerDeBaseDeDatos(usuario)
  mostrarHistorial(actividades)
```

```
}
```

5.3.2.22 Método: asignarRol

```
Método asignarRol(usuario, rol) {
  usuario.rol = rol
  registrarActividad(usuario, "Rol asignado: " + rol)
  mostrarConfirmacion("Rol asignado")
}
```

5.3.2.23 Método: revocarRol

```
Método revocarRol(usuario) {
  rolAnterior = usuario.rol
  usuario.rol = null
  registrarActividad(usuario, "Rol revocado: " + rolAnterior)
  mostrarConfirmacion("Rol revocado")
}
```

5.4 Modelo de Datos

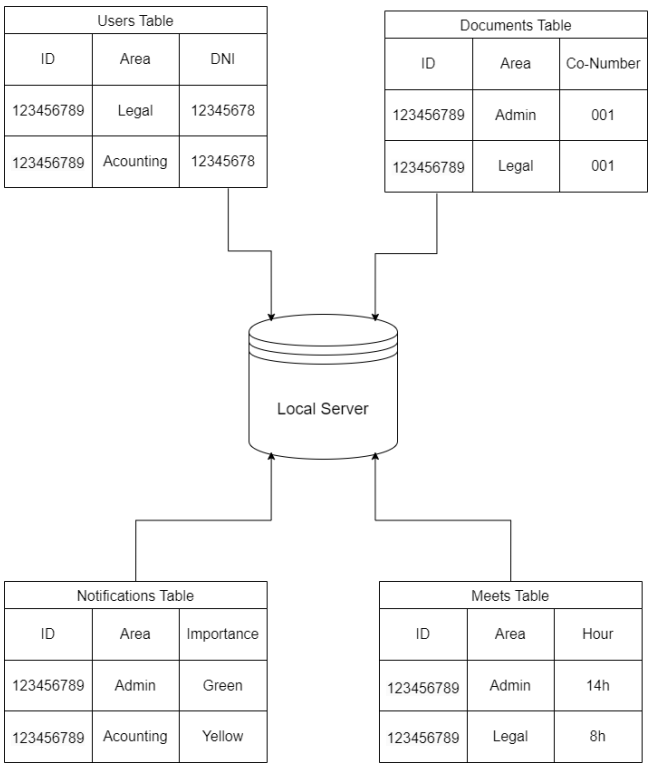

Ilustración 44: Modelo de Datos General

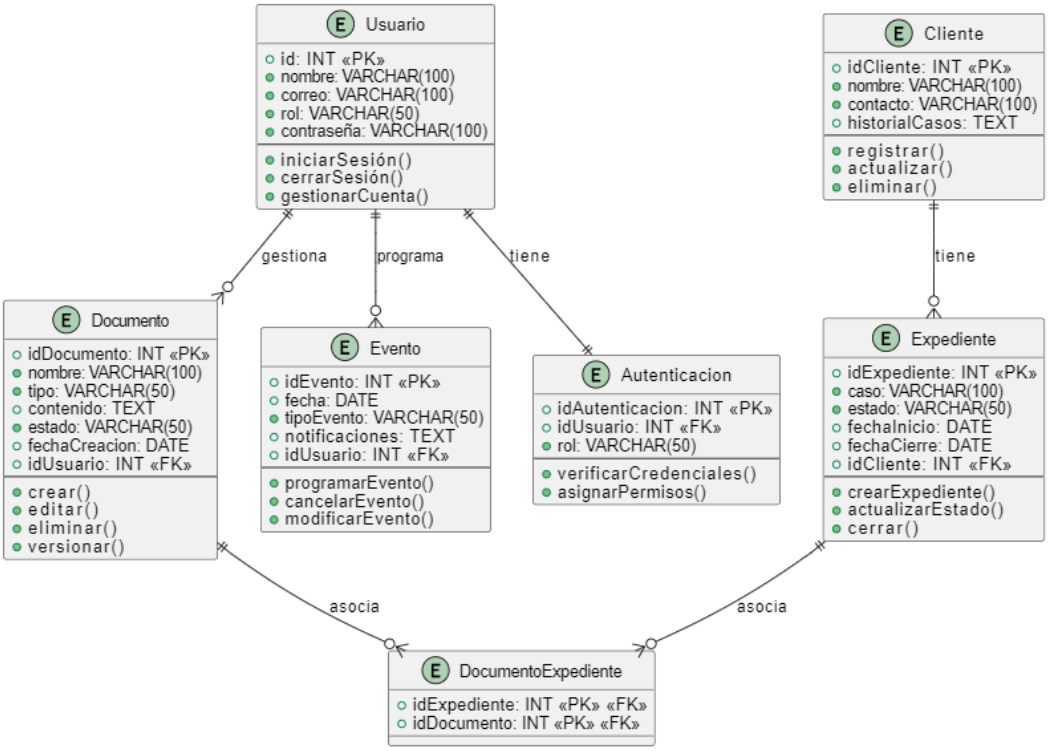

Ilustración 45: Modelo de Datos Detallado

5.5 Patrones de Diseño

5.5.1 Patrones creacionales

Factory Method Pattern (Creacional):

Documento: Interfaz que define los métodos crear(), editar(), eliminar() y versionar().

