



INFORMATIKA  
FAKULTATEA  
FACULTAD  
DE INFORMÁTICA

## Trabajo de Fin de Grado

Grado en Ingeniería Informática

Ingeniería de Software

---

# **Sistema de gestión automatizada de documentos de procuradores - AutoProc**

---

*Unai Roa Cepeda*

### **Dirección**

Ekaitz Jauregi Iztueta  
Idoia Berges González

28 de octubre de 2024



# Agradecimientos

Antes que nada, me gustaría expresar mi más sincero agradecimiento a todas las personas que me han rodeado durante este proceso.

En primer lugar, agradezco a mis tutores, Idoia Berges y Ekaitz Jauregi, por acoger mi propuesta de proyecto y por su inestimable orientación y dedicación. Sus valiosos consejos y paciencia han sido fundamentales en este proceso.

Quiero agradecer también a esos compañeros que se han convertido en buenos amigos durante esta etapa académica. Han hecho de este proceso una experiencia increíble, de la que me llevo incontables recuerdos que nunca olvidaré.

Y, por supuesto, mi agradecimiento más especial es para mi familia, por su apoyo incondicional y su confianza constante, y para mis mejores amigos, por estar siempre a mi lado, motivándome y sacándome una sonrisa incluso en los momentos más difíciles, haciendo que este camino fuera mucho más llevadero.



# Resumen

Este proyecto se ha centrado en el desarrollo de una plataforma web integral para la gestión automatizada de documentos de procuradores, especialmente diseñada para los profesionales del País Vasco. La solución, denominada AutoProc, permite optimizar su trabajo diario mediante la automatización de tareas repetitivas, reduciendo errores derivados de la gestión manual de múltiples documentos.

La aplicación ha sido desarrollada en estrecha colaboración con un cliente real, lo que ha permitido adaptar el sistema a sus necesidades y requerimientos específicos. De esta manera, se ha logrado alinear las funcionalidades implementadas con los procesos de trabajo existentes.

El sistema ha sido desarrollado utilizando tecnologías modernas, priorizando la modularidad y la escalabilidad. La plataforma está diseñada para ser flexible, permitiendo su adaptación a nuevas necesidades y expandirse a otras regiones de España en el futuro. Además, se han incorporado medidas de seguridad para garantizar la protección de la información sensible y la confidencialidad de los datos gestionados.

En definitiva, AutoProc representa un avance hacia la digitalización y modernización del trabajo de los procuradores, proporcionando una solución tecnológica innovadora, adaptable a nuevas exigencias del sector judicial y con potencial de expansión a nivel nacional.



# **Objetivos de Desarrollo Sostenible**

El proyecto realizado se relaciona con el ODS 8, específicamente con la meta 8.3, que promueve el desarrollo de actividades productivas y la innovación, apoyando a las pequeñas y medianas empresas.

La aplicación desarrollada optimiza la gestión de notificaciones y la organización de documentos, mejorando la eficiencia en el trabajo de los procuradores. Esta propuesta permite automatizar tareas repetitivas, liberando tiempo para actividades de mayor valor y fomentando un entorno laboral más productivo. Además, al ofrecer una plataforma integral con todas las herramientas e información que un procurador necesita, la solución optimiza su gestión profesional, proporcionando un entorno moderno y eficiente para su labor diaria. Esto fomenta la modernización de un sector basado en procesos manuales, adoptando nuevas tecnologías que mejoran la eficiencia y reducen el riesgo de errores. Al digitalizar y automatizar tareas, se promueve la innovación y se establece un enfoque más sostenible en la gestión profesional de los procuradores.



# Índice de contenidos

<b>Índice de contenidos</b>	<b>vii</b>
<b>Índice de figuras</b>	<b>xI</b>
<b>Índice de tablas</b>	<b>xIII</b>
<b>1 Introducción</b>	<b>1</b>
1.1. Contexto y justificación . . . . .	1
1.2. Público objetivo . . . . .	1
1.3. Visión del Proyecto . . . . .	2
<b>2 Planteamiento inicial</b>	<b>3</b>
2.1. Descripción del proyecto . . . . .	3
2.2. Objetivos principales y secundarios . . . . .	3
2.2.1. Objetivos principales del proyecto . . . . .	3
2.2.2. Objetivos secundarios del Proyecto . . . . .	4
2.3. Alcance del proyecto . . . . .	4
2.3.1. Ciclo de vida . . . . .	5
2.3.2. Estructura de Desglose de Trabajo (EDT) . . . . .	5
2.3.3. Descripción de exclusiones y supuestos . . . . .	11
2.3.4. Licencia del Proyecto . . . . .	11
2.4. Planificación temporal . . . . .	12
2.4.1. Estimación de dedicación a cada una de las tareas . . . . .	12
2.4.2. Hitos en el desarrollo del Proyecto . . . . .	12
2.4.3. Diagrama Gantt . . . . .	13
2.5. Gestión de riesgos . . . . .	15
2.5.1. Riesgos técnicos . . . . .	15
2.5.2. Riesgos de Seguridad . . . . .	16
2.5.3. Riesgos de Gestión del Proyecto . . . . .	16
2.6. Caracterización del sistema de información . . . . .	17
2.6.1. Estructura de los sistemas de información . . . . .	17
2.6.2. Formatos de los archivos . . . . .	18
2.6.3. Denominaciones . . . . .	18

2.6.4. Copias de seguridad . . . . .	18
2.7. Caracterización del sistema de comunicaciones . . . . .	19
2.7.1. Relaciones con el cliente . . . . .	19
2.7.2. Relaciones con los tutores . . . . .	19
<b>3 Captura de Requisitos</b>	<b>21</b>
3.1. Requisitos funcionales . . . . .	21
3.1.1. Gestión de usuarios . . . . .	21
3.1.2. Gestión de base de datos de abogados . . . . .	22
3.1.3. Gestión de base de datos de casos . . . . .	22
3.1.4. Gestión de notificaciones . . . . .	22
3.1.5. Envío de correos electrónicos . . . . .	22
3.1.6. Recursos y utilidades extra . . . . .	23
3.1.7. Integraciones . . . . .	23
3.2. Requisitos no funcionales . . . . .	23
3.2.1. Usabilidad . . . . .	23
3.2.2. Rendimiento . . . . .	23
3.2.3. Seguridad . . . . .	23
3.2.4. Mantenibilidad . . . . .	24
3.2.5. Escalabilidad . . . . .	24
3.2.6. Plataforma e infraestructura . . . . .	24
3.2.7. Estructura de organización local de notificaciones . . . . .	24
3.3. Diagrama de casos de uso . . . . .	24
3.4. Diagrama de dominio . . . . .	26
3.4.1. Entidades del dominio . . . . .	26
3.4.2. Relaciones del dominio . . . . .	29
<b>4 Herramientas y tecnologías</b>	<b>31</b>
4.1. Herramientas generales . . . . .	31
4.1.1. Editor de código fuente . . . . .	31
4.1.2. Control de versiones . . . . .	31
4.1.3. Diagramas y tablas . . . . .	32
4.2. Backend . . . . .	32
4.2.1. Lenguajes de programación . . . . .	32
4.2.2. Gestor de dependencias . . . . .	33
4.2.3. Frameworks . . . . .	33
4.2.4. Bibliotecas . . . . .	33
4.2.5. Bases de datos . . . . .	34
4.3. Frontend . . . . .	34
4.3.1. Lenguajes de programación . . . . .	34
4.3.2. Gestor de dependencias . . . . .	35
4.3.3. Frameworks y bibliotecas principales . . . . .	35
4.3.4. Herramientas de empaquetado . . . . .	36

4.3.5. Otras bibliotecas . . . . .	36
<b>5 Diseño de la solución</b>	<b>39</b>
5.1. Arquitectura del sistema . . . . .	39
5.1.1. Capa del modelo . . . . .	40
5.1.2. Capa de la vista . . . . .	40
5.1.3. Capa del controlador . . . . .	40
5.2. Diseño de las bases de datos . . . . .	40
5.2.1. Base de datos general . . . . .	40
5.2.2. Base de datos personal de cada usuario . . . . .	41
5.3. Interfaz de usuario . . . . .	42
5.3.1. Criterios de diseño . . . . .	42
5.3.2. Estructura general . . . . .	43
5.3.3. Componentes principales . . . . .	44
5.4. Lógica de negocio . . . . .	44
5.4.1. Servicios de seguridad . . . . .	44
5.4.2. Servicios CRUD para entidades . . . . .	45
5.4.3. Servicios de notificaciones . . . . .	45
<b>6 Desarrollo de la solución</b>	<b>47</b>
6.1. Configuración del entorno . . . . .	47
6.1.1. Instalación de herramientas y dependencias . . . . .	47
6.1.2. Configuración del entorno de desarrollo . . . . .	47
6.2. Desarrollo del backend . . . . .	48
6.2.1. Estructura del backend . . . . .	48
6.2.2. Configuración del servidor . . . . .	49
6.2.3. Enrutamiento y controladores . . . . .	50
6.2.4. Seguridad del servidor . . . . .	51
6.2.5. Conexión a la base de datos . . . . .	53
6.2.6. Implementación de la lógica de negocio . . . . .	54
6.2.7. Verificaciones y control de errores . . . . .	58
6.3. Desarrollo del frontend . . . . .	61
6.3.1. Estructura del frontend . . . . .	61
6.3.2. Enrutamiento . . . . .	63
6.3.3. Seguridad . . . . .	64
6.3.4. Vistas de la aplicación . . . . .	66
6.3.5. Validación y formateo de datos . . . . .	80
6.3.6. Manejo de errores y retroalimentación al usuario . . . . .	82
<b>7 Pruebas del sistema</b>	<b>85</b>
7.1. Pruebas del backend . . . . .	85
7.1.1. Pruebas mediante ThunderClient . . . . .	85
7.2. Pruebas del frontend . . . . .	90
7.2.1. Pruebas de validación de datos . . . . .	91

7.2.2. Pruebas de formateo de datos . . . . .	94
7.3. Pruebas de integración . . . . .	96
7.3.1. Pruebas de carga de las bases de datos . . . . .	97
7.3.2. Pruebas de envío de correos electrónicos . . . . .	97
7.4. Resumen de resultados . . . . .	97
<b>8 Seguimiento y control</b>	<b>99</b>
8.1. Reuniones y actas . . . . .	99
8.2. Retrasos significativos . . . . .	99
8.2.1. Causas del retraso . . . . .	99
8.2.2. Impacto del retraso . . . . .	100
8.2.3. Acciones adoptadas . . . . .	100
8.3. Cambios en la planificación . . . . .	100
8.3.1. Diagrama de Estructura de Desglose del Trabajo (EDT) extendido . . . . .	100
8.3.2. Diagrama de Gantt actualizado . . . . .	100
8.3.3. Diagrama de casos de uso extendido . . . . .	101
8.4. Seguimiento del tiempo invertido . . . . .	101
<b>9 Conclusión y trabajo a futuro</b>	<b>107</b>
9.1. Cumplimiento de los Objetivos . . . . .	107
9.2. Lecciones aprendidas . . . . .	107
9.3. Trabajo a futuro . . . . .	108
9.3.1. Despliegue y mantenimiento de la aplicación . . . . .	108
9.3.2. Expansión a nivel nacional . . . . .	108
9.3.3. Integración de nuevas funcionalidades . . . . .	108
9.3.4. Monitoreo de la aplicación . . . . .	108
9.4. Conclusiones . . . . .	108
<b>Apéndice</b>	<b>109</b>
<b>Actas de reuniones</b>	<b>111</b>
Acta UROA-TFG-AR-TUTORES-15/01/2024 . . . . .	111
Acta UROA-TFG-AR-CLIENTE-04/06/2024 . . . . .	111
Acta UROA-TFG-AR-CLIENTE-29/06/2024 . . . . .	112
Acta UROA-TFG-AR-CLIENTE-15/07/2024 . . . . .	112
Acta UROA-TFG-AR-CLIENTE-30/07/2024 . . . . .	112
Acta UROA-TFG-AR-CLIENTE-20/08/2024 . . . . .	113
Acta UROA-TFG-AR-CLIENTE-30/08/2024 . . . . .	113
Acta UROA-TFG-AR-TUTORES-04/09/2024 . . . . .	113
Acta UROA-TFG-AR-CLIENTE-20/09/2024 . . . . .	114
Acta UROA-TFG-AR-TUTORES-26/10/2024 . . . . .	114
<b>Bibliografía</b>	<b>115</b>

# Índice de figuras

2.1. Diagrama de Estructura de Desglose de Trabajo . . . . .	6
2.2. Diagrama de dependencia entre tareas . . . . .	10
2.3. Diagrama Gantt . . . . .	14
3.1. Diagrama de casos de uso . . . . .	25
3.2. Diagrama de dominio . . . . .	27
5.1. Modelo Vista Controlador . . . . .	39
5.2. Diseño de las bases de datos . . . . .	41
6.1. Estructura del backend . . . . .	49
6.2. Estructura del frontend . . . . .	62
6.3. Estructura del enrutamiento . . . . .	63
6.4. Estructura de las páginas . . . . .	66
6.5. Vista de HomePage.jsx . . . . .	67
6.6. Vista de RegisterPage.jsx . . . . .	68
6.7. Vista de LoginPage.jsx . . . . .	69
6.8. Instrucciones iniciales . . . . .	70
6.9. Enlaces de interés . . . . .	70
6.10. Mapa de los juzgados . . . . .	71
6.11. Configuración del perfil completada . . . . .	72
6.12. Actualización de los datos del perfil . . . . .	72
6.13. Diálogo de configuración de contraseña del correo . . . . .	73
6.14. Vista de carga de bases de datos . . . . .	74
6.15. Previsualización tras la carga de las bases de datos . . . . .	75
6.16. Visualización de las bases de datos . . . . .	76
6.17. Vista de notificaciones recibidas . . . . .	77
6.18. Envío automático de correos electrónicos . . . . .	78
6.19. Previsualización de un correo automático . . . . .	79
6.20. Envío manual de correos electrónicos . . . . .	79
6.21. Visualización de un correo manual . . . . .	80
6.22. Vista del calendario con un evento seleccionado . . . . .	81
6.23. Ejemplo de retroalimentación de éxito . . . . .	83
6.24. Ejemplo retroalimentación de advertencia con enlace . . . . .	83

6.25. Ejemplo de retroalimentación de error con enlace . . . . .	83
6.26. Retroalimentación del formulario de registro . . . . .	84
6.27. Diálogo con detalles de los errores . . . . .	84
7.1. Prueba de autorización mediante API Key - Error controlado . . . . .	86
7.2. Prueba de autorización mediante API Key - Éxito . . . . .	86
7.3. Prueba de autenticación mediante JWT - Error controlado . . . . .	87
7.4. Prueba de autenticación mediante JWT - Éxito . . . . .	87
7.5. Prueba de registro de usuario . . . . .	87
7.6. Bases de datos privadas creadas al registrar usuario . . . . .	88
7.7. Prueba de inicio de sesión - Error controlado . . . . .	88
7.8. Prueba de inicio de sesión - Éxito . . . . .	88
7.9. Prueba de creación de abogado - Éxito . . . . .	89
7.10. Prueba de lectura de abogado - Error controlado . . . . .	89
7.11. Prueba de lectura de abogado - Éxito . . . . .	90
7.12. Prueba de actualización de abogado - Error no previsto . . . . .	90
7.13. Prueba de actualización de abogado - Éxito . . . . .	90
7.14. Prueba de eliminación de abogado - Éxito . . . . .	91
7.15. Casos de prueba de ValidateEmailService.test.jsx . . . . .	91
7.16. Casos de prueba de ValidateNIGService.test.jsx . . . . .	92
7.17. Casos de prueba de ValidatePayService.test.jsx . . . . .	92
7.18. Casos de prueba de ValidatePhoneService.test.jsx . . . . .	92
7.19. Casos de pruebas de ValidateNameService.test.jsx . . . . .	93
7.20. Ejemplos de casos de prueba de ValidateDateService.test.jsx . . . . .	93
7.21. Casos de prueba de ValidateExpedientService.test.jsx . . . . .	93
7.22. Ejemplo de caso de prueba de ValidateHeadersService.test.jsx . . . . .	94
7.23. Ejemplo de caso de prueba de ValidateLawyerDataService.test.jsx . . . . .	94
7.24. Ejemplo de caso de prueba de ValidateCaseDataService.test.jsx . . . . .	95
7.25. Casos de prueba de FormatStringService.test.jsx . . . . .	95
7.26. Casos de prueba de FormatNameService.test.jsx . . . . .	96
7.27. Casos de prueba de FormatPhoneService.test.jsx . . . . .	96
7.28. Ejemplos de casos de prueba de FormatHeadersService.test.jsx . . . . .	96
7.29. Prueba de envío automático de 3 notificaciones . . . . .	97
7.30. Prueba de estructura de correo recibido . . . . .	98
7.31. Ejecución de las pruebas de validación y formateo de datos . . . . .	98
8.1. Diagrama de Estructura de Desglose de Trabajo extendido . . . . .	102
8.2. Diagrama Gantt actualizado . . . . .	103
8.3. Diagrama de casos de uso extendido . . . . .	104
8.4. Comparación entre dedicación prevista y real . . . . .	105

# Índice de tablas

2.1. Estimación de dedicación a cada una de las tareas . . . . .	13
1. UROA-TFG-AR-TUTORES-15/01/2024 . . . . .	111
2. UROA-TFG-AR-CLIENTE-04/06/2024 . . . . .	111
3. UROA-TFG-AR-CLIENTE-29/06/2024 . . . . .	112
4. UROA-TFG-AR-CLIENTE-15/07/2024 . . . . .	112
5. UROA-TFG-AR-CLIENTE-30/07/2024 . . . . .	112
6. UROA-TFG-AR-CLIENTE-20/08/2024 . . . . .	113
7. UROA-TFG-AR-CLIENTE-30/08/2024 . . . . .	113
8. UROA-TFG-AR-TUTORES-04/09/2024 . . . . .	113
9. UROA-TFG-AR-CLIENTE-20/09/2024 . . . . .	114
10. UROA-TFG-AR-TUTORES-26/10/2024 . . . . .	114



# Índice de fragmentos de código

6.1.	Configuración del servidor . . . . .	50
6.2.	Registro de rutas principales . . . . .	51
6.3.	Ejemplo de rutas de la API de abogados . . . . .	51
6.4.	Asignación de token JWT al usuario . . . . .	52
6.5.	Cifrado y descifrado mediante AES . . . . .	52
6.6.	Cifrado y verificación mediante hash . . . . .	53
6.7.	Ejemplo de conexión y consulta de usuario en MongoDB . . . . .	54
6.8.	Ejemplo de extracción del _id del usuario y asignación de la BD . . . . .	54
6.9.	Servicio para crear un abogado . . . . .	55
6.10.	Servicio para actualizar un caso . . . . .	55
6.11.	Ejemplo completo de verificación de datos y control de errores . . . . .	60
6.12.	Definición de rutas en AppRoutes.jsx . . . . .	64
6.13.	Configuración de rutas privadas en PrivateRoute.jsx . . . . .	64
6.14.	Ejemplo simplificado de solicitud de login con cabecera API Key . . . . .	65
6.15.	Verificación y validez del token JWT en AuthProvider.jsx . . . . .	66



## CAPÍTULO

# 1

# Introducción

## 1.1. Contexto y justificación

En el ámbito judicial, los procuradores representan legalmente a particulares, empresas y entidades ante los tribunales, actuando como intermediarios entre los clientes y el sistema judicial. Su labor implica la gestión y presentación de documentos judiciales, la tramitación de notificaciones, la coordinación de pagos, así como el seguimiento de los expedientes judiciales. Sin embargo, este proceso se realiza manualmente en gran medida, siendo propenso a errores y consumiendo mucho tiempo y recursos.

La digitalización y automatización de estas gestiones mediante la tecnología, puede marcar un cambio significativo en la manera en la que los procuradores llevan a cabo sus responsabilidades diarias. De hecho, actualmente en el País Vasco, los procuradores utilizan la plataforma *Avantius*, siendo el sistema oficial para la consulta de las notificaciones que deben gestionar. Sin embargo, esta herramienta simplemente actúa como intermediaria entre el sistema judicial y los procuradores, dándoles acceso a las notificaciones que han de gestionar. Por esa misma razón es por la que surge una oportunidad de mejora.