DocumentoTexto y DocumentoPDF: Implementaciones concretas de la interfaz Documento.

CreadorDocumento: Clase abstracta que define el método crearDocumento().

CreadorDocumentoConcreto: Implementación concreta de CreadorDocumento que crea instancias de DocumentoTexto o DocumentoPDF según el tipo.

5.5.2 Patrones estructurales

Adapter Pattern (Estructural):

Cliente: Clase que solicita documentos.

DocumentoAdapter: Interfaz que define el método procesarDocumento().

DocumentoTextoAdapter y DocumentoPDFAdapter: Implementaciones concretas de DocumentoAdapter que adaptan DocumentoTexto y DocumentoPDF respectivamente.

5.5.3 Patrones de comportamiento

Observer Pattern (Comportamiento):

Observador: Interfaz que define el método actualizar().

Usuario: Implementación concreta de Observador que incluye métodos adicionales como iniciarSesion(), cerrarSesion() y gestionarCuenta().

Notificacion: Clase que mantiene una lista de observadores y los notifica cuando hay cambios.

Relaciones adicionales:

CreadorDocumentoConcreto crea instancias de Documento.

Cliente utiliza CreadorDocumentoConcreto para crear documentos.

Cliente utiliza DocumentoAdapter para procesar documentos.

Clases adicionales:

Expediente: Clase que define los métodos crearExpediente(), actualizarEstado() y cerrar().

Evento: Clase que define los métodos programarEvento(), cancelarEvento() y modificarEvento().

Autenticacion: Clase que define los métodos verificarCredenciales() y asignarPermisos().

Historial: Clase que define los métodos registrarActividad() y consultarHistorial().

Rol: Clase que define los métodos asignarRol() y revocarRol().

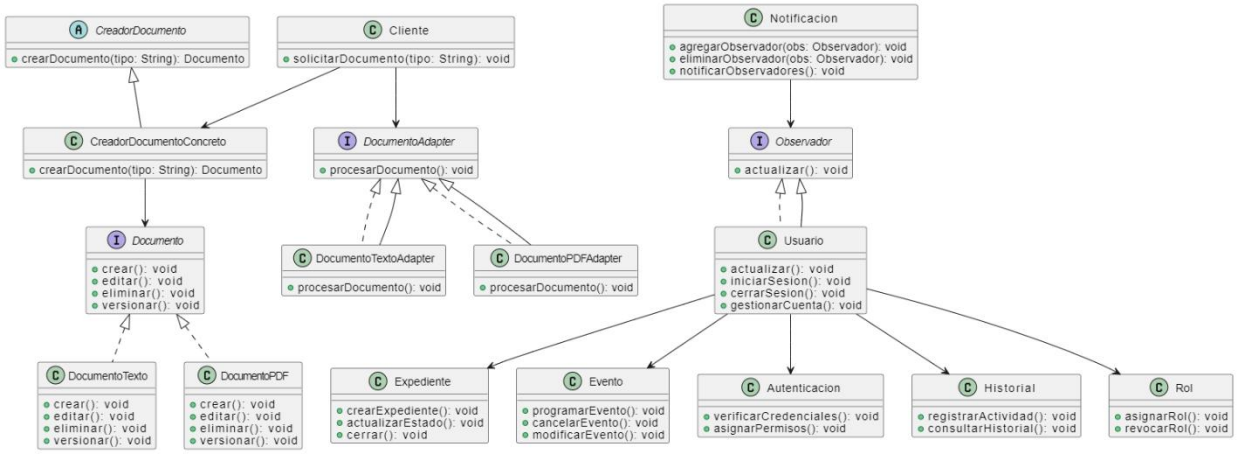

Ilustración 46: Diagrama de clases con Patrones de Diseño

5.6 Diseño de Interfaces de Usuario

Inicio de Sesión

Ilustración 47: Vista de Inicio de Sesión

Vista de Perfil

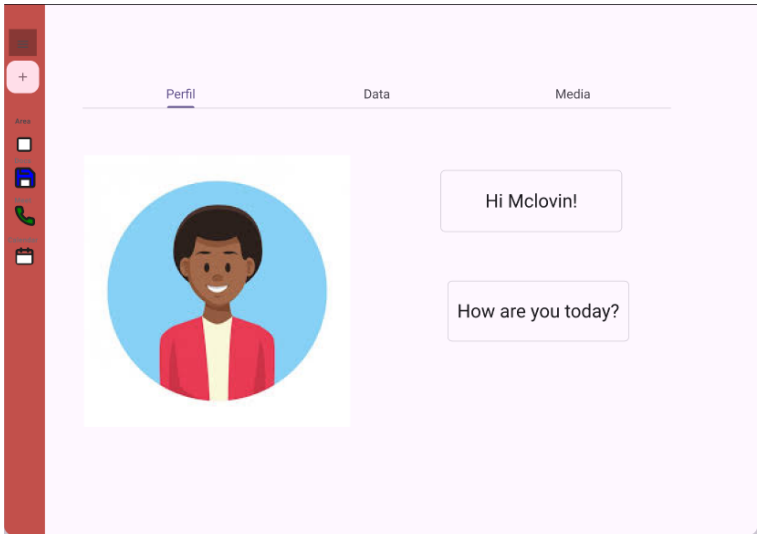

Ilustración 48: Vista de Perfil

Vista de Áreas

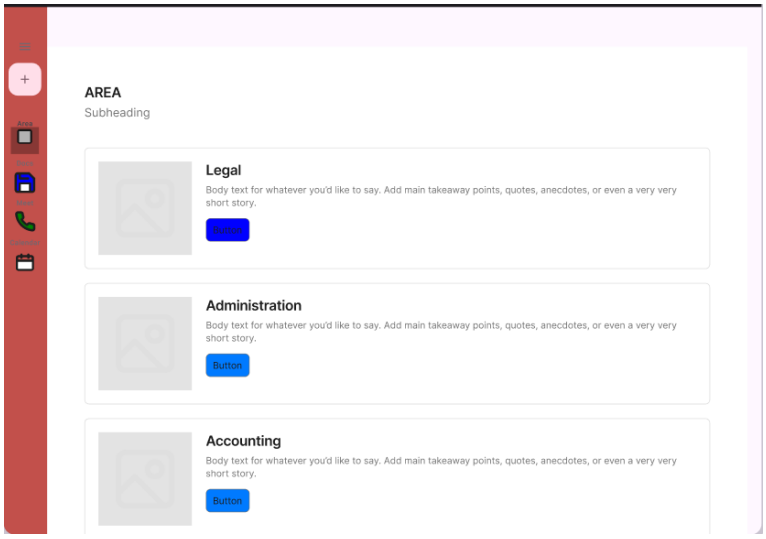

Ilustración 49: Vista de Áreas

Vista de Documentos

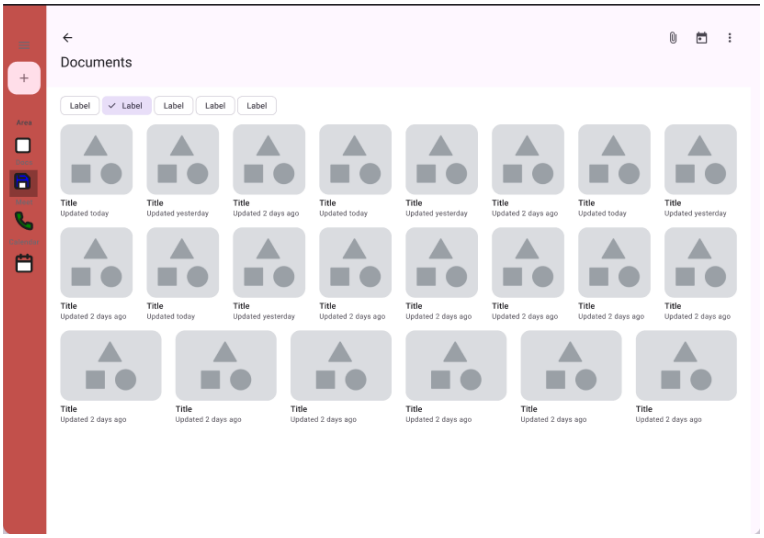

Ilustración 50: Vista de Documentos

Vista de Meets

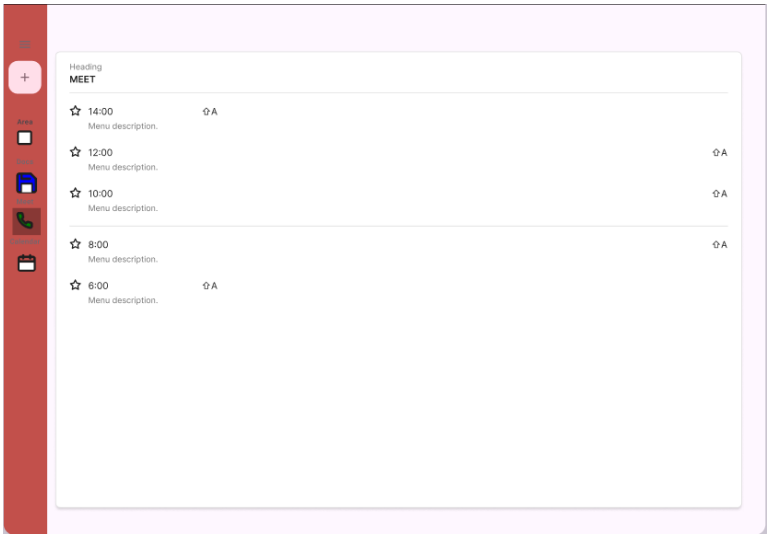

Ilustración 51: Vista de Meets

Vista de Eventos

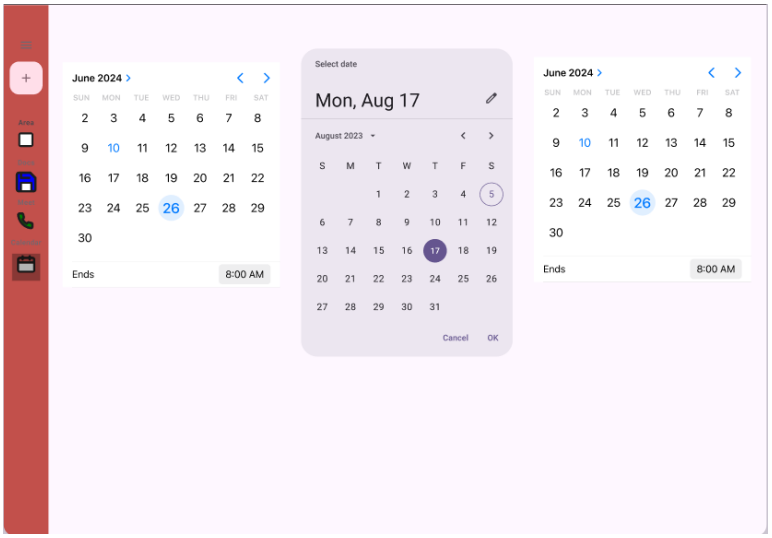

Ilustración 52: Vista de Eventos

5.7 Diseño detallado de Hardware

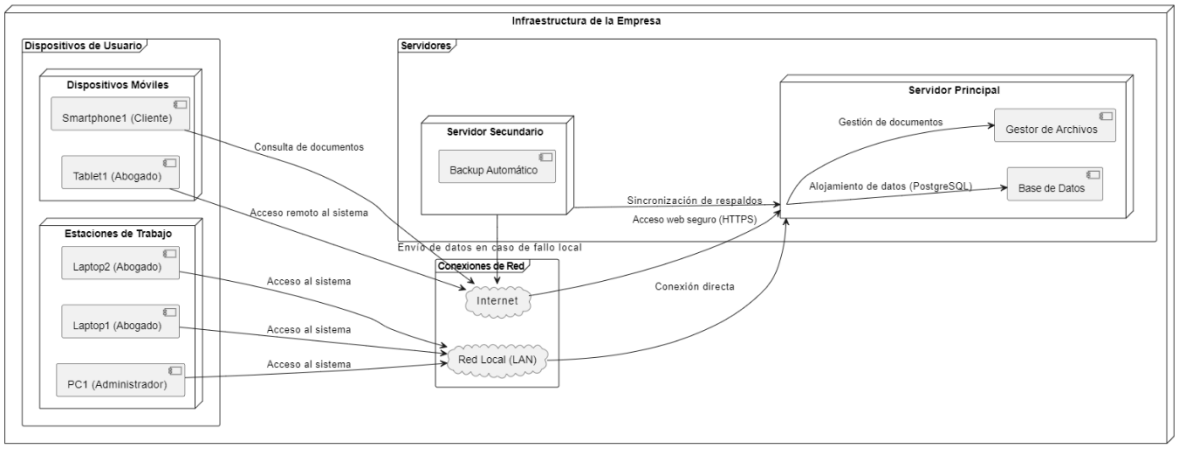

Ilustración 53: Diagrama detallado de Hardware