Partiendo de esta necesidad, el presente proyecto tiene como objetivo desarrollar una aplicación web integral que permita a los procuradores gestionar todos los aspectos relacionados con sus casos de manera eficiente. Esta plataforma no solo optimizará los procesos diarios y reducirá posibles errores, sino que centralizará toda la información relevante en un entorno accesible y seguro. Al integrar funcionalidades como la gestión de expedientes, notificaciones y pagos, se busca proporcionar una herramienta robusta.

## 1.2. Público objetivo

El público objetivo principal de este sistema son los procuradores en el País Vasco. La solución propuesta les proporcionará una herramienta para facilitar la gestión de documentos,

## **1. INTRODUCCIÓN**

---

mejorar la productividad y reducir posibles errores.

### **1.3. Visión del Proyecto**

La visión detrás del proyecto es ambiciosa y busca transformar por completo la experiencia laboral de los procuradores. Se aspira a modernizar la gestión de sus tareas diarias mediante una plataforma web integral. Esta herramienta centralizará y optimizará procesos clave como la gestión de expedientes o de notificaciones.

Al implementar tecnologías avanzadas y eficientes, el proyecto pretende automatizar tareas repetitivas y propensas a errores, facilitando un trabajo más eficiente y preciso. El objetivo es proporcionar una herramienta segura y accesible que permita a los procuradores agilizar procesos monótonos para poder enfocarse en actividades de mayor valor añadido.

# CAPÍTULO 2

## Planteamiento inicial

En este capítulo se presenta el sistema de planificación del proyecto planteado, incluyendo las metas a alcanzar, la estimación de dedicaciones y otros aspectos relacionados.

### 2.1. Descripción del proyecto

El proyecto consiste en la creación de una plataforma web donde los procuradores pueden gestionar sus casos, organizar expedientes, visualizar notificaciones, gestionar pagos, entre otros. La aplicación utilizará tecnologías modernas y buenas prácticas de desarrollo para asegurar su funcionalidad y facilidad de uso.

### 2.2. Objetivos principales y secundarios

Los objetivos del proyecto son fundamentales para guiar su desarrollo y asegurar que la solución propuesta cubra tanto las necesidades principales como las funcionalidades adicionales que mejoren la experiencia del usuario.

#### 2.2.1. Objetivos principales del proyecto

Los objetivos principales del proyecto forman la base funcional de la aplicación. A través de ellos, se garantiza que la solución sea eficaz, proporcionando al usuario las herramientas necesarias para una gestión completa de sus expedientes.

- Gestión de usuarios:** Los usuarios podrán registrarse y autenticarse de forma segura, garantizando la privacidad de sus datos y facilitando el acceso a las funcionalidades de la aplicación.
- Configuración de perfil:** Antes de proceder con el envío de correos electrónicos o con la gestión de notificaciones, los usuarios deberán completar toda la información de su perfil. A su vez, podrán visualizar y modificar estos datos en cualquier momento, asegurando que siempre estén actualizados y sean correctos.

## 2. PLANTEAMIENTO INICIAL

---

3. **Carga y gestión de bases de datos:** Los usuarios de la plataforma podrán cargar, visualizar y modificar las dos bases de datos con las que trabajan habitualmente. Una está relacionada con los casos judiciales a cargo del usuario, y la otra con los abogados vinculados a esos casos, permitiendo una organización eficiente, y garantizando que la información sea accesible y modificable.
4. **Organización de notificaciones:** El sistema permitirá al usuario gestionar sus notificaciones mediante la creación de una estructura de carpetas en un directorio local, establecido por el propio usuario. Aunque el usuario cargará inicialmente las notificaciones recibidas, el sistema organizará automáticamente los archivos tras el envío de los correos electrónicos.
5. **Extracción de datos de notificaciones:** El sistema contará con una funcionalidad interna que permitirá la extracción automática de los datos clave de las notificaciones (documentos PDF) mediante OCR (Reconocimiento Óptico de Caracteres). Estos datos se incluirán en los correos electrónicos enviados posteriormente.
6. **Envío automático de correos electrónicos:** Existirá una funcionalidad que permitirá el envío automático de correos electrónicos. Antes del envío, se ofrecerá una previsualización de los correos, los cuales incluirán los datos extraídos de las notificaciones, integrados en una plantilla predefinida.

### 2.2.2. Objetivos secundarios del Proyecto

Los objetivos secundarios del proyecto complementan las funcionalidades principales de la aplicación, mejorando la experiencia del usuario a través de la integración de herramientas externas.

1. **Información adicional y recursos:** Se integrará un apartado con acceso a recursos, enlaces de interés y guías detalladas que facilitarán el uso y la comprensión de la plataforma.
2. **Envío manual de correos electrónicos:** Además del envío automático, el sistema ofrecerá la opción de envío manual de correos electrónicos. En este caso, el correo se generará del mismo modo que en el envío automático, pero el usuario podrá editar el contenido antes de proceder con el envío.
3. **Integración con Google Maps:** Se integrará Google Maps en la plataforma, permitiendo al usuario visualizar la ubicación de los juzgados, facilitando así la planificación de los desplazamientos.

### 2.3. Alcance del proyecto

El alcance del proyecto abarca el desarrollo de una plataforma web integral para optimizar y facilitar el trabajo diario de los procuradores. El sistema incluye la gestión de las bases de

## 2.3. Alcance del proyecto

---

datos de cada procurador, así como de la gestión de las notificaciones y del envío de correos electrónicos de cada uno. De esta manera, se permite automatizar tareas repetitivas y reducir posibles errores a causa de la gestión manual de múltiples documentos diarios.

Además, la aplicación empleará herramientas avanzadas para la extracción automática de datos de los documentos mediante OCR, el envío automático de correos electrónicos y la planificación automática de los juicios mediante un calendario.

### 2.3.1. Ciclo de vida

Se ha optado por un ciclo de vida en cascada con una ligera variación, en el que la fase de desarrollo se organiza en iteraciones. Cada una corresponde con una parte funcional de la aplicación y, al final de cada iteración, se realizarán pruebas que permitirán hacer ajustes si fuera necesario. Este enfoque permite desarrollar la aplicación de manera flexible y eficiente, adaptándose a los posibles cambios y garantizando una entrega continua de valor al cliente. La retroalimentación activa del procurador a lo largo del proceso asegura un producto final de calidad. Además, este sistema favorece la mitigación de riesgos mediante la identificación y resolución temprana de posibles problemas.

### 2.3.2. Estructura de Desglose de Trabajo (EDT)

La Estructura de Desglose de Trabajo se refiere a la organización del trabajo fragmentado en componentes manejables. Esta descomposición proporciona una visión clara de las tareas necesarias para alcanzar los objetivos del proyecto. Al estructurar el trabajo de esta manera, se asegura tener en cuenta todos los aspectos necesarios, facilitando una ejecución ordenada y efectiva de las tareas.

#### 2.3.2.1. Diagrama EDT

En la figura 2.1 se muestra la representación visual de la Estructura de Desglose de Trabajo.

#### 2.3.2.2. Descripción de los paquetes y tareas

- **Paquete de Planificación y Gestión.** Este paquete se enfoca en establecer una estrategia estructurada para el desarrollo del proyecto. Busca definir y organizar tanto los objetivos como las tareas a realizar, asegurando una comunicación constante con las partes interesadas. El adecuado seguimiento y control del progreso garantizará un resultado final de calidad.
  - **Definición de objetivos.** Se establecen los objetivos clave del proyecto.
  - **Planificación del proyecto.** Se organiza el trabajo a través de un diagrama Gantt que define las tareas y los plazos.
  - **Comunicación con stakeholders.** Se asegura una comunicación efectiva y continua con los interesados para alcanzar las expectativas.
  - **Seguimiento y control del proyecto.** Se supervisa el avance y se implementan ajustes para cumplir con los objetivos establecidos.

## 2. PLANTEAMIENTO INICIAL

---

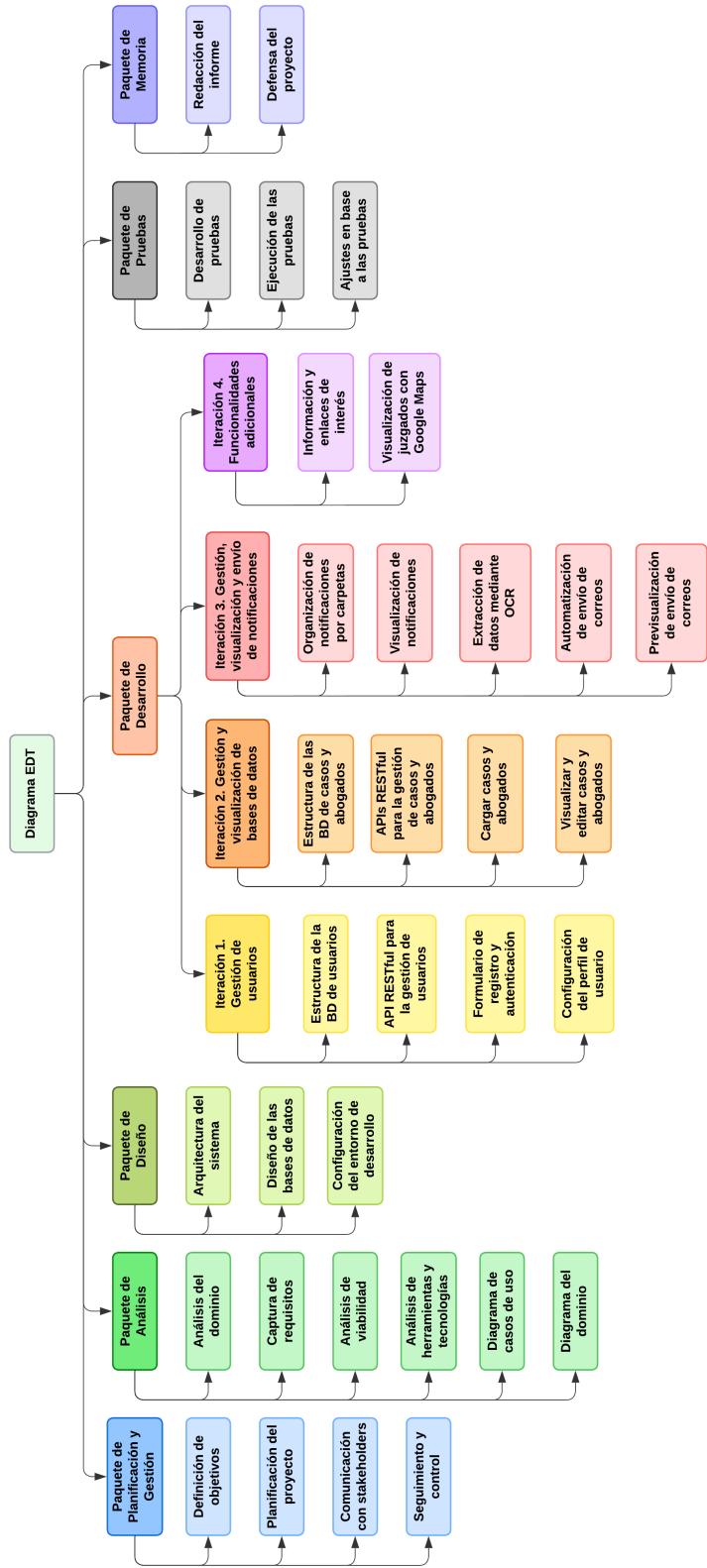


Figura 2.1: Diagrama de Estructura de Desglose de Trabajo

- **Paquete de trabajo Análisis.** El paquete de trabajo de análisis se enfoca en comprender el dominio del proyecto y las necesidades del cliente. Abarca la captura de requisitos, la viabilidad del proyecto y la evaluación de herramientas y tecnologías adecuadas. Los resultados de este apartado se reflejarán en diagramas que facilitarán la visualización de los componentes y relaciones dentro del sistema.
  - **Análisis del dominio.** Se estudia el entorno del proyecto y los conceptos clave para comprender las necesidades del cliente.
  - **Captura de requisitos.** Se recogen y documentan las funcionalidades y características necesarias para el sistema.
  - **Análisis de viabilidad.** Se evalúa la viabilidad del proyecto para asegurar su desarrollo exitoso.
  - **Análisis de herramientas y tecnologías.** Se identifican y seleccionan las tecnologías más adecuadas para implementar el sistema.
  - **Diagramas de casos de uso.** Se crea un diagrama para mostrar las interacciones entre los usuarios y el sistema.
  - **Diagrama del dominio.** Se representa gráficamente la estructura principal del sistema y las relaciones entre sus componentes.
- **Paquete de trabajo Diseño.** El diseño es clave para establecer una estructura organizada y eficiente en el desarrollo de la aplicación. Esto incluye el diseño de la arquitectura y de las bases de datos, así como la configuración del entorno donde se llevará a cabo el desarrollo.
  - **Arquitectura del sistema.** Se define la estructura y los componentes principales del sistema.
  - **Diseño de las bases de datos.** Se diseña la estructura de las bases de datos para almacenar y gestionar la información de manera eficiente.
  - **Configuración del entorno de desarrollo.** Se establecen las herramientas y configuraciones necesarias para facilitar el desarrollo del proyecto.
- **Paquete de trabajo Desarrollo.** Este paquete de trabajo se centra en la implementación de las funcionalidades definidas en fases anteriores. Está estructurado en varias iteraciones, donde cada una completa distintas partes de la aplicación. De esta modo, la solución será funcional desde la primera iteración, y a medida que avance el desarrollo, se incorporarán las demás funcionalidades.
  - **Iteración 1 - Gestión de usuarios.** En esta fase inicial, el objetivo es crear una funcionalidad básica pero fundamental, la gestión de usuarios. Esto implica implementar los mecanismos necesarios para que los usuarios puedan registrarse, autenticarse y gestionar su información dentro de la plataforma.

- Estructura de la BD de usuarios

## 2. PLANTEAMIENTO INICIAL

---

- API RESTful para la gestión de usuarios
- Formulario de registro y autenticación
- Configuración del perfil de usuario

**Iteración 2 - Gestión y visualización de bases de datos.** Durante esta iteración, se incorpora la gestión personalizada de los expedientes y casos, lo que permite a cada usuario contar con sus propias bases de datos. Esta estructura facilita la organización y visualización de la información, garantizando un acceso eficiente a las bases de datos de cada usuario.

- Estructura de las BD de casos y abogados (únicas para cada usuario)
- API RESTful para la gestión de casos y abogados
- Cargar casos y abogados
- Visualizar y editar casos y abogados

**Iteración 3 - Gestión, visualización y envío de notificaciones.** Esta fase introduce una funcionalidad avanzada enfocada a la automatización de procesos. Además de la gestión y visualización de las notificaciones, se implementa un sistema que extrae automáticamente los datos necesarios de notificaciones y envía los correos electrónicos. Antes de que se efectúe el envío, el usuario podrá previsualizar los mensajes. De esta manera se garantiza un control claro sobre la comunicación, mejorando a su vez la eficiencia del proceso.

- Organización de notificaciones por carpetas
- Visualización de notificaciones
- Extracción de datos mediante OCR
- Automatización de envío de correos
- Previsualización de envío de correos

**Iteración 4 - Integraciones y funcionalidades adicionales.** La última iteración incorpora funcionalidades complementarias que mejoran la experiencia del usuario. Se integran herramientas como Google Maps, que permite la visualización de los juzgados en un mapa, y un calendario que gestiona automáticamente las fechas de los juicios, facilitando así su seguimiento.

- Visualización de juzgados con Google Maps
- Calendario de fechas de juicios

■ **Paquete de trabajo Pruebas.** Las pruebas son esenciales para garantizar la calidad del producto. Este paquete abarca tanto el desarrollo como la ejecución de éstas, permitiendo identificar y corregir errores.

- **Desarrollo de pruebas.** Se definen y crean los casos de prueba necesarios para verificar las funcionalidades del sistema.

## 2.3. Alcance del proyecto

---

- **Ejecución de pruebas.** Se llevan a cabo las pruebas planificadas para evaluar el funcionamiento del sistema.
  - **Ajustes en base a las pruebas.** Se realizan correcciones y mejoras según los resultados obtenidos en las pruebas ejecutadas.
- **Paquete de trabajo Memoria.** El paquete de trabajo de la memoria recoge la documentación completa del proceso del desarrollo del proyecto. Incluye el informe que detalla todos los aspectos del proyecto y la defensa del mismo, asegurando la representación clara del trabajo realizado y de los resultados obtenidos.
    - **Redacción del Informe.** Se elabora la documentación que recoge todos los aspectos del proceso del proyecto.
    - **Defensa del Proyecto.** Preparación y presentación del proyecto ante el tribunal.

### 2.3.2.3. Dependencia entre tareas

Las dependencias entre tareas son fundamentales para el desarrollo efectivo del proyecto, ya que determinan el orden en que se deben realizar las actividades. La organización de estas relaciones permite una planificación adecuada y asegura que cada fase del proyecto se complete de manera ordenada.

En la siguiente figura 2.2, se emplean dos tipos de relaciones. La flecha continua indica que el inicio de la siguiente tarea o paquete depende de la finalización de la tarea actual. Por otro lado, la flecha discontinua significa que ambas tareas o paquetes pueden comenzar simultáneamente, pero que la finalización de la siguiente dependerá de que la actual se complete.

#### ▪ Dependencias entre los paquetes de trabajo principales.

- **Paquete de Planificación y Gestión y paquete de Análisis.** La **definición de objetivos** se basa en el **análisis del dominio**, ya que éste proporciona una comprensión clara del entorno y de las necesidades, lo cual resulta esencial para establecer objetivos coherentes.

A su vez, la **captura de requisitos** dependerá de estos objetivos, asegurando que las necesidades recogidas se adecúen al proyecto. Las siguientes **tareas del análisis** se desarrollaran de forma secuencial, dependiendo cada tarea de la finalización de la anterior.

Finalmente, la **planificación del proyecto** comenzará una vez completado el **diagrama del dominio**. Asimismo, tanto la **comunicación con los stakeholders** como el **seguimiento y control** del proyecto se realizarán de forma continua a lo largo de su desarrollo.

- **Paquete de trabajo Diseño.** Las tareas relacionadas con el diseño podrán comenzar durante el desarrollo de la planificación del **paquete de trabajo Análisis**.