5.7.1 Especificaciones Técnicas de los Componentes:

1. Dispositivos de Usuario

Dispositivos Móviles:

Smartphone1 (Cliente):

Función principal: Permite a los clientes consultar documentos específicos de manera remota.

Especificaciones detalladas:

Sistema operativo: Android 10 o superior, iOS 13 o superior.

Almacenamiento: 64GB mínimo para manejo temporal de archivos descargados.

Seguridad:

- Autenticación biométrica (huella dactilar o reconocimiento facial).
- Integración con autenticación multifactor (MFA).
- Encriptación de datos local (AES-256).
- Aplicación dedicada o acceso web responsivo.
- Soporte de conexiones VPN para una capa adicional de seguridad.

Tablet1 (Abogado):

Función principal: Gestión de documentos más avanzada y visualización en mayor formato.

Especificaciones detalladas:

- Resolución mínima: 1920x1080 píxeles.
- Compatibilidad con stylus (para anotaciones en documentos).

Sistema operativo: Android o iPadOS.

Aplicación preinstalada para sincronización en tiempo real con el servidor principal.

Estaciones de Trabajo:

Laptop1 y Laptop2 (Abogados):

Funciones principales:

- Creación de nuevos documentos.
- Modificación de expedientes y acceso al sistema.

Especificaciones adicionales:

Tarjeta de red con soporte para velocidades gigabit (LAN y Wi-Fi 6).

Almacenamiento SSD (mínimo 512GB para rendimiento y almacenamiento local temporal).

Integración con suites de productividad (Microsoft Office, LibreOffice).

Software de protección antivirus empresarial.

PC1 (Administrador):

Funciones principales:

Supervisión, mantenimiento y configuración del sistema.

Especificaciones adicionales:

Doble monitor para multitarea (mínimo Full HD).

Herramientas de gestión de servidores y monitoreo:

p.ej., software como Nagios, Zabbix, o Windows Server Manager.

Almacenamiento ampliable: RAID 5 para copias locales y disponibilidad.

2. Servidores

Servidor Secundario:

Función avanzada: Respaldo automático con sincronización periódica y rol de contingencia ante fallos.

Especificaciones adicionales:

Almacenamiento:

Al menos 8TB en discos duros configurados en RAID 6 para máxima redundancia.

Almacenamiento híbrido (HDD para almacenamiento masivo y SSD para operaciones críticas).

Conectividad:

Dos interfaces de red redundantes (Ethernet 1Gbps o superior).

Acceso VPN para soporte remoto en caso de mantenimiento.

Software de respaldo:

Incremental y diferencial, con herramientas como Acronis Backup o Bacula.

Integración con soluciones en la nube (AWS S3 o Azure Blob Storage).

Sistemas operativos recomendados:

Linux (Ubuntu Server o CentOS) o Windows Server con roles de backup.

Servidor Principal:

Rol avanzado: Procesa solicitudes de usuario, gestiona documentos y aloja la base de datos principal.

Especificaciones adicionales:

Hardware:

Procesador: AMD EPYC o Intel Xeon (mínimo 16 núcleos).

Memoria RAM: 64GB ECC para garantizar estabilidad y capacidad multitarea.

Almacenamiento: RAID 10 con discos NVMe para rapidez y seguridad.

Software:

Virtualización con Docker o Kubernetes para contenedores.

Servidor web: Nginx o Apache configurado con HTTPS.

Base de Datos:

PostgreSQL con replicación en tiempo real.

Copias de seguridad programadas y auditorías habilitadas.

Seguridad:

Firewall empresarial (p. ej., pfSense o Cisco ASA).

Detección y respuesta ante amenazas (EDR).

3. Conexiones de Red

Internet:

Características avanzadas:

Ancho de banda mínimo de 200 Mbps simétricos.

ISP con redundancia (doble proveedor).

Seguridad de red perimetral:

Implementación de un Firewall con inspección profunda de paquetes (Deep Packet Inspection, DPI).

Certificado SSL/TLS para cifrar las conexiones HTTPS.

Monitorización de tráfico en tiempo real para identificar posibles ataques o anomalías.

Red Local (LAN):

Características avanzadas:

Switches administrables con VLANs segmentadas para separar usuarios internos (administradores, abogados) y externos (clientes).

Estándar Ethernet Cat6 para cableado estructurado, soportando velocidades de hasta 10Gbps.

PoE (Power over Ethernet) para dispositivos adicionales (cámaras, puntos de acceso).

Herramientas de diagnóstico y pruebas de conexión como Wireshark o SolarWinds.

4. Otros componentes importantes

Gestión de autenticación:

Active Directory o LDAP para gestionar roles y permisos de usuarios.

Autenticación multifactor (MFA) integrada con tokens de hardware (YubiKey) o software (Google Authenticator).

Auditoría y Logs:

Sistema centralizado de logs como Graylog o ELK Stack.

Registro detallado de accesos, modificaciones y eventos críticos.

Sistema de notificaciones:

Integración con servicios de correo (SMTP configurado).

Alertas móviles mediante Push o SMS.

5. Consideraciones adicionales

Escalabilidad:

Soporte para añadir nuevos dispositivos o servidores sin afectar el rendimiento.

Configuración de balanceo de carga mediante Nginx o HAProxy.

Pruebas de seguridad:

Análisis periódico de vulnerabilidades (p. ej., con Nessus o OpenVAS).

Simulaciones de ataques como pruebas de penetración (pentesting).

Anexos

Diccionario de Términos:

Usuario: Entidad que representa a una persona que interactúa con el sistema.

Atributos: id, nombre, correo, rol, contraseña.