## 2. PLANTEAMIENTO INICIAL

---

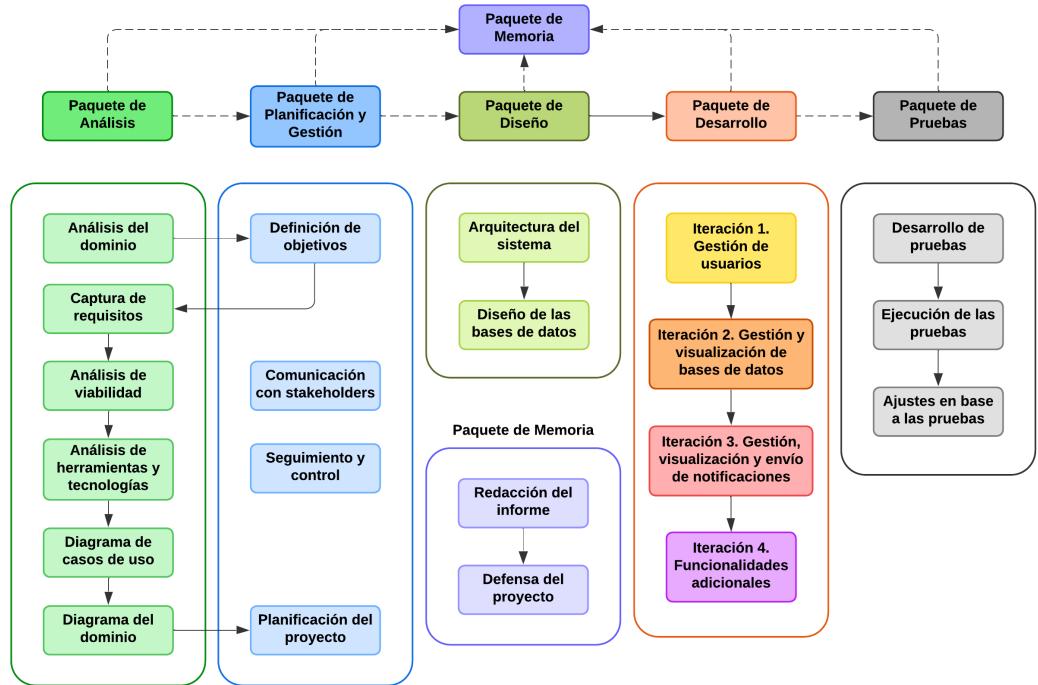


Figura 2.2: Diagrama de dependencia entre tareas

- **Paquete de trabajo Desarrollo.** La implementación de las funcionalidades comenzará tras la finalización del **paquete de trabajo Diseño**.
  - **Paquete de trabajo Pruebas.** Las tareas relacionadas con las pruebas comenzarán al mismo tiempo que el **paquete de trabajo Desarrollo** y se llevarán a cabo en cada iteración. Al final de cada una, se ejecutarán las pruebas correspondientes, lo que permitirá realizar ajustes si fuera necesario. La finalización de este paquete dependerá de que se completen todas las iteraciones.
  - **Paquete de trabajo Memoria.** La redacción de la memoria del proyecto se realizará de forma paralela a las demás fases. Sin embargo, su finalización requerirá que se completen los demás paquetes de trabajo.
- **Dependencias en el paquete de trabajo Desarrollo.**
- Las tareas de la **iteración 1** comenzarán tras la finalización del **paquete de diseño**.
  - Las tareas de la **iteración 2** dependerán de la **iteración 1**.
  - Las tareas de la **iteración 3** requerirán la finalización de la **iteración 2**.
  - Las tareas de la **iteración 4** se llevarán a cabo una vez finalizada la **iteración 3**.

■ **Dependencias en el paquete de trabajo Pruebas.**

- El **desarrollo de las pruebas** podrá comenzar de forma paralela a cada iteración del **paquete de trabajo Desarrollo**.
- La **ejecución de las pruebas** se realizará una vez completado su desarrollo.
- Los **ajustes en base a las pruebas** dependerán de los resultados obtenidos tras su desarrollo.

■ **Dependencias en el paquete de trabajo Memoria.**

- La **redacción del informe** es una tarea que abarca todas las fases del proyecto. Su finalización dependerá de que se completen los demás paquetes de trabajo.
- La **defensa del proyecto** tendrá lugar una vez finalizado y entregado el informe.

### 2.3.3. Descripción de exclusiones y supuestos

Las exclusiones y supuestos de este proyecto determinan las condiciones bajo las cuales la plataforma funcione correctamente y su desarrollo sea óptimo.

#### 2.3.3.1. Exclusiones del proyecto

- La aplicación está diseñada exclusivamente para los procuradores del País Vasco, por lo que no se ha considerado la forma de trabajo o plataformas empleadas por procuradores de otras regiones.

#### 2.3.3.2. Supuestos del proyecto

- Se asume que los usuarios se encargarán de añadir manualmente los documentos de las notificaciones descargadas de su plataforma *Avantius*<sup>1</sup>. Estos documentos deberán ser almacenados en la carpeta de notificaciones recibidas, que se generará automáticamente dentro del directorio local establecido por el propio usuario.
- Se asume que los usuarios harán un uso responsable de la plataforma, lo que incluye verificar los correos electrónicos generados automáticamente mediante las previsualizaciones disponibles, garantizando que el contenido sea correcto.
- Se asume que los usuarios mantendrán su información personal actualizada para la correcta generación automática de los correos electrónicos.

#### 2.3.4. Licencia del Proyecto

Este proyecto se acoge a la Licencia Creative Commons Atribución-NoComercial-SinDerivadas 4.0 Internacional (CC BY-NC-ND 4.0). Esto implica que se permite compartir el proyecto siempre que se atribuya adecuadamente al autor, no se use con fines comerciales y no se realicen modificaciones ni obras derivadas [1].

---

<sup>1</sup><https://egoitza.justizia.eus/psp-euskadi-es/webjus01-contentgen/es/>

## 2.4. Planificación temporal

La planificación temporal del proyecto es fundamental para asegurar un desarrollo eficiente y organizado. Esta sección ofrece una visión detallada de los hitos y tareas clave, así como una representación visual que incluye las duraciones correspondientes.

### 2.4.1. Estimación de dedicación a cada una de las tareas

Estimar el tiempo necesario para completar cada tarea es crucial para lograr una organización temporal clara y cumplir con los plazos establecidos. En la tabla 2.1 se muestra la organización de las tareas definidas por paquetes de trabajo, junto con la estimación de dedicación para cada una de ellas.

### 2.4.2. Hitos en el desarrollo del Proyecto

Los hitos representan puntos clave durante el desarrollo del proyecto, indicando la finalización de etapas importantes. Para este proyecto, los principales hitos serán la finalización de cada uno de los paquetes de trabajo, además de la entrega del TFG y su defensa.

- **Finalización del paquete de trabajo Planificación y Gestión.** No solo marca la conclusión de la planificación del proyecto, sino que también refleja la finalización de la gestión a lo largo de su desarrollo. Fecha de finalización: **07/09/2024**.
- **Finalización del paquete de trabajo Análisis.** Indica que se ha completado el análisis completo del proyecto, incluyendo el análisis del dominio, la captura de requisitos y la viabilidad entre otros. Fecha de finalización: **17/06/2024**.
- **Finalización del paquete de trabajo Diseño.** Significa que la arquitectura del sistema y el diseño de las bases de datos han sido definidos, dando lugar al comienzo del desarrollo del proyecto. Fecha de finalización: **02/07/2024**.
- **Finalización del paquete de trabajo Desarrollo.** Refleja la finalización de todas las iteraciones propuestas, donde se han implementado tanto las funcionalidades principales como las secundarias. Fecha de finalización: **20/08/2024**.
- **Finalización del paquete de trabajo Pruebas.** Indica que las pruebas necesarias han sido desarrolladas y ejecutadas. Si fuese necesario, también se habrán realizado los ajustes pertinentes en la aplicación. Fecha de finalización: **29/08/2024**.
- **Entrega del TFG.** Representa la entrega final del proyecto completo. Fecha de entrega: **14/09/2024**.
- **Defensa del TFG.** Indica el momento en el que se presenta el proyecto ante un tribunal. Fecha de defensa: entre el **23/09/2024** y el **27/09/2024**.

## 2.4. Planificación temporal

Paquete de Trabajo / Iteraciones	Tareas	Estimación (h)	Total por paquete / iteración (h)
Paquete de trabajo Planificación y Gestión	Definición de objetivos	4	44
	Planificación del proyecto	20	
	Comunicación con stakeholders	10	
	Seguimiento y control	10	
Paquete de trabajo Análisis	Análisis del dominio	4	34
	Captura de requisitos	4	
	Análisis de viabilidad	2	
	Análisis de herramientas y tecnologías	8	
	Diagrama de casos de uso	8	
	Diagrama del dominio	8	
Paquete de trabajo Diseño	Arquitectura del sistema	10	23
	Diseño de las bases de datos	8	
	Configuración del entorno de desarrollo	5	
Paquete de trabajo Desarrollo	Estructura de la BD de usuarios	4	30
	API RESTful para la gestión de usuarios	12	
	Formulario de registro y autenticación	8	
	Configuración del perfil de usuario	6	
	Estructura de la BD de casos y abogados	6	34
	API RESTful para la gestión de casos y abogados	12	
	Cargar casos y abogados	8	
	Visualizar y editar casos y abogados	8	
	Organización de notificaciones por carpetas	8	40
	Visualización de notificaciones	6	
	Extracción de datos mediante OCR	6	
	Automatización de envío de correos	12	
IT4. Integraciones y funcionalidades	Previsualización de envío de correos	8	14
	Información y enlaces de interés	6	
	Visualización de juzgados con Google Maps	8	
	Desarrollo de pruebas	10	
Paquete de trabajo Pruebas	Ejecución de pruebas	4	24
	Ajustes en base a las pruebas	10	
	Redacción del Informe	60	66
Paquete de trabajo Memoria	Defensa del Proyecto	6	
<b>Horas totales</b>		<b>309</b>	

Tabla 2.1: Estimación de dedicación a cada una de las tareas

### 2.4.3. Diagrama Gantt

El diagrama de Gantt, el cual se muestra en la figura 2.3, ofrece una representación visual de la planificación temporal del proyecto. Muestra de manera clara el orden y la duración de las tareas, así como las fechas de los hitos más importantes. Además, facilita el seguimiento y control del proyecto, permitiendo identificar rápidamente cualquier desviación respecto a la planificación.

## 2. PLANTEAMIENTO INICIAL

---

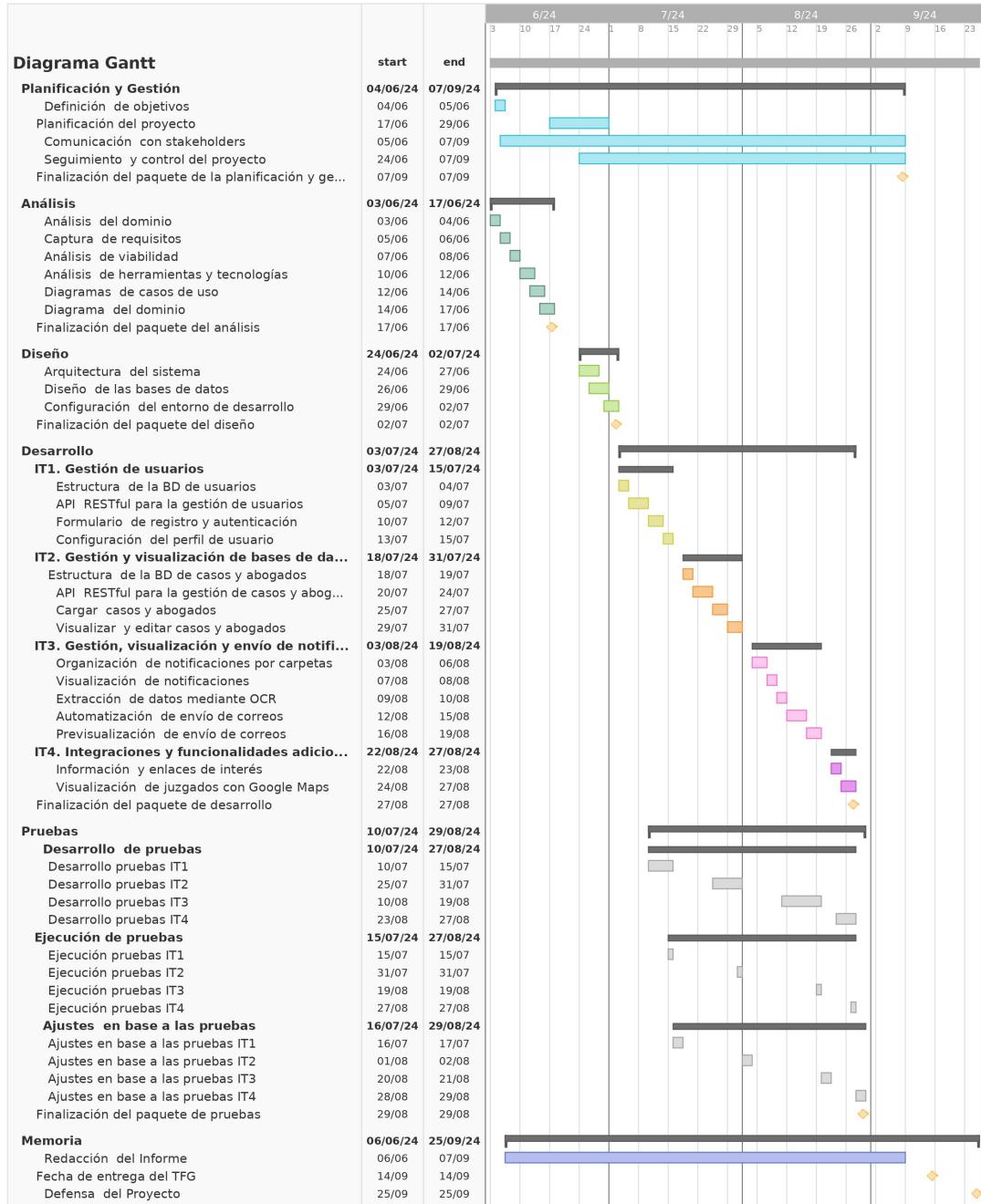


Figura 2.3: Diagrama Gantt

## 2.5. Gestión de riesgos

La gestión de riesgos es un aspecto clave para el éxito del proyecto propuesto. Identificar los posibles riesgos con antelación permite implementar medidas preventivas que minimicen su impacto, así como acciones de mitigación en caso de que ocurran. A continuación, se detallan los riesgos identificados en el proyecto, organizados en riesgos técnicos, de seguridad y de gestión, junto con sus respectivas estrategias de prevención y mitigación.

### 2.5.1. Riesgos técnicos

Los riesgos técnicos contemplan los posibles problemas relacionados con la compatibilidad de tecnologías y herramientas utilizadas en el proyecto, así como la integración de APIs y otros componentes clave.

- **R1 - Compatibilidad de formatos de las bases de datos**

**Descripción:** Problemas de compatibilidad al cargar las bases de datos del usuario.

**Impacto:** Alto. Puede causar pérdida de datos o errores en la migración.

**Probabilidad:** Media.

**Prevención:** Establecer formatos claros para las bases de datos y desarrollar *scripts* de validación que verifiquen la compatibilidad de los datos antes de la migración.

**Mitigación:** Advertir al usuario de cualquier incompatibilidad de datos y evitar la actualización hasta que se ajusten al formato especificado.

- **R2 - Compatibilidad de versiones**

**Descripción:** Problemas de compatibilidad entre versiones de las tecnologías y herramientas empleadas en el proyecto.

**Impacto:** Medio. Puede causar errores en la aplicación y dificultades en la integración de componentes.

**Probabilidad:** Media.

**Prevención:** Realizar un análisis exhaustivo de las versiones de todas las tecnologías y herramientas antes de comenzar el desarrollo, asegurando su compatibilidad.

**Mitigación:** Disponer de herramientas y tecnologías alternativas para cubrir posibles incompatibilidades durante el proyecto e implementar pruebas de integración continuas.

## 2. PLANTEAMIENTO INICIAL

---

### ■ R3 - Limitaciones en la integración de APIs externas

**Descripción:** Limitaciones en la integración de las APIs de Google Maps y las relacionadas con el correo electrónico.

**Impacto:** Alto. Afecta a varias funcionalidades del proyecto.

**Probabilidad:** Media.

**Prevención:** Realizar pruebas tempranas de integración de las APIs para identificar y resolver posibles problemas con antelación.

**Mitigación:** Diseñar una solución alternativa en caso de que existan restricciones.

### 2.5.2. Riesgos de Seguridad

Los riesgos de seguridad se centran en las posibles vulnerabilidades que puedan comprometer la confidencialidad y la integridad de los datos gestionados por la aplicación. Es crucial implementar medidas que minimicen la posibilidad de fugas o accesos no autorizados.

### ■ R4 - Fuga de datos y brechas de seguridad

**Descripción:** Acceso no autorizado a datos sensibles.

**Impacto:** Muy Alto. Puede comprometer la confidencialidad y la integridad de la información de los usuarios.

**Probabilidad:** Baja.

**Prevención:** Implementar políticas de contraseñas y de autenticación fuertes, además del cifrado de los datos sensibles.

**Mitigación:** Monitoreo continuo y respuesta rápida ante incidencias de seguridad.

### 2.5.3. Riesgos de Gestión del Proyecto

Los riesgos relacionados con la gestión del proyecto incluyen modificaciones en los requisitos del usuario y posibles desviaciones en los plazos de entrega.

### ■ R5 - Cambios en los requisitos del usuario

**Descripción:** Modificaciones significativas en los requisitos durante el desarrollo.

**Impacto:** Alto. Puede llevar a retrasos y sobrecostes de tiempo.

**Probabilidad:** Media.

**Prevención:** Adoptar un enfoque ágil con iteraciones cortas y retroalimentación continua. Mantener una comunicación clara y constante con los stakeholders.

## 2.6. Caracterización del sistema de información

---

**Mitigación:** Reevaluar y ajustar el plan de trabajo y los recursos en función de los nuevos requisitos.

### ■ R6 - Retrasos respecto a la planificación

**Descripción:** Desviaciones significativas respecto al plan de trabajo.

**Impacto:** Alto. Afecta a la entrega y la implementación del proyecto.

**Probabilidad:** Media.

**Prevención:** Desglosar el proyecto en tareas manejables con hitos claros. Incorporar un margen de tiempo adicional en la planificación para abordar posibles imprevistos.

**Mitigación:** Redefinir el alcance del proyecto, reasignar recursos para tareas críticas y ajustar el calendario de entregas.

## 2.6. Caracterización del sistema de información

Este apartado establece una estructura de organización adecuada de los archivos y recursos desarrollados en el proyecto.

### 2.6.1. Estructura de los sistemas de información

La estructura general del proyecto se organizará de manera clara y lógica, lo que facilita su comprensión y desarrollo. A continuación, se describen las principales carpetas y su contenido.

- **Código fuente.** La carpeta principal del código fuente se dividirá en dos secciones: frontend y backend. Esta separación permite gestionar de manera eficiente las distintas funcionalidades de la aplicación.
  - **Backend.** En este apartado se encontrarán los archivos del servidor, desarrollados en Python. El código estará estructurado en subcarpetas, facilitando la navegación y comprensión del sistema.
  - **Frontend.** Esta carpeta contendrá los archivos relacionados con la interfaz de usuario, desarrollados en React. Se estructura en componentes pequeños y reutilizables, lo que mejora la mantenibilidad y el entendimiento del código.

Además, estas carpetas incluirán archivos de configuración del entorno y pruebas de la aplicación, que son esenciales para el correcto funcionamiento de la aplicación.

- **Memoria.** Esta carpeta recogerá los recursos utilizados en la elaboración de la memoria del proyecto, incluyendo esquemas, diagramas y cualquier elemento que complemente la documentación escrita.

### 2.6.2. Formatos de los archivos

A lo largo del proyecto se desarrollarán diferentes tipos de archivos, cada uno con el formato adecuado según su tipo. A continuación, se detallan cada uno de los formatos utilizados.

- **Código fuente.** El código fuente se almacena principalmente en archivos “.py” para las funcionalidades del backend, desarrolladas en Python, y en archivos “.jsx” para aquellas del frontend, realizadas con React.
- **Bases de datos.** La estructura y el almacenamiento de datos también exigen un formato específico. Los datos del proyecto se manejarán en formato “.json”, que resulta ideal para la estructura NoSQL de MongoDB, siendo muy flexible.
- **Documentación.** La documentación del proyecto se presentará en un formato que asegure una presentación clara y profesional, utilizando la extensión “.tex” en Overleaf. Cabe destacar que su entrega final será exportada en formato “.pdf”.
- **Recursos adicionales.** La memoria del proyecto se complementará con recursos adicionales que facilitarán la representación visual de diversos esquemas. Los diagramas se almacenarán en formato “.vdx”, pero se incluirán en la documentación como imágenes con formatos “.png” y “.jpg”. Además, se utilizarán tablas construidas en archivos Excel, que se guardarán con la extensión “.xlsx”.

### 2.6.3. Denominaciones

Para mantener la constancia en la organización de las actas de reunión, cada documento será denominado mediante un formato claro.

- **Reuniones.** Las actas de reuniones se archivarán en tablas mediante la siguiente denominación: “UROA-TFG-AR-TUTORES/CLIENTE-dd/mm/aaaa”. Las siglas “AR” corresponden con “acta de reunión”, y se especificará si el encuentro fue con tutores o con el cliente en los siguientes caracteres. Este formato finalizará con la fecha de la reunión para su rápida identificación.

### 2.6.4. Copias de seguridad

Las copias de seguridad son esenciales para prevenir la pérdida de datos y asegurar la continuidad del proyecto. Éstas se gestionarán a través de diferentes plataformas en función del tipo de archivo.