Métodos: iniciarSesión(), cerrarSesión(), gestionarCuenta().

Documento: Entidad que representa un archivo o registro gestionado por el sistema.

Atributos: idDocumento, nombre, tipo, contenido, estado, fechaCreación.

Métodos: crear(), editar(), eliminar(), versionar().

Cliente: Entidad que representa a una persona o empresa que utiliza los servicios del sistema.

Atributos: idCliente, nombre, contacto, historialCasos.

Métodos: registrar(), actualizar(), eliminar().

Expediente: Entidad que representa un caso o conjunto de documentos relacionados con un cliente.

Atributos: idExpediente, caso, estado, fechaInicio, fechaCierre, documentosAsociados.

Métodos: crearExpediente(), actualizarEstado(), cerrar().

Evento: Entidad que representa una actividad programada en el sistema.

Atributos: idEvento, fecha, tipoEvento, notificaciones.

Métodos: programarEvento(), cancelarEvento(), modificarEvento().

Autenticación: Entidad que gestiona la verificación de credenciales y permisos de los usuarios.

Atributos: idAutenticación, usuario, rol.

Métodos: verificarCredenciales(), asignarPermisos().

Relaciones:

Usuario - Documento: Relación 1:N (Un usuario puede gestionar varios documentos).

Cliente - Expediente: Relación 1:N (Un cliente puede tener múltiples expedientes).

Expediente - Documento: Relación 1:N (Un expediente puede asociarse a varios documentos).

Usuario - Evento: Relación 1:N (Un usuario puede programar múltiples eventos).

Autenticación - Usuario: Relación 1:1 (Cada usuario tiene un registro de autenticación).

Estados:

Documento: Creado, En_Edición, Publicado, Versionado, Eliminado, Archivado, Revisado, Aprobado.

Expediente: Abierto, En_Proceso, Cerrado, Eliminado, Suspendido, En_Espera, Reabierto.

Evento: Programado, Modificado, Cancelado, Completado, En_Progreso.

Usuario: Registrado, Activo, Inactivo, Eliminado, Bloqueado, Verificado.

Acceso a Documentos: Solo los usuarios activos tienen acceso a los documentos.

Evidencia de Reunión

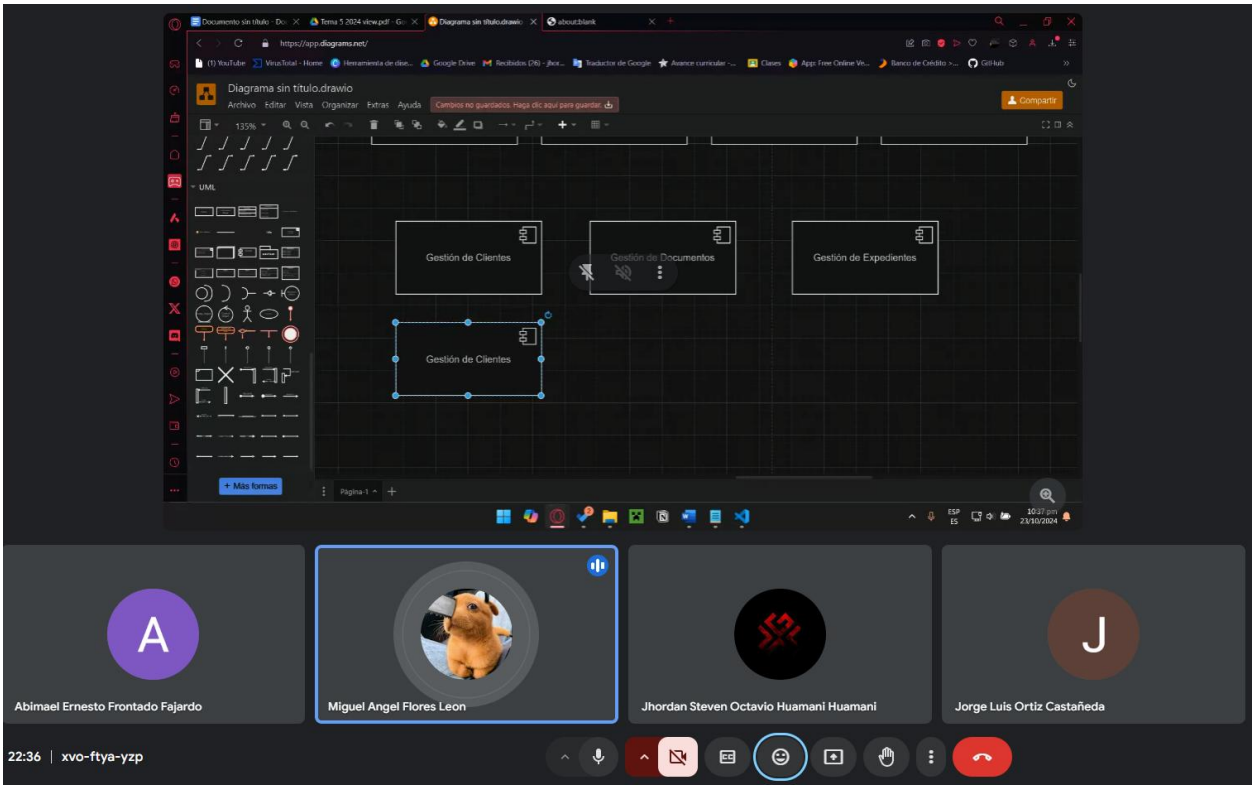

Ilustración 54: Reunión de meet

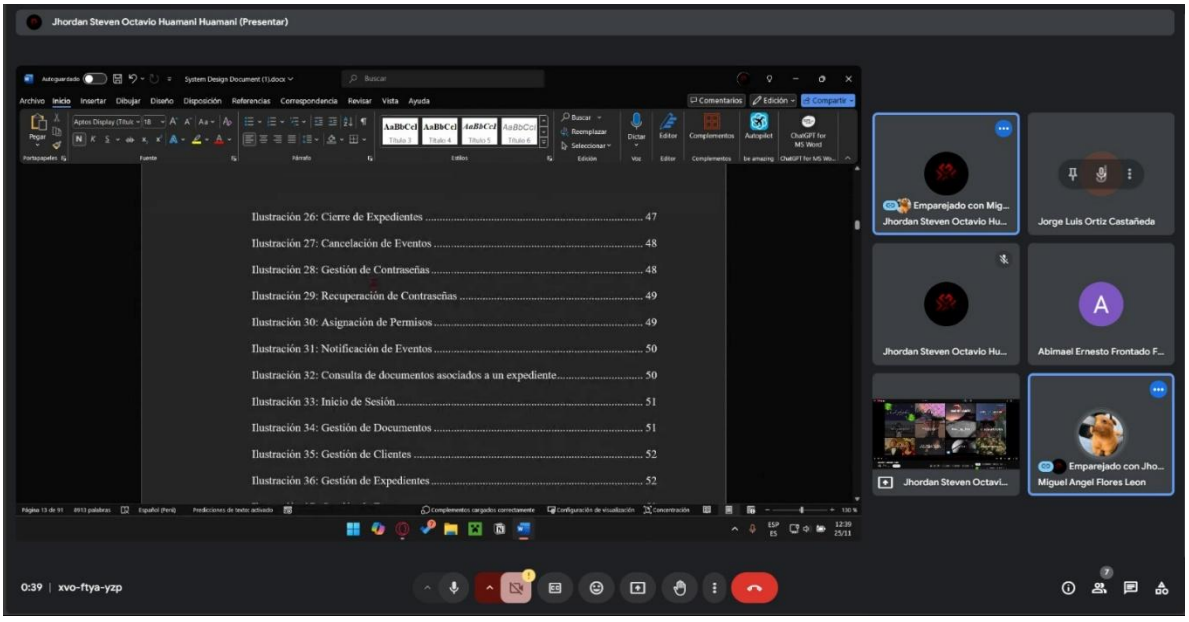

Ilustración 55: Evidencia Meet, avance parte 5