- **Código fuente.** GitHub será la herramienta principal para almacenar todo el código fuente de la aplicación. Además de ofrecer una copia de seguridad, GitHub permite mantener un control de versiones de la aplicación, lo que facilita el seguimiento de cambios y posibilita de revertir los cambios realizados en caso de errores.

## 2.7. Caracterización del sistema de comunicaciones

---

- **Bases de datos.** Las bases de datos se almacenan en la nube mediante MongoDB Atlas. Esta plataforma proporciona copias de seguridad automáticas, garantizando la protección de los datos y facilitando la recuperación en caso de pérdida.
- **Documentación.** La redacción del informe se realizará en la plataforma OverLeaf, que permite el almacenamiento en la nube de los documentos y su edición en tiempo real.
- **Recursos adicionales.** Los archivos relacionados con la memoria del proyecto, como diagramas, tablas e imágenes, se guardan en Google Drive, plataforma fiable y segura para el almacenamiento de archivos.

## 2.7. Caracterización del sistema de comunicaciones

La gestión eficaz de las comunicaciones es esencial para asegurar el correcto desarrollo del proyecto y mantener una coordinación fluida entre todas las partes involucradas. A continuación, se describen los principales canales a través de los cuales se mantendrá el contacto con los tutores y el cliente.

### 2.7.1. Relaciones con el cliente

La comunicación con los clientes se realizará de manera presencial en la mayoría de los casos, especialmente cuando se discutan aspectos clave del proyecto. Sin embargo, también se recurrirá a videoconferencias cuando las circunstancias lo requieran. De esta manera, se ofrece una mayor flexibilidad manteniendo conversaciones fluidas sin importar la distancia. Para concertar estas reuniones, se utilizarán canales de comunicación rápidos como *WhatsApp* o llamadas telefónicas.

### 2.7.2. Relaciones con los tutores

La comunicación con los tutores se llevará a cabo principalmente vía correo electrónico. No obstante, en ocasiones puntuales se organizarán reuniones presenciales para abordar discusiones más detalladas.



# CAPÍTULO 3

## Captura de Requisitos

En este capítulo se describen los requisitos del sistema, así como la representación gráfica de los casos de uso y del dominio de la aplicación. Estos elementos ofrecen una comprensión completa de las funcionalidades clave, la estructura interna del sistema y las interacciones entre sus diferentes componentes.

### 3.1. Requisitos funcionales

Los requisitos funcionales definen las capacidades clave que el sistema debe cumplir para satisfacer las necesidades del proyecto. A continuación, se muestran organizados por categorías.

#### 3.1.1. Gestión de usuarios

- **API para la gestión de usuarios:** El sistema debe permitir realizar operaciones CRUD (crear, leer, actualizar, eliminar) y funciones adicionales como el registro y autenticación de usuarios.
- **Registro de nuevos usuarios:** La aplicación permitirá a los usuarios registrarse proporcionando información básica, así como el nombre, apellidos, correo y contraseña.
- **Autenticación de usuarios:** Los usuarios podrán acceder a su cuenta iniciando sesión mediante su correo y contraseña.
- **Cierre de sesión:** En cualquier momento, los usuarios podrán terminar su sesión en la plataforma.
- **Configuración del perfil:** La aplicación contará con un apartado donde los usuarios podrán acceder y editar su información personal.

### 3. CAPTURA DE REQUISITOS

---

#### 3.1.2. Gestión de base de datos de abogados

- **API para la gestión de abogados:** El sistema debe permitir realizar principalmente operaciones CRUD (crear, leer, actualizar, eliminar) para gestionar la base de datos de abogados.
- **Cargar base de datos de abogados:** Los usuarios podrán cargar nuevos registros de abogados.
- **Ver y modificar base de datos de abogados:** La información de los abogados registrados podrá ser consultada y actualizada fácilmente por los usuarios.

#### 3.1.3. Gestión de base de datos de casos

- **API para la gestión de casos:** El sistema debe permitir realizar principalmente operaciones CRUD (crear, leer, actualizar, eliminar) para gestionar la base de datos de casos.
- **Cargar base de datos de casos:** Los usuarios podrán cargar nuevos registros de casos.
- **Ver y modificar base de datos de casos:** La información de los casos registrados podrá ser consultada y actualizada fácilmente por los usuarios.
- **Gestión de pagos:** La base de datos de los casos incluirá la consulta y actualización del estado de los pagos asociados a cada caso.

#### 3.1.4. Gestión de notificaciones

- **API para la gestión de notificaciones:** Se debe desarrollar un servicio que permita gestionar las notificaciones almacenadas localmente, incluyendo operaciones para obtener y mover notificaciones.
- **Visualización de notificaciones recibidas y enviadas:** Los usuarios podrán consultar desde la propia aplicación las notificaciones enviadas y recibidas almacenadas localmente.

#### 3.1.5. Envío de correos electrónicos

- **Extracción de datos clave de las notificaciones mediante OCR:** Se debe desarrollar un servicio que permita extraer información clave de los archivos PDF de las notificaciones, utilizando tecnología OCR.
- **Previsualización y envío automático de los correos electrónicos:** La plataforma contará con un sistema automatizado para el envío de correos electrónicos, que permitirá a los usuarios previsualizar el contenido antes de su envío.

### 3.1.6. Recursos y utilidades extra

- **Recopilación de recursos y enlaces de interés:** Se incluirá una sección con recursos útiles para los procuradores, como enlaces de interés y ubicaciones de juzgados.

### 3.1.7. Integraciones

- **Integración con los servicios de correo electrónico:** La aplicación permitirá al usuario enviar correos electrónicos directamente desde su interfaz, sin necesidad de acceder a plataformas externas de correo.
- **Integración de Google Maps:** La plataforma deberá permitir visualizar un mapa que muestre las ubicaciones de los juzgados del País Vasco.

## 3.2. Requisitos no funcionales

Los requisitos no funcionales describen las propiedades y características del sistema que garantizan su calidad.

### 3.2.1. Usabilidad

La interfaz de usuario debe ser intuitiva y fácil de utilizar, permitiendo a cualquier tipo de usuario interactuar con la aplicación sin complicaciones. El sistema debe garantizar una experiencia de usuario fluida y coherente en todas sus funcionalidades, manteniendo un diseño visual claro y organizado.

### 3.2.2. Rendimiento

El tiempo de respuesta para cualquier operación no debe exceder los 3 segundos, salvo para conexiones externas como la API de Google Maps o el servicio de correo electrónico. El sistema debe ser capaz de manejar hasta 5 usuarios concurrentes sin afectar al rendimiento. Además, debe tener la capacidad de procesar hasta 30 notificaciones diarias por usuario, garantizando así un funcionamiento ágil y eficiente.

### 3.2.3. Seguridad

El cifrado de los datos sensibles almacenados en la base de datos es esencial para proteger la información ante posibles amenazas. Además, el sistema debe contar con un proceso de autenticación robusto que verifique la identidad de los usuarios, asegurando un acceso controlado y seguro a la plataforma. Para garantizar una mayor seguridad en las interacciones con las APIs, se implementarán las siguientes medidas:

- Las peticiones realizadas para el registro o la autenticación del usuario deben incluir una API Key para asegurar que provienen de la propia aplicación.

### 3. CAPTURA DE REQUISITOS

---

- El sistema usará autenticación basada en JWT (JSON Web Token) para validar las peticiones una vez el usuario haya iniciado sesión.

#### 3.2.4. Mantenibilidad

La aplicación debe contar con un código fuente bien documentado que facilite su comprensión y modificación. Esto implica seguir buenas prácticas de programación, y herramientas como ESLint y Prettier pueden ayudar a mantener la calidad y coherencia del código. De esta manera, se facilitarán el mantenimiento y las futuras mejoras en la aplicación.

#### 3.2.5. Escalabilidad

El software debe tener un diseño modular que facilite la incorporación de nuevas funcionalidades sin necesidad de realizar cambios significativos. Además, la infraestructura debe poder expandirse de manera flexible en función de la demanda, asegurando que el sistema se adapte a futuros crecimientos sin comprometer su rendimiento.

#### 3.2.6. Plataforma e infraestructura

La plataforma será accesible desde cualquier navegador moderno, como Chrome, Firefox y otros. Además, se hospedará en un servicio en la nube confiable, lo que garantizará su disponibilidad y rendimiento. Se empleará MongoDB como base de datos NoSQL, lo que facilitará un almacenamiento y manejo eficiente de los datos.

#### 3.2.7. Estructura de organización local de notificaciones

Se implementará una estructura de carpetas en el repositorio local del usuario. Las notificaciones recibidas deberánadirse manualmente, y al ser enviadas, se reubicarán automáticamente.

### 3.3. Diagrama de casos de uso

El diagrama de casos de uso proporciona una representación gráfica de las interacciones entre los usuarios y el sistema. A través de este diagrama (ver figura 3.1), se puede visualizar de manera clara las acciones que los usuarios pueden realizar en la plataforma, así como las funcionalidades principales que ofrece el sistema.

A continuación, se describen los actores involucrados en el sistema y los casos de uso correspondientes.

- **Usuario no autenticado.**

- **Acceder.** Es el punto de inicio para los usuarios no autenticados, donde pueden optar por autenticarse o registrarse si aun no lo han hecho.

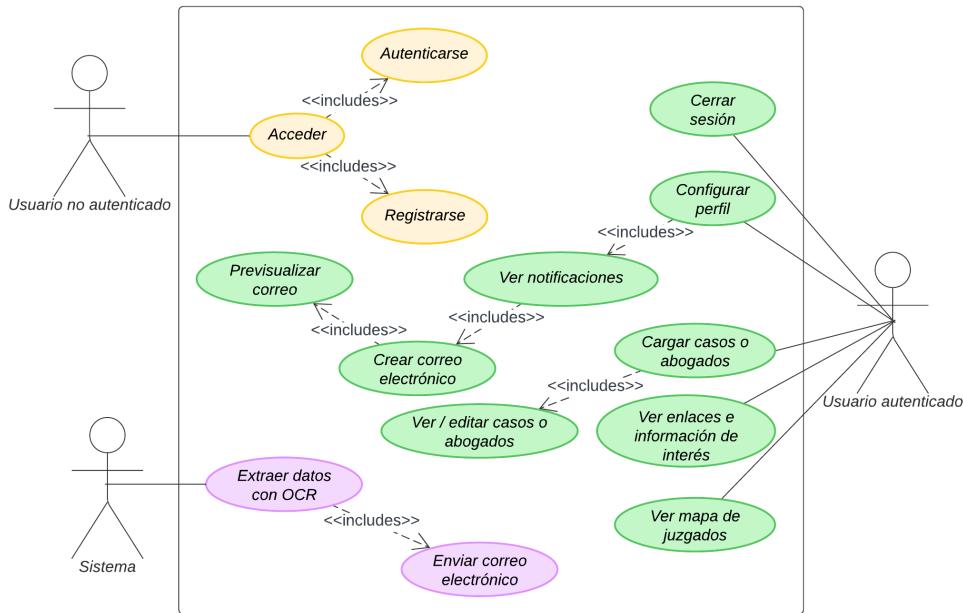


Figura 3.1: Diagrama de casos de uso

- **Autenticarse.** Permite a los usuarios registrados ingresar al sistema mediante sus credenciales (correo y contraseña).
- **Registrarse.** Opción disponible para usuarios no autenticados, que les permite crear una cuenta proporcionando su información básica (nombre, apellidos, correo y contraseña).

#### ■ Usuario autenticado

- **Cargar casos o abogados.** Permite añadir nuevos registros de casos o abogados al sistema.
- **Ver / editar casos o abogados.** Los usuarios pueden consultar y modificar la información de los casos o abogados registrados.
- **Configurar perfil.** Permite al usuario autenticado actualizar sus datos personales. Además, es necesario completar el perfil con información adicional, como la contraseña del correo electrónico o un directorio local, para habilitar el acceso a otras funcionalidades del sistema.
- **Ver notificaciones.** Los usuarios pueden consultar las notificaciones almacenadas en su directorio local desde la plataforma.
- **Crear correo electrónico.** Este caso de uso está vinculado con los procesos del sistema. Una vez que el usuario acciona el botón de crear correo electrónico, el sistema extrae automáticamente los datos relevantes de las notificaciones, completando de forma automática los campos del correo.

### 3. CAPTURA DE REQUISITOS

---

- **Previsualizar correo.** Antes de enviarlo, los usuarios pueden revisar el contenido del correo tal como será recibido, lo que les permite verificar que toda la información sea correcta antes de proceder con el envío.
- **Ver enlaces e información de interés.** Ofrece a los usuarios acceso a una lista de enlaces y recursos útiles, facilitando la consulta de información necesaria dentro de la plataforma.
- **Ver mapa de juzgados.** Proporciona un mapa interactivo que muestra la ubicación de los juzgados.
- **Cerrar sesión.** Permite finalizar la sesión de manera segura, terminando con el acceso hasta que el usuario vuelva a iniciar sesión.

#### ■ Sistema

- **Extraer datos con OCR.** Proceso automático en el que el sistema extrae información clave de archivos PDF (de las notificaciones) utilizando tecnología OCR, para su posterior uso.
- **Enviar correo electrónico.** Una vez creado y previsualizado, el sistema se encarga de enviar el correo electrónico automáticamente.

## 3.4. Diagrama de dominio

El diagrama de dominio representa los elementos clave del sistema y las relaciones entre ellos, permitiendo entender cómo interactúan los distintos componentes de la aplicación. Esta se muestra en la figura 3.2.

A continuación, se procede a explicar en detalle las entidades y relaciones que aparecen en el diagrama, proporcionando una descripción clara de su función y de cómo interactúan entre sí.

### 3.4.1. Entidades del dominio

Las principales entidades involucradas en el sistema son:

- **User:** Representa a los procuradores que hagan uso de la plataforma.
  - **\_id:** Identificador único del usuario (valor generado automáticamente).
  - **name:** Nombre del usuario.
  - **lastNames:** Apellidos del usuario.
  - **email:** Correo electrónico del usuario.
  - **password:** Contraseña del usuario.
  - **emailPassword:** Contraseña del correo electrónico del usuario.
  - **phone:** Número de teléfono del usuario.

### 3.4. Diagrama de dominio

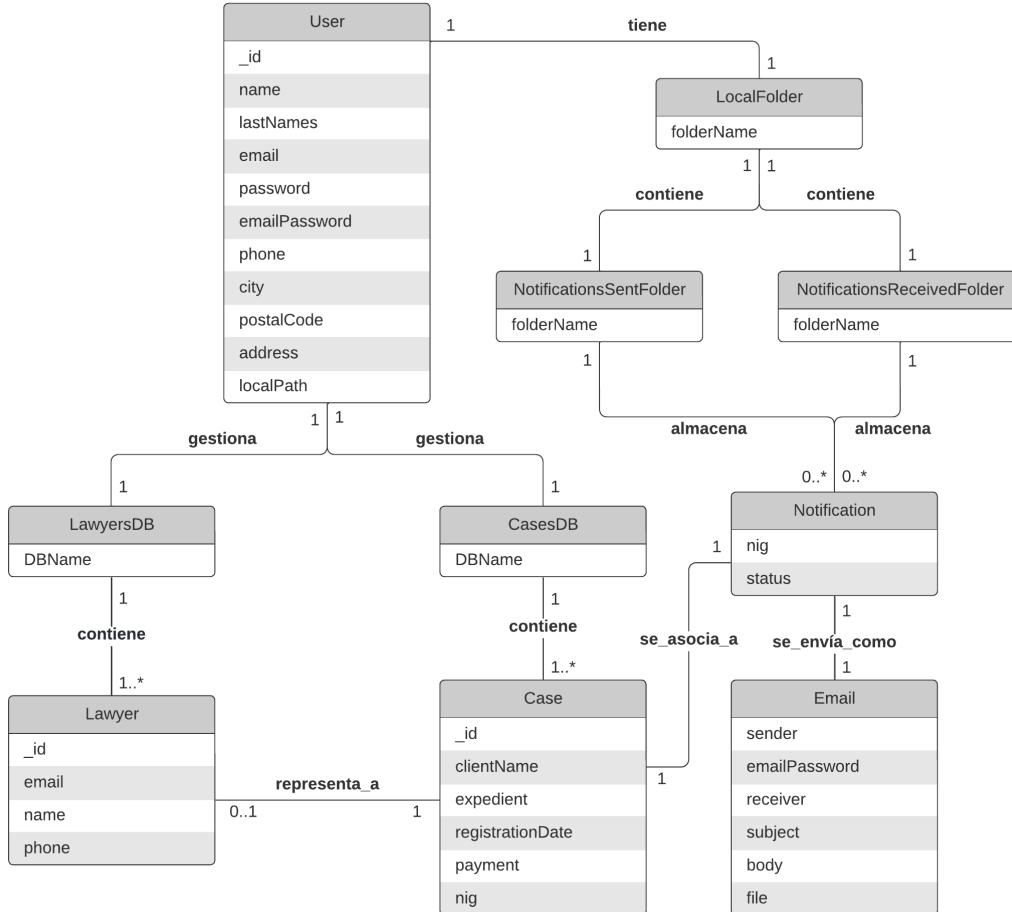


Figura 3.2: Diagrama de dominio

- **city**: Ciudad del usuario.
- **postalCode**: Código postal del usuario.
- **address**: Dirección del usuario.
- **localPath**: Ruta local donde se almacenan los documentos.
- **CasesDB**: Base de datos que almacena la información de los casos gestionados por el usuario.
  - **DBName**: Nombre de la base de datos de casos, construido por “user\_” seguido del **\_id** correspondiente al usuario propietario de la base de datos.
- **Case**: Representa los casos judiciales gestionados dentro del sistema.
  - **\_id**: Identificador único del caso (valor generado automáticamente).

### 3. CAPTURA DE REQUISITOS

---

- **clientName:** Nombre del cliente relacionado con el caso.
  - **expedient:** Expediente del caso.
  - **registrationDate:** Fecha de registro del caso.
  - **payment:** Información sobre el pago relacionado con el caso.
  - **nig:** Número identificador general del caso (NIG).
- **LawyersDB:** Base de datos que almacena la información de los abogados gestionados por el usuario.
    - **DBName:** Nombre de la base de datos de abogados, construido por “user\_” seguido del \_id correspondiente al usuario propietario de la base de datos.
  - **Lawyer:** Representa a los abogados asociados a los casos judiciales.
    - **\_id:** Identificador único del abogado (valor generado automáticamente).
    - **email:** Correo electrónico del abogado.
    - **name:** Nombre del abogado.
    - **phone:** Número de teléfono del abogado.
  - **Notification:** Almacena la información de las notificaciones relacionadas con los casos judiciales.
    - **nig:** Número de identificación general de la notificación (NIG).
    - **trialDate:** Fecha del juicio asociada a la notificación.
    - **status:** Estado de la notificación (enviada o recibida).
  - **Email:** Representa el correo electrónico utilizado para enviar notificaciones relacionadas con los casos.
    - **sender:** Remitente del correo.
    - **emailPassword:** Contraseña del correo del remitente.
    - **receiver:** Destinatario del correo.
    - **subject:** Asunto del correo.
    - **body:** Cuerpo del correo.
    - **file:** Archivo adjunto en formato PDF.
  - **LocalFolder:** Representa una carpeta en el directorio local del usuario donde se almacenan los documentos de las notificaciones.
    - **folderName:** Nombre de la carpeta local.
  - **NotificationsSentFolder:** Carpeta local que almacena las notificaciones enviadas por el usuario.

### 3.4. Diagrama de dominio

---

- **folderName:** Nombre de la carpeta de notificaciones enviadas.
- **NotificationsReceivedFolder:** Carpeta local que almacena las notificaciones recibidas por el usuario.
  - **folderName:** Nombre de la carpeta de notificaciones recibidas.

#### 3.4.2. Relaciones del dominio

A continuación se describen las relaciones entre las entidades del sistema.

- **User - LocalFolder:** Cada usuario **tiene** una carpeta local (LocalFolder) donde se almacenan los documentos y notificaciones.
- **LocalFolder - NotificationsSentFolder / NotificationsReceivedFolder:** La carpeta LocalFolder **contiene** a su vez las carpetas de notificaciones enviadas y recibidas.
- **NotificationsSentFolder / NotificationsReceivedFolder - Notification:** Las carpetas de notificaciones enviadas y recibidas **almacenan** las notificaciones judiciales. Estas carpetas pueden estar vacías.
- **Notification - Email:** Cada notificación **se envía como** un correo electrónico, que contiene la información relevante y el archivo PDF correspondiente.
- **User - CasesDB / LawyersDB:** Cada usuario **gestiona** dos bases de datos privadas, la de los casos y la de los abogados.
- **LawyersDB - Lawyer:** La base de datos de abogados **contiene** abogados.
- **CasesDB - Case:** La base de datos de casos **contiene** los casos gestionados por el usuario.
- **Case - Lawyer:** Un abogado puede **representar\_a** un caso, pero todos los casos tienen que ser representados por un abogado.
- **Case - Notification:** Cada caso **se asocia a** una notificación judicial.



# CAPÍTULO 4

## Herramientas y tecnologías

En esta sección se presenta un análisis de las herramientas y tecnologías para el desarrollo del proyecto. El objetivo es identificar aquellas que mejor se adapten a los requisitos del sistema, garantizando una implementación eficiente y un producto final de calidad.

Para llevar a cabo esta selección, se han considerado varios criterios clave, como la modernidad, popularidad en la industria, rendimiento, compatibilidad, facilidad de uso y experiencia previa con estas herramientas y tecnologías.

### 4.1. Herramientas generales

Este apartado recoge las herramientas fundamentales que facilitan el desarrollo del proyecto, incluyendo la creación de diagramas y tablas necesarias para el informe.

#### 4.1.1. Editor de código fuente

- **Visual Studio Code**<sup>1</sup>. Es un editor de código fuente ampliamente reconocido por su flexibilidad y rendimiento. Se trata de una herramienta ligera pero muy potente, que ofrece una amplia variedad de extensiones para adaptarse a las necesidades específicas de cada proyecto. Entre éstas, destaca su integración con Git, facilitando la gestión del control de versiones directamente desde el editor.

#### 4.1.2. Control de versiones

- **Git**<sup>2</sup>. Se ha seleccionado esta herramienta como el sistema de control de versiones debido a su robustez y flexibilidad. Permite gestionar cambios en el código de forma eficiente, con un historial detallado de todas las modificaciones.

---

<sup>1</sup><https://code.visualstudio.com/>

<sup>2</sup><https://git-scm.com/>

## 4. HERRAMIENTAS Y TECNOLOGÍAS

---

- **GitHub<sup>3</sup>**. Esta será la plataforma utilizada para alojar el repositorio de código de forma remota, permitiendo su acceso desde cualquier ubicación. Esto no solo actúa como copia de seguridad sino que también permite recuperar versiones anteriores si fuera necesario.

### 4.1.3. Diagramas y tablas

Para la creación de los diagramas se ha priorizado el uso de plataformas modernas que ofrezcan representaciones visuales claras y estéticamente agradables, facilitando notablemente su comprensión. En cuanto a las tablas, se ha optado por una herramienta más robusta.

- **LucidChart<sup>4</sup>**. Es una plataforma versátil para generar diagramas y mapas conceptuales, caracterizada por sus diseños limpios y profesionales, lo que facilita la representación visual de ideas complejas.
- **TeamGantt<sup>5</sup>**. Esta herramienta está diseñada específicamente para la planificación visual de proyectos mediante diagramas de Gantt. Ofrece una interfaz intuitiva y diversas funcionalidades para la gestión eficaz de tareas.
- **Microsoft Excel<sup>6</sup>**. Se trata de una herramienta robusta y confiable que se empleará para la creación de tablas.

## 4.2. Backend

Se ha realizado un análisis comparativo para seleccionar las herramientas y tecnologías que mejor se adapten al proyecto, priorizando la sencillez, la flexibilidad y la consistencia.

### 4.2.1. Lenguajes de programación

- **Java vs JavaScript vs Python [2]**

- **Java<sup>7</sup>**. A pesar de que este lenguaje es reconocido por su rendimiento y robustez, su enfoque orientado a objetos lo hace menos adecuado para proyectos que requieren flexibilidad y una estructura modular más simple. Mientras que Java permite organizar el código de manera estructurada, su sintaxis rígida y la necesidad de configuraciones complejas dificultan una implementación y adaptación rápida, siendo menos ideal para un desarrollo ágil.
- **JavaScript<sup>8</sup>**. Aunque originalmente fuera diseñado para el frontend, JavaScript también puede ejecutarse en el backend mediante Node.js. Esto ofrece la ventaja

---

<sup>3</sup><https://github.com/>

<sup>4</sup><https://www.lucidchart.com/>

<sup>5</sup><https://www.teamgantt.com/>

<sup>6</sup><https://www.microsoft.com/es-es/microsoft-365/excel>

<sup>7</sup><https://www.java.com/es/>

<sup>8</sup><https://www.javascript.com/>

de usar un solo lenguaje para ambos entornos, simplificando el desarrollo. Además, su amplio ecosistema le permite abordar las tareas del proyecto, convirtiéndolo en un candidato viable.

- **Python<sup>9</sup>**. Este lenguaje es reconocido por su sencillez y versatilidad. Su sintaxis clara facilita un desarrollo ágil, mientras que su enfoque modular permite una organización eficiente del código. Aunque JavaScript es una opción viable, Python ofrece una implementación igualmente flexible y rápida mediante frameworks como Flask, que destaca por su facilidad de uso y capacidad para crear APIs robustas y escalables. Esto, combinado con el enfoque más robusto de Python para tareas específicas del backend, lo convierte en una mejor opción.
- **Elección del lenguaje para el backend.** Se ha elegido **Python** como el lenguaje de programación para el backend debido a su sencillez, claridad y versatilidad, que permiten un desarrollo ágil y organizado. También destaca por su amplio ecosistema de bibliotecas que abordan de manera eficiente las tareas específicas del proyecto. Además, frameworks como Flask facilitan la creación de APIs robustas, asegurando una integración eficiente con el frontend.

#### 4.2.2. Gestor de dependencias

- **Virtualenv<sup>10</sup>**. Esta herramienta permite crear un entorno virtual donde gestionar de manera independiente las dependencias del backend. De esta manera, se asegura que las bibliotecas utilizadas no entren en conflicto con otros proyectos que puedan requerir versiones diferentes. Esto garantiza un entorno de desarrollo limpio y aislado, facilitando la réplica del mismo en otros sistemas.

#### 4.2.3. Frameworks

- **Flask<sup>11</sup>**. Se ha seleccionado este framework debido a su ligereza, facilidad de uso y flexibilidad, además de su compatibilidad con otras tecnologías. Es ideal para la construcción de APIs RESTful y el desarrollo de aplicaciones web ligeras y escalables.

#### 4.2.4. Bibliotecas

En el proyecto se utilizarán numerosas bibliotecas de Python para abordar las funcionalidades clave. Entre ellas, destacan las siguientes:

- **PyMongo<sup>12</sup>**, para la interacción con MongoDB.

---

<sup>9</sup><https://www.python.org/>

<sup>10</sup><https://virtualenv.pypa.io/>

<sup>11</sup><https://flask.palletsprojects.com/>

<sup>12</sup><https://pymongo.readthedocs.io/>

## 4. HERRAMIENTAS Y TECNOLOGÍAS

---

- **Flask-cors**<sup>13</sup>, para habilitar el intercambio de recursos entre diferentes dominios.
- **Flask-Mail**<sup>14</sup>, para el manejo de correos electrónicos.
- **Cryptography**<sup>15</sup> y **flask\_jwt\_extended**<sup>16</sup>, para temas relacionados con la seguridad.
- **Pdfplumber**<sup>17</sup>, para la extracción de datos de documentos PDF.

### 4.2.5. Bases de datos

- **MongoDB**<sup>18</sup>. Se ha elegido este sistema de base de datos NoSQL por su flexibilidad y escalabilidad en el manejo de datos no estructurados, siendo una opción perfecta para este proyecto.

## 4.3. Frontend

Para el frontend, se han comparado varias tecnologías para seleccionar las más adecuadas, enfocadas en ofrecer una experiencia de usuario dinámica e interactiva. Las herramientas elegidas permitirán crear interfaces modernas, optimizando el rendimiento y la flexibilidad durante el desarrollo.

### 4.3.1. Lenguajes de programación

#### ▪ JavaScript vs TypeScript [3]

- **JavaScript**<sup>19</sup>. Reconocido por su capacidad para crear interfaces interactivas y dinámicas, JavaScript permite modificar el contenido de una página sin necesidad de recargarla. Además, no solo es compatible con todas las plataformas y navegadores sino que ofrece múltiples librerías y frameworks que facilitan el desarrollo rápido y eficiente de aplicaciones web modernas.
- **TypeScript**<sup>20</sup>. Este lenguaje es un superconjunto de JavaScript que añade tipado estático y otras características avanzadas. Esto facilita la detección de errores y la comprensión y mantenimiento del código, especialmente en proyectos de gran volumen. Aunque es compatible con todas las bibliotecas y frameworks de JavaScript, requiere una configuración más compleja.

---

<sup>13</sup><https://pypi.org/project/Flask-Cors/>

<sup>14</sup><https://flask-mail.readthedocs.io/>

<sup>15</sup><https://cryptography.io/>

<sup>16</sup><https://flask-jwt-extended.readthedocs.io/>

<sup>17</sup><https://pypi.org/project/pdfplumber/>

<sup>18</sup><https://www.mongodb.com/>

<sup>19</sup><https://www.javascript.com/>

<sup>20</sup><https://www.typescriptlang.org/>

- **Elección del lenguaje principal para el frontend.** Se ha elegido **JavaScript** como el lenguaje de programación para el frontend porque se ajusta perfectamente a las necesidades del proyecto. Su capacidad para facilitar un desarrollo ágil, junto con su amplia compatibilidad con plataformas y navegadores, lo convierten en la opción ideal. Además, su flexibilidad y dinamismo permiten crear aplicaciones web modernas. Aunque TypeScript ofrece algunas ventajas, JavaScript proporciona todas las herramientas necesarias para cumplir con los requisitos del proyecto de forma más ágil y eficaz.
- **HTML<sup>21</sup>**. Es el lenguaje que define la estructura y el contenido de la aplicación web, organizando los distintos elementos que aparecen en la interfaz.
- **CSS<sup>22</sup>**. Se encarga de dar estilo a la aplicación, asegurando que los elementos se vean atractivos y consistentes.

#### 4.3.2. Gestor de dependencias.

- **Npm<sup>23</sup>**. Este gestor de paquetes facilita la instalación y gestión eficiente de las dependencias del frontend. No solo garantiza que las bibliotecas se mantengan organizadas y actualizadas, sino que también asegura la consistencia en las versiones de las dependencias a lo largo del desarrollo frontend. De esta manera, contribuye a un entorno más estable y controlado.

#### 4.3.3. Frameworks y bibliotecas principales

- **React vs Angular [4]**
  - **React<sup>24</sup>**. Se trata de una biblioteca de JavaScript enfocada en la creación de interfaces de usuario, que destaca por su flexibilidad y modularidad. Permite construir componentes reutilizables, lo que agiliza el desarrollo y el mantenimiento de aplicaciones web dinámicas. React destaca por su integración fácil con otras herramientas o bibliotecas y un ecosistema sólido, incluyendo herramientas como React Router para la navegación.
  - **Angular<sup>25</sup>**. Es un framework completo que utiliza TypeScript como base. Ofrece una solución integral para el desarrollo de aplicaciones web más robustas y escalables, con funcionalidades como el enrutamiento, la gestión de formularios, y renderizado en el servidor. Sin embargo, su mayor complejidad lo hace más adecuado para proyectos de gran envergadura que requieren una estructura más sólida.

<sup>21</sup><https://developer.mozilla.org/es/docs/Web/HTML>

<sup>22</sup><https://developer.mozilla.org/es/docs/Web/CSS>

<sup>23</sup><https://www.npmjs.com/>

<sup>24</sup><https://es.react.dev/>

<sup>25</sup><https://angular.dev/>

- **Elección del framework o biblioteca para el frontend.** Se ha optado por **React** como la biblioteca principal para el frontend, ya que ofrece un equilibrio ideal entre simplicidad y capacidad de personalización. Su enfoque modular facilita el desarrollo y mantenimiento de componentes reutilizables, mientras que su integración con diversas herramientas proporciona la flexibilidad necesaria para el proyecto. Además, React permite un renderizado dinámico eficiente, actualizando solo los elementos que cambian en la interfaz. Todo esto lo convierte en la opción más adecuada para el desarrollo ágil y eficaz que se busca.

#### 4.3.4. Herramientas de empaquetado

- **Webpack vs Vite [5]**

- **Webpack<sup>26</sup>**. Se trata de una de las herramientas de empaquetado más populares y ampliamente utilizadas en el desarrollo frontend, facilitando la creación de aplicaciones web. Sin embargo, durante el desarrollo puede ser menos eficiente, ya que sus procesos pueden ralentizar los tiempos de arranque, especialmente en proyectos grandes.
- **Vite<sup>27</sup>**. La rapidez de Vite lo convierte en una opción ideal para el desarrollo de aplicaciones modernas y dinámicas. Utiliza ES Modules, para importar y exportar código, y un servidor de desarrollo basado en demanda, lo que proporciona tiempos de arranque casi instantáneos. Al ser una herramienta más reciente y optimizada, Vite es una solución más eficiente para proyectos que necesitan un entorno de desarrollo ágil y de alto rendimiento.
- **Elección de la herramienta de empaquetado.** Se ha optado por **Vite** como la herramienta de empaquetado para este proyecto debido a su velocidad de arranque y eficiencia durante el desarrollo. Su capacidad para importar y exportar código desde el propio navegador lo hace ideal para entornos ágiles, convirtiéndolo en la mejor opción para este proyecto.

#### 4.3.5. Otras bibliotecas

En el proyecto se utilizarán diversas bibliotecas de JavaScript para abordar las funcionalidades clave del frontend. Entre ellas, destacan las siguientes:

- **PrimeReact<sup>28</sup>**, biblioteca de componentes de UI avanzados para React, que facilita la creación de interfaces interactivas y modernas.
- **PrimeIcons<sup>29</sup>**, conjunto de iconos diseñado específicamente para integrarse con PrimeReact y enriquecer la interfaz gráfica.

---

<sup>26</sup><https://webpack.js.org/>

<sup>27</sup><https://es.vitejs.dev/>

<sup>28</sup><https://primereact.org/>

<sup>29</sup><https://primeng.org/icons>

- **React Router DOM**<sup>30</sup>, para gestionar la navegación y enrutamiento de la aplicación en el navegador.
- **Axios**<sup>31</sup>, cliente HTTP basado en promesas, utilizado para hacer solicitudes a APIs de forma sencilla y eficiente.
- **JWT-decode**<sup>32</sup>, herramienta para decodificar JSON Web Tokens (JWT), generalmente utilizada para la autenticación de usuarios.
- **Formik**<sup>33</sup>, para la gestión de formularios en React, que facilita el manejo del estado y la validación.
- **Yup**<sup>34</sup>, para la validación de esquemas y datos, comúnmente utilizada junto con Formik para la validación de formularios.
- **ExcelJS**<sup>35</sup>, biblioteca que permite crear, leer y manipular archivos Excel en formato JavaScript.
- **@react-google-maps/api**<sup>36</sup>, para integrar fácilmente los servicios de Google Maps en aplicaciones React.
- **Vitest**<sup>37</sup>, para ejecutar pruebas de manera rápida y optimizada en proyectos con Vite.

---

<sup>30</sup><https://reactrouter.com/>

<sup>31</sup><https://axios-http.com/>

<sup>32</sup><https://www.npmjs.com/package/jwt-decode>

<sup>33</sup><https://formik.org/>

<sup>34</sup><https://www.npmjs.com/package/yup>

<sup>35</sup><https://www.npmjs.com/package/exceljs>

<sup>36</sup><https://www.npmjs.com/package/@react-google-maps/api>

<sup>37</sup><https://vitest.dev/>



# CAPÍTULO 5

## Diseño de la solución

En este capítulo se describen los aspectos fundamentales de la arquitectura del sistema, y se presentan los diseños de las diferentes partes de la aplicación.

### 5.1. Arquitectura del sistema

Para el diseño de la arquitectura de esta aplicación, se ha optado por el modelo MVC (Modelo-Vista-Controlador). Éste es una excelente opción para el proyecto, ya que permite organizar la aplicación de manera clara al separar las responsabilidades en distintas capas. Esta división facilita tanto el desarrollo estructurado como el mantenimiento y la escalabilidad del sistema, como se puede ver en la figura 5.1 obtenido desde [6]. Además, la utilización de herramientas modernas como React, Flask y MongoDB asegura un desarrollo ágil y eficiente, adaptado a las necesidades del proyecto. A continuación se describen las distintas capas que conforman este modelo [7].

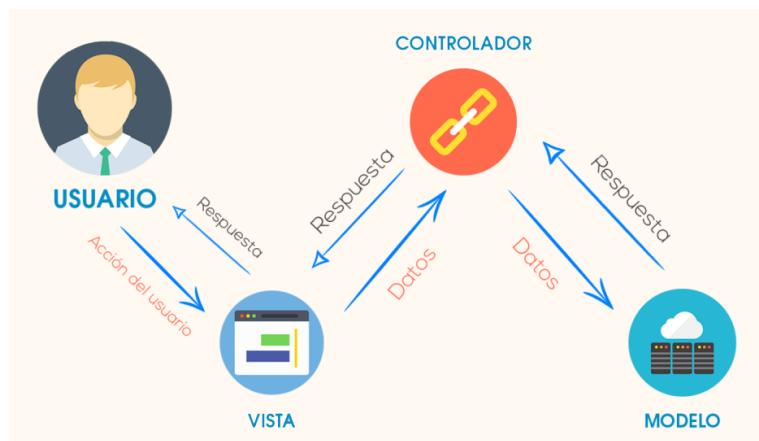


Figura 5.1: Modelo Vista Controlador

## 5. DISEÑO DE LA SOLUCIÓN

---

### 5.1.1. Capa del modelo

La capa del modelo es responsable de gestionar los datos y la lógica de negocio de la aplicación. En este proyecto, se ha elegido MongoDB como sistema de base de datos, una solución NoSQL que ofrece la flexibilidad necesaria para manejar datos no estructurados de manera eficiente. Esta capa se encargará de interactuar con la base de datos, realizando operaciones como la creación, lectura, actualización y eliminación de los datos (operaciones CRUD). Además, gracias a las capacidades de MongoDB, el sistema podrá gestionar grandes volúmenes de información de forma eficiente.

### 5.1.2. Capa de la vista

La capa de las vistas se encarga de mostrar los datos del modelo y gestionar las interacciones con el usuario. El uso de React, permite construir interfaces dinámicas y modernas utilizando un enfoque de componentes reutilizables que facilita la modularización. Esta estructura, además de agilizar el desarrollo y mantenimiento, permite una actualización dinámica de la interfaz, ya que solo se renderizan los elementos que cambian. Esto mejora el rendimiento y garantiza una experiencia de usuario rápida y fluida.

### 5.1.3. Capa del controlador

El controlador actúa como intermediario entre las capas del modelo y la vista, facilitando el intercambio de datos entre ambas. El uso de Flask permite crear APIs RESTful de manera sencilla y eficiente, lo que facilita esta comunicación a través de peticiones HTTP. El controlador recibe las peticiones del usuario a través de la vista, interactúa con el modelo para obtener o modificar datos, y finalmente actualiza la vista para reflejar los cambios. Gracias a la ligereza y flexibilidad de Flask, se asegura una integración fluida entre todas las partes del sistema.

## 5.2. Diseño de las bases de datos

Un buen diseño de las bases de datos es fundamental para garantizar un manejo eficiente de la información. En este proyecto se ha optado por MongoDB como sistema de gestión de bases de datos, ofreciendo una estructura flexible y eficiente para manejar datos no estructurados. En la figura 5.2 se muestra el diagrama que recoge la organización de las bases de datos y sus colecciones.

### 5.2.1. Base de datos general

Esta base de datos contiene una única colección, donde se almacena toda la información personal de los usuarios registrados (procuradores) en la aplicación.

- **Colección Users.** Esta colección guarda tanto los datos necesarios para la autenticación de los usuario, información relacionada con el envío de correos y un directorio local. Todos los atributos serán cadenas de texto (*str*), puesto que, además de la seguridad que ofrece la plataforma MongoDB Atlas, los datos se almacenan cifrados.

Base de datos general



Base de datos personal de cada usuario

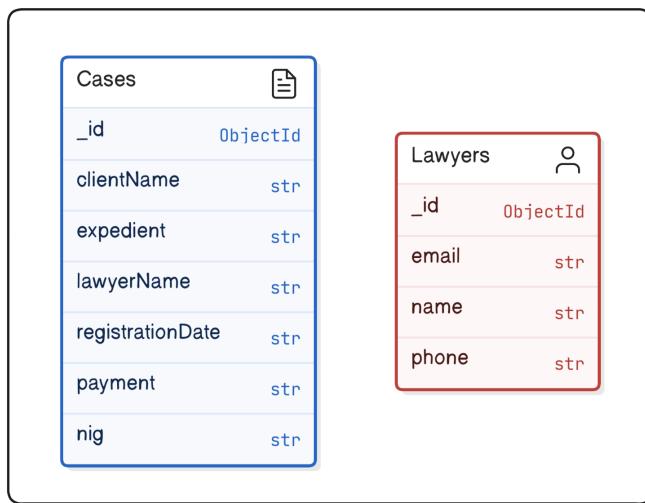


Figura 5.2: Diseño de las bases de datos

- **\_id.** Identificador único de tipo *ObjectId*, generado automáticamente por MongoDB.
- **name.** Nombre del usuario.
- **lastNames.** Apellidos del usuario.
- **email.** Correo electrónico del usuario.
- **password.** Contraseña del usuario.
- **emailPassword.** Contraseña del correo electrónico del usuario.
- **phone.** Número de teléfono del usuario.
- **city.** Ciudad del usuario.
- **postalCode.** Código postal del usuario.
- **address.** Dirección del usuario.
- **localPath.** Ruta local de almacenamiento para el usuario.

### 5.2.2. Base de datos personal de cada usuario

Cada procurador registrado en la aplicación tendrá asignada una base de datos personal, creada automáticamente al registrarse. Estas bases de datos contienen dos colecciones, “Cases”

## 5. DISEÑO DE LA SOLUCIÓN

---

y “Lawyers”, donde se almacenan datos relacionados con los casos gestionados y los abogados involucrados en ellos. El nombre de cada base de datos seguirá el formato: “user\_{\_id del usuario}”.

- **Colección Cases.** Esta colección almacena la información relevante de los casos gestionados por el usuario. Al igual que en la colección “Users”, todos los atributos se guardarán como cadenas de texto (*str*), ya que los datos estarán cifrados.
  - **\_id.** Identificador único de tipo *ObjectId*, generado automáticamente por MongoDB.
  - **clientName.** Nombre del cliente asociado al caso.
  - **expedient.** Número del expediente del caso.
  - **lawyerName.** Nombre del abogado asignado al caso.
  - **registrationDate.** Fecha en la que se registró el caso.
  - **payment.** Estado del pago relacionado con el caso, pudiendo ser “pendiente” o “completado”.
  - **nig.** Número de identificación general (NIG) del caso.
- **Colección Lawyers.** Esta colección almacena los datos de contacto de los abogados relacionados con los casos. Siguiendo el enfoque de las demás colecciones, los datos se guardarán como cadenas de texto (*str*), correspondiendo con el cifrado de los datos reales.
  - **\_id.** Identificador único de tipo *ObjectId*, generado automáticamente por MongoDB.
  - **name.** Nombre del abogado.
  - **email.** Correo electrónico del abogado.
  - **phone.** Número de teléfono del abogado.

### 5.3. Interfaz de usuario

La interfaz de usuario es un aspecto clave en la experiencia del usuario con la aplicación. Su diseño debe ser claro, intuitivo y funcional, permitiendo una navegación fluida.

#### 5.3.1. Criterios de diseño

El diseño de la interfaz de usuario seguirá varios principios clave que buscan ofrecer una experiencia intuitiva, accesible y visualmente atractiva. PrimeReact jugará un papel clave en la creación de componentes interactivos y modernos. Se trata de una biblioteca de react que proporciona un conjunto robusto de elementos de interfaz que asegurarán la consistencia y eficiencia en el diseño. A continuación, se detallan los criterios principales que guiarán el desarrollo de la interfaz:

- **Accesibilidad.** La aplicación permitirá acceder a todas las funcionalidades con pocos clics, optimizando la experiencia del usuario. Los botones y accesos serán visibles e intuitivos, garantizando una navegación fluida. Además, se proporcionará retroalimentación inmediata tras cada interacción para asegurar una comunicación clara sobre las acciones realizadas.
- **Simplicidad y claridad.** Se adoptará un estilo minimalista, mostrando únicamente los elementos necesarios para evitar la sobrecarga visual. El objetivo es ofrecer una navegación sencilla y centrada en las tareas clave del usuario.
- **Estilo visual y coherencia.** Se mantendrá una estética uniforme con el tema Lara-Light-Indigo de PrimeReact, que proporciona una paleta de colores moderna y equilibrada. Asimismo, PrimeIcons se utilizará para garantizar la consistencia visual en todos los iconos de la aplicación.
- **Modularidad y reutilización.** El diseño de la interfaz será modular, permitiendo reutilizar componentes en distintas secciones de la aplicación. De este modo, se garantiza la escalabilidad del sistema y su mantenimiento.

#### 5.3.2. Estructura general

La interfaz de usuario estará organizada en varias secciones que faciliten la navegación y accesibilidad de las principales funcionalidades.

- **Página de registro y autenticación.** Permitirá a los nuevos usuarios registrarse en la plataforma y a los ya existentes autenticarse de forma segura.
- **Barra de navegación.** Ubicada en la parte superior, esta barra ofrecerá acceso rápido a las diferentes páginas de la aplicación. Estará visible en todas las páginas destinadas a usuarios autenticados.
- **Página de gestión de bases de datos.** Los usuarios podrán gestionar sus bases de datos relacionadas con los casos y abogados desde esta sección.
- **Página de gestión de notificaciones y envío de correos.** Esta página facilitará la visualización de notificaciones y permitirá a los usuarios enviar los correos electrónicos correspondientes.
- **Página de información.** Este apartado mostrará información relevante para los procuradores, así como enlaces de interés y un mapa de juzgados.
- **Página de configuración del perfil.** Los usuarios podrán visualizar y modificar sus datos personales desde esta sección.

## 5. DISEÑO DE LA SOLUCIÓN

---

### 5.3.3. Componentes principales

La aplicación se compone de diversos elementos diseñados para facilitar la visualización de datos y las interacciones del usuario con la plataforma. A continuación, se describen los principales componentes que conformarán la aplicación.

- **Barra de navegación.** Facilitará el acceso rápido a las diferentes secciones de la plataforma para usuarios autenticados, garantizando una navegación fluida.
- **DataTables.** Mostrarán los datos de los casos y abogados en formato tabla, con opciones de filtrado, ordenación y paginación para una presentación clara y eficiente.
- **DataView.** Ofrecerá un formato visual más atractivo para la visualización de notificaciones.
- **Cards.** Se utilizará para agrupar y mostrar información de manera visual y compacta. Por ejemplo, la previsualización de los correos electrónicos, donde se muestren los detalles del mensaje en un *card* personalizado.
- **Formularios.** Recogerán, validarán y enviarán los datos del usuario en diferentes secciones de la aplicación, como en el registro y autenticación. Tendrán campos bien definidos con restricciones específicas para cada caso.
- **Botones.** Facilitarán las interacciones del usuario con el sistema, así como con el envío de formularios, la navegación y otras funcionalidades.
- **Mapa.** Mostrará la localización de los juzgados cercanos de manera interactiva, integrando el servicio de Google Maps para una visión precisa y atractiva.

## 5.4. Lógica de negocio

La lógica de negocio de este sistema se enfocará en los servicios relacionados con la seguridad, la gestión de las entidades, y la manipulación de las notificaciones judiciales, incluyendo el envío de correos electrónicos. A continuación se describen los principales servicios que se encontrarán en cada una de estas áreas.

### 5.4.1. Servicios de seguridad

El sistema implementará medidas de seguridad para proteger los datos sensibles y garantizar que solo los usuarios autorizados puedan acceder a su información.

- **Autenticación.** se implementará un servicio de autenticación basado en tokens JWT (JSON Web Token). Cuando el usuario acceda a la plataforma con sus credenciales, el sistema generará un token que se almacenará en la *session storage* con una validez de 2 horas. Tras este período, será necesario volver a iniciar sesión. Los tokens permitirán verificar la identidad del usuario en cada solicitud, garantizando la seguridad de la sesión.

- **Cifrado de datos.** Se implementará un servicio de cifrado simétrico AES para proteger la información privada almacenada. Estos datos solo serán descifrados cuando sean necesarios.

#### 5.4.2. Servicios CRUD para entidades

Las operaciones CRUD serán el núcleo de los servicios relacionados con los usuarios, casos y abogados. Estos servicios están diseñados para manejar de manera eficiente la información sensible y garantizar la integridad de los datos.

- **Creación de entidades.** Esta operación permitirá crear usuarios, casos y abogados, almacenándolos en las bases de datos correspondientes. Los usuarios se registrarán de forma individual. Sin embargo, para los casos y abogados, además de permitir la creación individual, también se podrán procesar listas de datos para generar varias entidades de manera simultánea, optimizando así el proceso de carga de información.
- **Lectura de entidades.** Esta lectura permitirá buscar las entidades del sistema, principalmente por su `_id` (identificador generado automáticamente por MongoDB). No obstante, también se podrán realizar búsquedas mediante otros atributos, como el nombre o correo para los usuarios y abogados, o el NIG para los casos.
- **Actualización de entidades.** Esta operación permitirá actualizar la información de las entidades del sistema. En el caso de los usuarios, se podrá completar la configuración del perfil y modificar cualquier dato personal. Para los casos y abogados, la actualización de datos se llevará a cabo directamente desde el DataTable donde se visualicen.
- **Eliminación de entidades.** La eliminación de entidades estará disponible únicamente para los casos y abogados, y se podrá realizar directamente desde el DataTable donde se visualicen.

#### 5.4.3. Servicios de notificaciones

La gestión de notificaciones es una de las funcionalidades clave del sistema. Para automatizar y optimizar distintos procesos, se implementarán servicios dedicados a la manipulación eficiente de estas notificaciones.

- **Creación de la estructura de carpetas.** Para organizar las notificaciones judiciales de forma eficiente, se implementará un servicio que generará una estructura de carpetas en el directorio local definido por el usuario. Estas carpetas clasificarán las notificaciones en enviadas y recibidas.
- **Extracción de datos mediante OCR.** Se desarrollará un servicio capaz de extraer los datos relevantes de los documentos PDF correspondientes a las notificaciones judiciales. Se empleará tecnología OCR para identificar datos clave para el posterior envío de correos electrónicos, así como el NIG.

## 5. DISEÑO DE LA SOLUCIÓN

---

- **Envío de correos electrónicos.** Este servicio recibirá la información necesaria para construir el correo electrónico, incluyendo el remitente, destinatario, asunto, cuerpo y el PDF de la notificación como archivo adjunto. se proporcionarán las credenciales del usuario para acceder a su cuenta de correo y realizar el envío, garantizando una comunicación segura.

# CAPÍTULO 6

## Desarrollo de la solución

En este capítulo se detalla el proceso de desarrollo de la solución y se presentan las principales implementaciones del sistema.

### 6.1. Configuración del entorno

Para asegurar un desarrollo eficiente y coherente, es esencial configurar adecuadamente el entorno de trabajo. De esta manera, se optimiza la integración de las herramientas, manteniendo la consistencia entre versiones y evitando posibles conflictos durante el desarrollo.

#### 6.1.1. Instalación de herramientas y dependencias

A la hora de comenzar con el desarrollo de la aplicación, algunas herramientas como Visual Studio Code, Git o Python, ya estaban instaladas. Sin embargo, fue necesario proceder con la instalación de las demás herramientas mencionadas en el capítulo 4.

#### 6.1.2. Configuración del entorno de desarrollo

Para garantizar un entorno de desarrollo adecuado y aislado se instalaron y configuraron dos herramientas específicas, virtualenv para el backend y npm para el frontend.

- **Virtualenv.** Con esta herramienta se creó un entorno virtual específico para la versión 3.12.3 de Python, con el nombre *venv*. Una vez configurado, se instalaron las dependencias necesarias para el backend, como el framework Flask y las bibliotecas de Python requeridas. Estas instalaciones se realizaron a través de pip, el gestor de paquetes de Python, que facilita la instalación de paquetes conforme se van necesitando durante el desarrollo. Además, todas las versiones de las dependencias han quedado registradas en el archivo *requirements.txt*, ubicado en la carpeta *backend*, para posibles consultas.

## 6. DESARROLLO DE LA SOLUCIÓN

---

- **Npm.** En el frontend, se utilizó npm como gestor de dependencias, a través del cual se instalaron y gestionaron todas las bibliotecas y herramientas necesarias. Al igual que en el backend, las versiones de las dependencias del frontend se han registrado automáticamente en los archivos package.json y package-lock.json, ubicados en la carpeta *frontend*. Por otro lado, también se generó la carpeta *node\_modules*, que contiene los archivos de las bibliotecas instaladas.

### 6.2. Desarrollo del backend

El desarrollo del backend ha sido una parte fundamental del proyecto, ya que se encarga de configurar el servidor y gestionar tanto la lógica de negocio como las conexiones con las bases de datos. Utilizando Python junto con el framework Flask, se han desarrollado varias APIs que permiten una comunicación eficiente con el frontend.

#### 6.2.1. Estructura del backend

Antes de comenzar a detallar el desarrollo del backend, es importante conocer la organización clara de los archivos y directorios que componen este apartado. La siguiente imagen 6.1 muestra esta estructura modular, la cual ha sido clave para lograr una implementación clara, escalable y mantenible.

A continuación, se describen las principales carpetas y archivos que se muestran en la imagen anterior.

- **src.** Es la carpeta principal del servidor, que agrupa los archivos esenciales para su funcionamiento.
  - **config.** Contiene los archivos de configuración que se encargan de gestionar las variables de entorno y la conexión a la base de datos.
  - **routes.** Agrupa las rutas o endpoints del servidor que gestionan las solicitudes HTTP del frontend.
  - **security.** Implementa los mecanismos de cifrado utilizados en el servidor.
  - **services.** Contiene los archivos que implementan la lógica de negocio del servidor.
  - **app.py.** Punto de entrada principal del servidor, donde se crea la instancia de Flask, se configuran las rutas y se inicializan las conexiones y bibliotecas necesarias.
- **venv.** Carpeta que contiene el entorno virtual de Python, donde se instalan las dependencias del servidor de manera aislada, evitando conflictos con otros proyectos.
- **.env.** Archivo que almacena las variables de entorno, incluyendo credenciales sensibles y configuraciones importantes que no deben incluirse en el código fuente.
- **.gitignore.** Lista los archivos y directorios que deben ser excluidos del control de versiones en Git, como las variables de entorno.



**Figura 6.1:** Estructura del backend

- **requirements.txt.** Contiene todas las versiones de las bibliotecas y frameworks del servidor, extraídas del entorno virtual.

### 6.2.2. Configuración del servidor

Se ha utilizado Flask para crear la aplicación del servidor, configurando elementos esenciales como la conexión a MongoDB, el manejo de autenticación mediante JWT y el control de acceso mediante CORS. Dado que la configuración incluye credenciales y claves sensibles, éstas se gestionaron como variables de entorno utilizando la biblioteca *dotenv*, lo que permitió mantenerlas separadas del código fuente.

Inicialmente, no se habilitó CORS, pero surgieron problemas relacionados con restricciones de origen al realizar peticiones entre el frontend y el backend. Para resolverlo, se integró este mecanismo mediante el paquete *flask\_cors*, que se adapta perfectamente a Flask al pertenecer al mismo ecosistema.

En el fragmento de código 6.1 se muestran las configuraciones más relevantes del servidor, distribuidas en los archivos app.py, ubicado en /backend/src, y en config.py y mongo.py,

## 6. DESARROLLO DE LA SOLUCIÓN

---

ubicados en /backend/src/config.

```
1  # Crear aplicación Flask
2  app = Flask(__name__)
3
4  # Cargar variables de entorno
5  load_dotenv()
6
7  # Configuración de la aplicación
8  app.config['MONGO_URI'] = os.getenv('MONGO_URI')
9  app.config['JWT_SECRET_KEY'] = os.getenv('JWT_SECRET_KEY')
10 app.config['JWT_ACCESS_TOKEN_EXPIRES'] = timedelta(hours=2)
11 app.config['CORS_HEADERS'] = 'Content-Type'
12
13 # Inicializar MongoDB
14 mongo = PyMongo()
15 mongo.init_app(app)
16
17 # Configurar JWT
18 jwt = JWTManager(app)
19
20 # Configurar CORS
21 CORS(app)
22
23
```

**Fragmento de código 6.1:** Configuración del servidor

### 6.2.3. Enrutamiento y controladores

En la aplicación se han definido rutas principales para gestionar las diferentes APIs relacionadas con los usuarios, casos, abogados y notificaciones. En el fragmento de código 6.2 se muestran las rutas principales definidas en el archivo app.py.

Estas APIs actúan como puntos de acceso dentro del sistema, permitiendo que el frontend interactúe con el backend a través de peticiones HTTP. Cada API gestiona un conjunto de rutas específicas que controlan la lógica de negocio correspondiente a su entidad, facilitando la modularización de los componentes y mejorando la escalabilidad del sistema.

Estas rutas específicas llaman a los servicios definidos, los cuales cubren principalmente las operaciones CRUD para cada entidad. Sin embargo, también se han implementado servicios adicionales como la extracción de datos mediante OCR o el envío de correos electrónicos. Todas las rutas están protegidas por mecanismos de seguridad que garantizan que solo usuarios autenticados puedan acceder a los servicios, asegurando además que las peticiones provengan del frontend autorizado.

Para gestionar las rutas, como se muestra en el fragmento de código 6.3 cada API sigue una estructura común basada en el uso de Blueprints, que agrupan todas las rutas relacionadas

con una entidad. A partir de estos Blueprints, se definen las rutas indicando el endpoint y el método HTTP correspondiente (GET, POST, PUT, DELETE). Cada ruta está vinculada a una función que llama al servicio asociado, encargada de procesar la lógica de negocio.

```
1 # Registrar rutas de usuarios
2 app.register_blueprint(users, url_prefix='/users')
3
4 # Registrar rutas de abogados
5 app.register_blueprint(lawyers, url_prefix='/lawyers')
6
7 # Registrar rutas de casos
8 app.register_blueprint(cases, url_prefix='/cases')
9
10 # Registrar rutas de notificaciones
11 app.register_blueprint(notifications, url_prefix='/notifications')
12
13
```

**Fragmento de código 6.2:** Registro de rutas principales

```
1 # Crear un Blueprint para manejar las rutas de abogados
2 lawyers = Blueprint('lawyers', __name__)
3
4 # Definir la ruta para crear un nuevo abogado
5 @lawyers.route('/', methods=['POST'])
6 @jwt_required()
7 def createLawyer():
8     return createLawyerService()
9
10 # Definir la ruta para obtener un abogado dado su id
11 @lawyers.route('/<id>', methods=['GET'])
12 @jwt_required()
13 def getLawyer(id):
14     return getLawyerService(id)
15
16
```

**Fragmento de código 6.3:** Ejemplo de rutas de la API de abogados

### 6.2.4. Seguridad del servidor

Para garantizar la protección de los servicios y datos sensibles de la aplicación, se han implementado varios mecanismos de seguridad. Estos incluyen sistemas de autenticación y autorización, así como el cifrado de la información.

#### 6.2.4.1. Autenticación y autorización

La seguridad en el acceso a las APIs se gestiona mediante la autenticación basada en JWT (JSON Web Token) y la autorización mediante API Keys. Estos mecanismos se aplican

## 6. DESARROLLO DE LA SOLUCIÓN

---

mediante los decoradores `@require_api_key` y `@jwt_required()`, situados sobre las funciones definidas en cada ruta específica, que verifican la clave o token. En el fragmento de código 6.3, se puede observar un ejemplo del uso del decorador `@jwt_required()` para la autenticación.

Para los servicios relacionados con el registro y el inicio de sesión de los usuarios, se utiliza una API Key para autorizar el acceso, ya que en estos casos el usuario aún no cuenta con un token JWT. Una vez el usuario ha iniciado sesión, se le asigna un token JWT, que se utiliza para autenticar todas las solicitudes posteriores. Ambos mecanismos precisan de una clave, que fue generada aleatoriamente para cada uno y almacenada de forma segura en el archivo `.env`.

Es importante destacar que, como se muestra en el fragmento de código 6.4, al generar el token JWT, se utiliza el `_id` del usuario, de manera que a través de este token más adelante se pueda recuperar el identificador.

```
1 # Generar un token JWT para el nuevo usuario
2 access_token = create_access_token(identity=str(user_id))
3
4
```

**Fragmento de código 6.4:** Asignación de token JWT al usuario

### 6.2.4.2. Cifrado de datos

Además del algoritmo simétrico AES planteado para el cifrado de la información, se ha complementado con un mecanismo de cifrado hash para las contraseñas de los usuarios.

- **Cifrado AES.** El cifrado AES se ha implementado mediante la biblioteca `cryptography`, con una clave segura generada y almacenada de manera segura como variable de entorno, en el archivo `.env`. Este mecanismo se utiliza para proteger información que debe ser recuperada, como el teléfono y dirección de los usuarios, aplicando el cifrado antes de almacenarlos y el descifrado cuando se necesitan (fragmento de código 6.5).

```
1 # Función para cifrar datos
2 def encrypt_data(self, data):
3     return self.cipher_suite.encrypt(data.encode('utf-8')).decode('utf-8')
4
5
6 # Función para descifrar datos
7 def decrypt_data(self, encrypted_data):
8     return self.cipher_suite.decrypt(encrypted_data.encode('utf-8')).decode('utf-
9 -8')
```

**Fragmento de código 6.5:** Cifrado y descifrado mediante AES

- **Cifrado Hash.** Para las contraseñas de los usuarios, se ha optado por un cifrado hash utilizando la biblioteca bcrypt, tal y como se muestra en el fragmento de código 6.6. A diferencia del cifrado simétrico, el hash no requiere que las contraseñas originales sean recuperadas, solo verificadas. Bcrypt genera un hash único para cada contraseña, que se almacena de manera segura en la base de datos. Durante el proceso de autenticación, se aplica nuevamente la función hash a la contraseña proporcionada por el usuario, y se compara con el hash almacenado para confirmar su validez.

```

1      # Función para hashear datos (como contraseñas)
2      def hash_data(self, data):
3          hashed_data = bcrypt.hashpw(data.encode('utf-8'), bcrypt.gensalt())
4          return hashed_data.decode('utf-8')
5
6      # Función para verificar datos contra un hash
7      def verify_data(self, data, hashed_data):
8          return bcrypt.checkpw(data.encode('utf-8'), hashed_data.encode('utf-8'))
9
10

```

**Fragmento de código 6.6:** Cifrado y verificación mediante hash

## 6.2.5. Conexión a la base de datos

Una vez configurada la conexión a MongoDB, se han tratado las consultas a las diferentes bases de datos.

### 6.2.5.1. Conexión y gestión de la base de datos de usuarios

En el caso de los usuarios, la conexión a la base de datos general de la aplicación se ha configurado automáticamente antes de cada solicitud mediante el decorador `@users.before_request`. Éste se ejecuta antes de cada petición, asignando la base de datos general, llamada `App_Web_Procuradores`, que contiene la colección `Users`. Este enfoque no solo simplifica el código, sino que también garantiza que las operaciones se ejecuten sobre la colección correspondiente, tal y como se muestra en el fragmento de código 6.7.

### 6.2.5.2. Conexión y gestión de las bases de datos privadas de cada usuario

Para las consultas de los casos y abogados, como se muestra en el fragmento de código 6.8 se ha implementado un sistema similar, con la diferencia de que antes de cada solicitud se obtiene el `_id` del usuario que realiza la petición. Esto es posible gracias a que el token se generó mediante el `_id` del usuario, lo que permite recuperarlo.

Este identificador es necesario para asignar la base de datos privada correspondiente, ya que el nombre de dichas bases de datos sigue el formato: “`user_` + `_id`”. De este modo, cada

## 6. DESARROLLO DE LA SOLUCIÓN

---

usuario accede únicamente a su base de datos, garantizando que las consultas se realicen sobre su propia información de casos y abogados.

De esta forma, cada usuario accede exclusivamente a su base de datos privada, lo que garantiza que la información se mantenga aislada y protegida. A continuación, se muestra cómo se obtiene y asigna la base de datos privada correspondiente antes de cada solicitud.

```
1  # Configurar la base de datos antes de cada solicitud
2  @users.before_request
3  def set_database():
4      g.db = get_db('App_Web_Procuradores')
5
6
7  # Obtener un usuario dado su id
8  def getUserService(id):
9      user = g.db.users.find_one({'_id': ObjectId(id)})
10
11     ## Aquí se realiza el descifrado de datos
12
13     # Devolver el json del usuario
14     return jsonify({
15         '_id': str(user['_id']),
16         'name': user['name'],
17         ## Lo mismo para los demás atributos
18     })
19
```

**Fragmento de código 6.7:** Ejemplo de conexión y consulta de usuario en MongoDB

```
1  # Configurar la base de datos antes de cada solicitud
2  @lawyers.before_request
3  @jwt_required()
4  def set_database():
5      # Obtener el user_id del token JWT
6      user_id = get_jwt_identity()
7
8
9      # Establecer la base de datos en g.db
10     g.db = get_db(f"user_{user_id}")
11
```

**Fragmento de código 6.8:** Ejemplo de extracción del \_id del usuario y asignación de la BD

### 6.2.6. Implementación de la lógica de negocio

Una vez definidas las rutas y controladores en la aplicación, cada archivo de rutas está vinculado a un archivo de servicios, encargado de gestionar la lógica de negocio correspondiente.

## 6.2. Desarrollo del backend

Estos servicios interactúan directamente con la base de datos y ejecutan las operaciones necesarias.

Todos los archivos de servicios del backend incluyen las operaciones CRUD, esenciales para la gestión de las entidades del sistema: crear, leer, actualizar y eliminar. Dado que estas operaciones siguen un patrón similar para todas las entidades. En los fragmentos de código [6.9](#) y [6.10](#) se presentan algunos ejemplos.

```
1  # Crear nuevo abogado
2  def createLawyerService():
3      # Obtener los datos de la petición
4      data = request.json
5
6
7      ## Aquí se realizan verificaciones y se cifran los datos obtenidos
8
9      # Insertar el nuevo abogado en la base de datos
10     id = g.db.lawyers.insert_one(data).inserted_id
11
12    ## Se hace un control de errores y se devuelve un mensaje
13
```

**Fragmento de código 6.9:** Servicio para crear un abogado

```
1  # Actualizar información de un caso dado su id
2  def updateCaseService(id):
3      # Obtener los datos de la petición
4      data = request.json
5
6
7      ## Aquí se realizan verificaciones y se cifran los datos obtenidos
8
9      # Actualiza el caso en la base de datos
10     result = g.db.cases.update_one({'_id': ObjectId(id)}, {"$set": data})
11
12    ## Se hace un control de errores y se devuelve un mensaje
13
```

**Fragmento de código 6.10:** Servicio para actualizar un caso

Aunque estas operaciones CRUD son comunes en los archivos de servicios *users\_services.py*, *lawyers\_services.py* y *cases\_services.py*, es importante señalar que las notificaciones siguen un enfoque distinto. Dado que éstas se gestionan de manera local y no se almacenan en MongoDB, sus operaciones no implican el uso de estas funciones CRUD tradicionales.

Además de las operaciones mencionadas, todos los archivos de servicios incluyen una serie de funciones adicionales que cubren necesidades específicas del sistema. A continuación se detallan estos servicios complementarios.

## 6. DESARROLLO DE LA SOLUCIÓN

---

### ▪ Funciones adicionales del archivo `users_services.py`

- **email\_exists(email).** Verifica si el correo electrónico ya está registrado durante la creación o actualización de un usuario.
- **getAllUsersService().** Recupera todos los usuarios almacenados en la base de datos.
- **loginUserService().** Gestiona el proceso de inicio de sesión de los usuarios. En primer lugar busca en la base de datos si el correo electrónico está registrado. Si lo encuentra, compara la contraseña proporcionada con el hash almacenado en la base de datos utilizando el servicio de cifrado hash. Si la verificación es correcta, se genera un token JWT único basado en el identificador del usuario, permitiendo autenticar futuras solicitudes del mismo.
- **addTrialToUserService(id).** Esta función añade un juicio a un usuario específico, verificando que los datos necesarios (fecha, hora y lugar) estén presentes. Luego, cifra la información del juicio y la agrega al campo trials del usuario en la base de datos.

### ▪ Funciones adicionales del archivo `lawyers_services.py`

- **getLawyerByNameService(name).** Obtiene un abogado dado su nombre, en vez de el `_id`.
- **getAllLawyersService().** Recupera todos los abogados almacenados en la base de datos.
- **email\_exists(email).** Verifica si el correo electrónico ya está registrado durante la creación o actualización de un abogado.
- **uploadLawyersService().** Recibe una lista de abogados a través de la petición y los carga en la base de datos, verificando que no hayan duplicados. Gestiona tanto advertencias como posibles errores que puedan surgir durante el proceso, devolviendo un objeto JSON con los campos `message`, `created_ids`, `errors` y `warnings`. El campo `message` indica el estado general de la operación, `created_ids` contiene los identificadores de los abogados creados, `errors` lista los errores encontrados asociados a cada abogado y `warnings` detalla las advertencias, como los abogados que ya estaban registrados.
- **checkLawyersDataService().** Devuelve un valor booleano indicando si al menos hay un abogado registrado en la base de datos. Esta función se utiliza para que, al representar los abogados en una tabla en el frontend, si no hay registros, se muestre la tabla vacía junto con un mensaje de advertencia.

### ▪ Funciones adicionales del archivo `cases_services.py`

- **getCaseByNIGService(nig).** Obtiene un caso dado su nig, en vez de el `_id`.
- **getAllCasesService().** Recupera todos los casos almacenados en la base de datos.

- **case\_exists(nig).** Verifica si el nig ya está registrado durante la creación o actualización de un caso.
- **uploadCasesService().** Recibe una lista de casos a través de la petición y los carga en la base de datos, verificando que no hayan duplicados. Gestiona tanto advertencias como posibles errores que puedan surgir durante el proceso, devolviendo un objeto JSON con los campos *message*, *created\_ids*, *errors* y *warnings*. El campo *message* indica el estado general de la operación, *created\_ids* contiene los identificadores de los casos creados, *errors* lista los errores encontrados asociados a cada caso y *warnings* detalla las advertencias, como los casos que ya estaban registrados.
- **checkCasesDataService().** Devuelve un valor booleano indicando si al menos hay un caso registrado en la base de datos. Esta función se utiliza para que, al representar los casos en una tabla en el frontend, si no hay registros, se muestre la tabla vacía junto con un mensaje de advertencia.

- **Funciones del archivo notifications\_services.py**

- **getFileService(directory\_path, fileName).** Recibe como parámetros de entrada el *directory\_path*, que indica la ruta del directorio local donde se encuentra el archivo, y *fileName*, que es el nombre del archivo a obtener. Esta función busca el archivo en el directorio especificado y, si lo encuentra, lo devuelve como archivo descargable.
- **getFileListService(directory\_path).** Recibe como parámetro de entrada *directory\_path*, que indica la ruta del directorio local donde se buscarán los archivos. Esta función devuelve una lista de todos los archivos que se encuentran en esa ruta.
- **getNigFromFileService(file\_path).** Recibe como parámetro de entrada *file\_path*, que corresponde a la ruta del archivo PDF del cual se tratará de extraer el número de identificación general (NIG). La función procesa el contenido del PDF utilizando la librería *pdfplumber* para extraer el texto de cada página. Una vez extraído el texto, se busca un patrón específico utilizando una expresión regular que identifica un número de 19 dígitos (formato del NIG) precedido de la palabra “NIG”. Finalmente, si ha encontrado el valor, lo devuelve.
- **deleteFileService(file\_path).** Recibe como parámetro de entrada *file\_path*, que corresponde a la ruta del archivo PDF que se desea eliminar. Si la función lo encuentra, lo elimina.
- **createFoldersService(directory\_path).** Recibe como parámetro de entrada *directory\_path*, que corresponde con la ruta del directorio local donde se va a crear la estructura de carpetas. La función comprueba si existen las carpetas “Notificaciones recibidas” y “Notificaciones enviadas” dentro del directorio especificado, y, si no existen, las crea automáticamente, asegurando que la estructura esté correctamente configurada.

## 6. DESARROLLO DE LA SOLUCIÓN

---

- **sendEmailService(sender, password, recipient, subject, htmlBody, filePath)**. Esta función recibe los siguientes parámetros de entrada: el correo electrónico del remitente (*sender*), la clave asociada (*password*), el destinatario (*recipient*), el asunto del correo (*subject*), el cuerpo en formato HTML (*htmlBody*), y la ruta completa del archivo PDF correspondiente a la notificación que se ha de enviar (*filePath*). Utiliza la librería *Flask-Mail* para configurar el servidor de correo, definiendo parámetros como el servidor SMTP basado en el dominio del remitente, el puerto 587 para conexiones seguras, la habilitación de TLS para garantizar la seguridad en la transmisión de datos, y las credenciales del correo. Tras configurar estos valores, se construye el mensaje con los detalles especificados, se adjunta el archivo y se procede a enviar el correo electrónico al destinatario indicado.
- **extractDateFromPDFService(file\_path)**. Esta función se encarga de extraer, si está presente, la fecha, hora y lugar del juicio asociado a partir de un archivo PDF utilizando técnicas de OCR. Este proceso resulta más complejo que la extracción del NIG, ya que requiere identificar múltiples detalles específicos que pueden no ser únicos en todo el documento, como fechas relacionadas con otros sucesos.  
En primer lugar, la función filtra un fragmento de texto que suele contener los datos asociados a un juicio mediante una expresión regular flexible, donde partes del fragmento son variables. Después, limpia el texto eliminando espacios y tabulaciones innecesarias, y aplica patrones específicos para localizar la fecha, hora y lugar. Si finalmente se han encontrado coincidencias, se devuelven, de lo contrario, se indicará que el documento no contiene ningún juicio asociado.
- **moveFileService(file\_path, target\_directory)**. Esta función recibe como parámetros de entrada el *file\_path*, que indica la ruta completa del archivo a mover, y el *target\_directory*, correspondiente con la ruta del directorio de destino.

### 6.2.7. Verificaciones y control de errores

Durante el desarrollo de la lógica de negocio, se ha puesto especial énfasis en la verificación de los datos y en el manejo adecuado de los errores. En los fragmentos de código mostrados en los apartados anteriores se ha suprimido esta parte para simplificar y centrar la atención en los aspectos clave de cada función. Sin embargo, en la implementación real se han realizado estas verificaciones y controles de manera robusta.

#### 6.2.7.1. Verificación de datos

Antes de procesar cualquier operación en los servicios del backend, se llevan a cabo verificaciones sobre los datos recibidos. Aunque la validación de los datos se lleve a cabo en gran medida en el frontend, el backend se encarga de realizar alguna comprobación adicional. En los siguientes puntos se detallan las principales verificaciones que se realizan para asegurar que las operaciones se ejecuten de manera correcta y segura.

- **Verificación de datos requeridos.** Antes de procesar cualquier operación, se verifica que todos los campos obligatorios estén presentes en los datos recibidos.
- **Verificación de rutas y archivos existentes.** Para operaciones que involucran rutas de archivos o directorios locales, el sistema comprueba que éstos existan y sean accesibles.
- **Verificación de duplicados en las bases de datos.** Al realizar acciones como la creación o actualización de usuarios, abogados o casos, se verifica que los datos no estén ya registrados en la base de datos para evitar inconsistencias.

### 6.2.7.2. Control de errores

El sistema ha sido diseñado para gestionar de manera adecuada posibles errores que puedan surgir durante las operaciones, garantizando así la estabilidad de la aplicación. A continuación, se describen los mecanismos implementados en los servicios del backend.

- **Manejo de excepciones.** En ciertos casos donde es probable que ocurran fallos, como al enviar correos electrónicos, crear carpetas en el sistema de archivos, o extraer datos de un archivo PDF, se utilizan bloques try-except para capturar posibles excepciones. Estos errores se gestionan de forma controlada, devolviendo un mensaje de error claro al frontend sin interrumpir el funcionamiento del sistema.
- **Estructura de respuestas.** Todas las respuestas de las operaciones del sistema, ya sean exitosas o fallidas, siguen una estructura consistente en formato JSON. Incluyen un mensaje que describe el estado de la operación, y cuando es necesario se devuelven los datos solicitados. Además, se utilizan códigos de estado HTTP para indicar el resultado de la operación.
- **Manejo de errores en operaciones con múltiples elementos.** En los servicios que gestionan múltiples elementos, como sucede al cargar una lista de abogados o casos, es posible que algunos elementos se procesen correctamente y que otros generen errores. Para estos casos, la estructura de retorno no solo incluye un mensaje de estado general, sino también detalles específicos sobre qué elementos fallaron y por qué. Estos errores también se organizan en un objeto JSON que contiene listas de éxitos, advertencias y errores. Esto permite al frontend procesar de manera eficiente las respuestas y mostrar detalladamente los errores o advertencias ocurridos.

En el ejemplo 6.11 se muestra la implementación de las verificaciones y el control de errores en el servicio de carga de casos, destacando cómo se gestionan tanto las operaciones exitosas como los fallos y advertencias durante el procesamiento de múltiples entidades.

## 6. DESARROLLO DE LA SOLUCIÓN

---

```
1  # Cargar una lista de casos
2  def uploadCasesService():
3      data = request.json
4      required_fields = ['cliente', 'expediente', 'letrado', 'dato_en_fecha', 'pago', 'nig']
5      created_ids = []
6      errors = []
7      warnings = []
8
9      for case in data:
10          # Verificar que todos los campos requeridos estén presentes
11          if not all(field in case for field in required_fields):
12              errors.append({'case': case, 'error': 'Faltan_campos_requeridos'})
13              continue
14
15          # Verificar que el caso no esté registrado
16          if case_exists(case['nig']):
17              warnings.append({'case': case, 'message': 'El_caso_ya_existe'})
18              continue
19
20          ## Aquí se cifran los datos y se inserta el caso en la BD
21
22          created_ids.append(str(id))
23
24          if len(errors) == 0 and len(warnings) == 0:
25              # Todos los casos se cargaron correctamente
26              return jsonify({
27                  'message': 'Todos_los_casos_cargados_correctamente',
28                  'created_ids': created_ids
29              }), 201
30
31          elif len(errors) == 0 and len(warnings) > 0:
32              return jsonify({
33                  'message': 'Algunos_casos_ya_estaban_registrados',
34                  'created_ids': created_ids,
35                  'warnings': warnings
36              }), 200
37
38          elif len(errors) > 0 and len(created_ids) > 0:
39              return jsonify({
40                  'message': 'Proceso_completado_parcialmente',
41                  'created_ids': created_ids,
42                  'errors': errors,
43                  'warnings': warnings
44              }), 207 # 207 Multi-Status
45
46      else:
47          return jsonify({
48              'message': 'Ningún_caso_fue_cargado',
49              'errors': errors
50          }), 400
```

**Fragmento de código 6.11:** Ejemplo completo de verificación de datos y control de errores

## 6.3. Desarrollo del frontend

El desarrollo del frontend también desempeña un papel clave en el proyecto, ya que se encarga de la interacción directa con los usuarios y de la presentación clara y accesible de los datos procesados por el backend. Utilizando React como framework principal, se ha implementado una interfaz modular y eficiente, que asegura una experiencia de usuario fluida. Por otro lado, Vite ha proporcionado un entorno de desarrollo ágil, reduciendo los tiempos de arranque y permitiendo un flujo de trabajo más rápido y eficiente.

### 6.3.1. Estructura del frontend

Antes de entrar en detalle sobre el desarrollo del frontend, hay que comprender como se organizan los archivos y directorios que componen este apartado (ver figura 6.2). La siguiente imagen muestra esta estructura que ha permitido un desarrollo ágil y ordenado, facilitando la implementación de las distintas funcionalidades.

A continuación, se describen las principales carpetas y archivos que se muestran en la imagen anterior.

- **node\_modules.** Carpeta que contiene las dependencias instaladas mediante npm, definidas en package.json. Estas bibliotecas son esenciales para el desarrollo y ejecución de la aplicación.
- **public.** Almacena archivos estáticos como imágenes y otros recursos que se utilizan directamente en la aplicación, sin necesidad de ser procesados. Esto permite que se carguen rápidamente en el navegador, mejorando la experiencia del usuario.
- **src.** Carpeta principal del servidor, que agrupa los archivos esenciales para su funcionamiento
  - **components.** Contiene los componentes de la interfaz de usuario, divididos en subcarpetas según su función.
  - **context.** Gestiona el estado global de la aplicación, facilitando la comunicación y el intercambio de datos entre componentes.
  - **pages.** Almacena las diferentes páginas de la aplicación, donde se combinan componentes para formar vistas completas con las que el usuario interactúa.
  - **routes.** Contiene la configuración de las rutas del frontend, permitiendo una navegación fluida entre las distintas páginas.
  - **service.** Carpeta que contiene los servicios utilizados en el frontend para facilitar la comunicación con el backend y realizar distintas operaciones.
  - **styles.** Incluye archivos CSS que definen el estilo de la aplicación.
  - **tests.** Contiene los archivos de las pruebas de validación y formateo de los datos.

## 6. DESARROLLO DE LA SOLUCIÓN

---

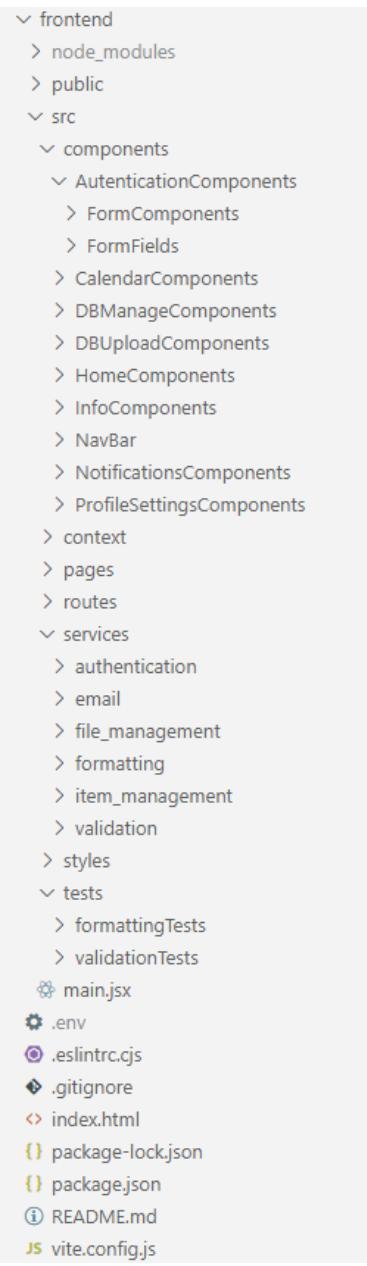


Figura 6.2: Estructura del frontend

- **main.jsx.** Archivo principal de la aplicación donde se inicializa React y se renderiza la interfaz de usuario en la página web.
- **.env.** Archivo que contiene las variables de entorno del frontend, como claves y configuraciones sensibles que no deben incluirse en el código fuente.

- **.eslintrc.** Archivo de configuración de ESLint, que establece las reglas de estilo y buenas prácticas a seguir en el código JavaScript para asegurar su calidad y coherencia.
- **.gitignore.** Lista los archivos y directorios que deben ser excluidos del control de versiones en Git, como las variables de entorno.
- **index.html.** Archivo HTML principal que sirve como punto de entrada para la aplicación React. En este archivo se carga el script main.jsx, que inicializa la aplicación.
- **package.json y package-lock.json.** Archivos que gestionan las dependencias del proyecto. package.json define las bibliotecas y configuraciones, y package-lock.json garantiza la consistencia de sus versiones.
- **vite.config.js.** Archivo de configuración generado automáticamente por Vite al iniciar el proyecto.

### 6.3.2. Enrutamiento

El enrutamiento en el frontend se gestiona utilizando react-router-dom, una biblioteca que permite la navegación entre diferentes páginas de manera sencilla y eficiente. Se han creado dos archivos que gestionan el enrutamiento en la carpeta de routes, como se muestra en la siguiente imagen.



**Figura 6.3:** Estructura del enrutamiento

- **AppRoutes.jsx.** Este archivo configura las rutas generales de la aplicación. Define las rutas públicas, como la de inicio de sesión y registro, y redirige a la página de inicio si el usuario accede a la raíz del sitio. Además, envuelve las rutas protegidas dentro de PrivateRoute, asegurando que solo los usuarios autenticados puedan acceder a ellas (ver fragmento de código 6.12).

Además de las rutas definidas, se ha creado el componente *Navbar* que facilita la navegación entre las distintas páginas de la aplicación. Este componente funciona como una barra de navegación, proporcionando acceso rápido a cada una de las páginas disponibles en la aplicación.

## 6. DESARROLLO DE LA SOLUCIÓN

---

```
1 <Routes>
2     /* Ruta raíz que redirige a la página de inicio de sesión */
3     <Route path="/" element={<Navigate to="/home" />} />
4
5     /* Rutas públicas */
6     <Route path="/home" element={<HomePage />} />
7     <Route path="/login" element={<LoginPage />} />
8     <Route path="/register" element={<RegisterPage />} />
9
10    /* Rutas protegidas por autenticación */
11    <Route path="/" element={<PrivateRoute />}>
12        <Route path="/databaseUpload" element={<DBUploadPage />} />
13        <Route path="/databaseManage" element={<DBManagePage />} />
14        <Route path="/notifications" element={<NotificationsPage />} />
15        <Route path="/calendar" element={<CalendarPage />} />
16        <Route path="/info" element={<InfoPage />} />
17        <Route path="/profileSettings" element={<ProfileSettingsPage />} />
18    </Route>
19 </Routes>
20
21
```

**Fragmento de código 6.12:** Definición de rutas en AppRoutes.jsx

- **PrivateRoute.jsx.** Este otro archivo, verifica el estado de autenticación del usuario utilizando *AuthProvider.jsx*. Si el usuario está autenticado, permite el acceso a las rutas protegidas, de lo contrario, lo redirige a la página de inicio de sesión (ver fragmento de código 6.13).

```
1 const PrivateRoute = () => {
2     // Obtener el estado de autenticación desde el contexto
3     const { auth } = useContext(AuthContext);
4
5     // Si el usuario no está autenticado, se le redirige a /login
6     return auth.isAuthenticated ? <Outlet /> : <Navigate to="/login" />;
7 };
8
9
```

**Fragmento de código 6.13:** Configuración de rutas privadas en PrivateRoute.jsx

### 6.3.3. Seguridad

La seguridad en el frontend se enfoca en garantizar que solo los usuarios autenticados tengan acceso a las funcionalidades internas de la aplicación. Este enfoque sigue la implementación de los sistemas de autenticación y autorización basados en JSON Web Tokens (JWT) y

la clave API en el backend, asegurando que la comunicación entre el frontend y el backend sea segura y controlada.

- **API Key.** La clave API se almacena en el archivo .env de la carpeta frontend. Se envía como cabecera en las solicitudes al backend cuando el usuario aún no ha iniciado sesión, permitiendo al servidor comprobar de dónde provienen estas peticiones (ver fragmento de código 6.14).

```

1  // Obtener la clave API
2  const API_KEY = import.meta.env.VITE_API_KEY;
3
4  // Opciones de la solicitud HTTP
5  const reqOptions = {
6      url: `http://127.0.0.1:5000/users/login}`,
7      method: "POST",
8      headers: {
9          'Content-Type': 'application/json',
10         'X-API-KEY': API_KEY,
11     },
12     data: userData
13 };
14
15 // Realizar la solicitud HTTP utilizando axios
16 const response = await axios.request(reqOptions);
17
18 // Retornar el resultado
19 return response.data;
20

```

**Fragmento de código 6.14:** Ejemplo simplificado de solicitud de login con cabecera API Key

- **JSON Web Token (JWT).** Al iniciar sesión, el frontend recibe el token JWT generado por el backend, y lo almacena en el sessionStorage del usuario. Este token se incluye como cabecera en todas las peticiones que se realicen al backend tras el inicio de sesión.
- **AuthProvider.jsx** Dado que el token JWT tiene una validez de dos horas, al cargar las rutas privadas, el sistema implementado en este archivo verifica si el token está almacenado en el sessionStorage del usuario y si éste sigue siendo válido.

En el fragmento de código 6.15 se muestra cómo se realizan las comprobaciones descritas sobre el token. Como se ha mostrado en el código 6.13, AuthProvider controla el acceso a las rutas privadas, asegurando que solo los usuarios autenticados y con un token válido puedan acceder a ellas.

## 6. DESARROLLO DE LA SOLUCIÓN

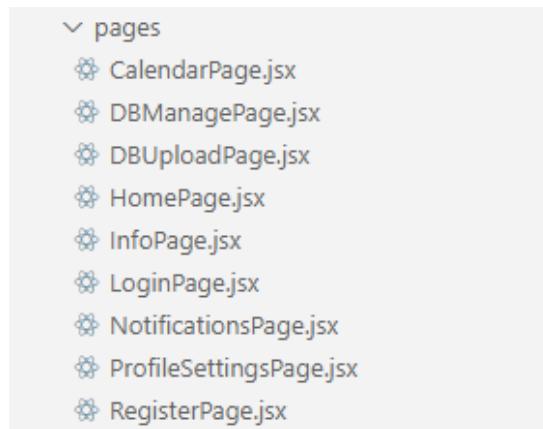
---

```
1 // Cargar el token del almacenamiento local de la sesión
2 const token = sessionStorage.getItem('jwt');
3
4 // Si hay un token, actualizar el estado de autenticación
5 if (token) {
6     setAuth({ token, isAuthenticated: true, loading: false });
7 } else {
8     setAuth({ token: null, isAuthenticated: false, loading: false });
9 }
10
11 // Proveer el estado de autenticación a los componentes hijos
12 return (
13     <AuthContext.Provider value={{ auth, setAuth }}>
14         {children}
15     </AuthContext.Provider>
16 );
17
18
```

**Fragmento de código 6.15:** Verificación y validez del token JWT en AuthProvider.jsx

### 6.3.4. Vistas de la aplicación

Las páginas de la aplicación son componentes que representan las distintas vistas a las que los usuarios pueden navegar. Cada página utiliza componentes específicos para construir una interfaz completa y funcional. Todas las páginas se agrupan en la carpeta *pages*, como se muestra en la figura 6.4.



**Figura 6.4:** Estructura de las páginas

- **HomePage.** Esta página está diseñada para ofrecer una introducción clara y atractiva sobre las ventajas y funcionalidades de la plataforma antes de acceder a ella, tal como se muestra en la figura 6.5.

### 6.3. Desarrollo del frontend

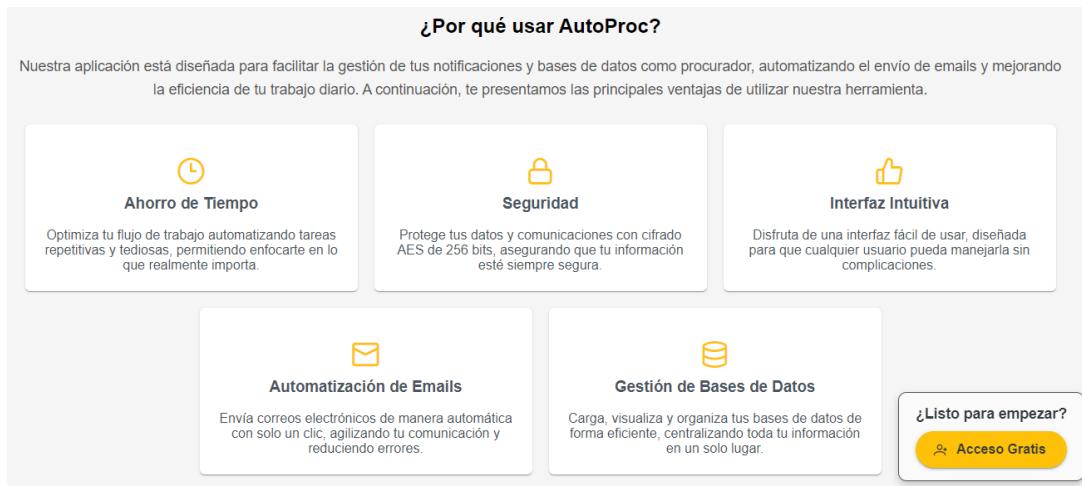


Figura 6.5: Vista de HomePage.jsx

La página comienza con un encabezado que plantea la pregunta "¿Por qué usar AutoProc?", acompañado de un breve texto explicativo que introduce al usuario en los beneficios de la aplicación. A continuación, se presentan los siguientes componentes que completan la página.

- **BenefitsComponent.jsx.** Este componente presenta las principales ventajas de la aplicación de forma visual y organizada, utilizando tarjetas que destacan aspectos clave como la seguridad, la optimización de tiempo, la interfaz intuitiva, la automatización de correos electrónicos y la gestión de bases de datos. Cada tarjeta utiliza íconos y texto para comunicar los beneficios de manera clara y efectiva.
- **FunctionalitiesImagesComponent.jsx.** Proporciona una galería de imágenes que representan las funcionalidades clave de la aplicación. Estas imágenes son un adelanto visual que proporcionan a los usuarios una idea clara de cómo se ven las funcionalidades de la plataforma.
- **FloatingWindowComponent.jsx** Se trata de una ventana flotante fija que aparece en la esquina inferior derecha de la pantalla. Su objetivo es captar la atención del usuario y ofrecer un acceso rápido para entrar en la plataforma. Incluye un botón que redirige al usuario a la página de inicio de sesión.
- **RegisterPage.** Esta página permite a los nuevos usuarios crear una cuenta en la plataforma proporcionando su nombre, apellidos, correo electrónico y una contraseña segura, tal y como se muestra en la figura 6.6. Además, incluye un enlace que redirige a los usuarios ya registrados a la página de inicio de sesión.

## 6. DESARROLLO DE LA SOLUCIÓN

---

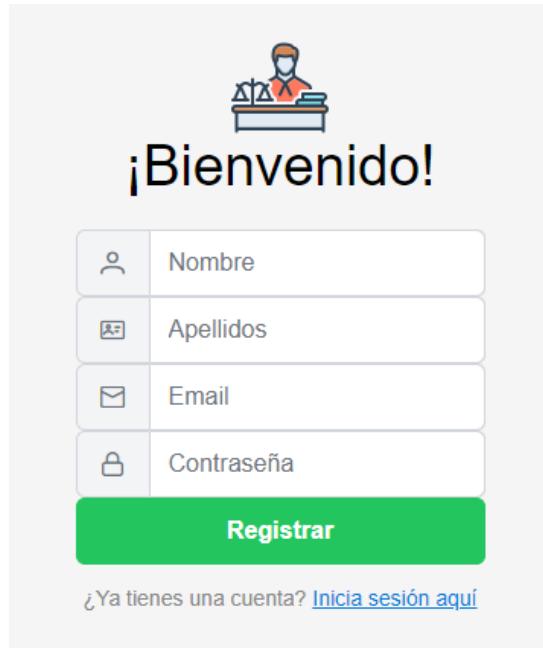


Figura 6.6: Vista de RegisterPage.jsx

La página de registro está compuesta por los siguientes componentes.

- **NameField.jsx y LastNamesField.** Estos campos permiten a los usuarios introducir sus nombres y apellidos, admitiendo nombres compuestos y diferentes acentuaciones. Los componentes validan el formato y muestran posibles mensajes de error en tiempo real.
- **EmailField.jsx.** Este campo permite al usuario ingresar su dirección de correo electrónico, validando en tiempo real que cumpla con un formato adecuado.
- **PasswordField.jsx.** La contraseña debe contener al menos ocho caracteres, una letra mayúscula, una letra minúscula y un número.
- **SignupButton.jsx** Este botón permite al usuario enviar la información proporcionada al backend mediante el servicio *SignupService.jsx*, donde se lleva a cabo el proceso de registro. Si éste es exitoso, el backend genera un token JWT, que se almacena en la sesión del usuario. Al igual que en el inicio de sesión, esta página proporciona retroalimentación clara sobre el estado de la operación.

Finalmente se muestra una retroalimentación clara sobre el estado de la operación, y en caso de fallo, incluirá el motivo del mismo.

- **LoginPage.** Esta página permite a los usuarios registrados iniciar sesión en la plataforma utilizando sus credenciales, como se muestra en la figura 6.7. Además, ofrece una opción clara a los nuevos usuarios para dirigirse al formulario de registro mediante un

enlace directo. La interfaz es sencilla y accesible, con campos de entrada para el correo electrónico y la contraseña, y un botón para enviar la información y autenticarse.

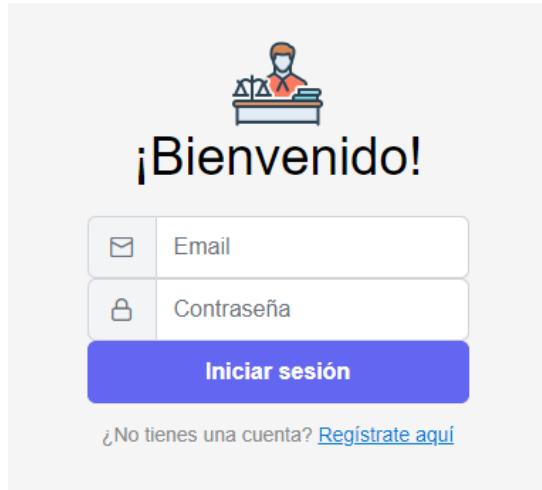


Figura 6.7: Vista de LoginPage.jsx

Esta página sigue una estructura similar a la del registro, compuesta por los siguientes componentes.

- **EmailField.jsx.** Este campo permite a los usuarios introducir su dirección de correo electrónico, validando que el formato sea correcto y mostrando posibles errores de formato en tiempo real.
- **PasswordField.jsx.** Este campo permite a los usuarios introducir su contraseña, la cual no recibe ninguna validación hasta llegar al backend, donde se comprobará con la almacenada.
- **LoginButton.jsx.** Este botón permite enviar la información al backend mediante el servicio *LoginService.jsx*, que se encarga de realizar la solicitud al servidor para autenticar al usuario. Si la autenticación es exitosa, el backend devuelve un token JWT que se almacena en la sesión del usuario.

Finalmente se muestra una retroalimentación clara sobre el estado de la operación, y en caso de fallo, incluirá el motivo del mismo.

- **InfoPage.** Esta página proporciona información clave y recursos útiles para los usuarios, organizados en tres secciones principales: instrucciones para el uso de la aplicación, enlaces de interés para los procuradores y un mapa interactivo de juzgados. Además, cuenta con la barra de navegación que facilita el acceso rápido no solo a estas secciones, sino también a las demás funcionalidades de la aplicación.
- **AnimatedTimeLineComponent.jsx.** Componente que muestra una línea de tiempo animada con las instrucciones que el usuario debe seguir para configurar

## 6. DESARROLLO DE LA SOLUCIÓN

y utilizar la aplicación. Cada paso se muestra en una tarjeta animada que aparece progresivamente a medida que el usuario navega por la página, mejorando la experiencia visual, tal y como se muestra en la figura 6.8.

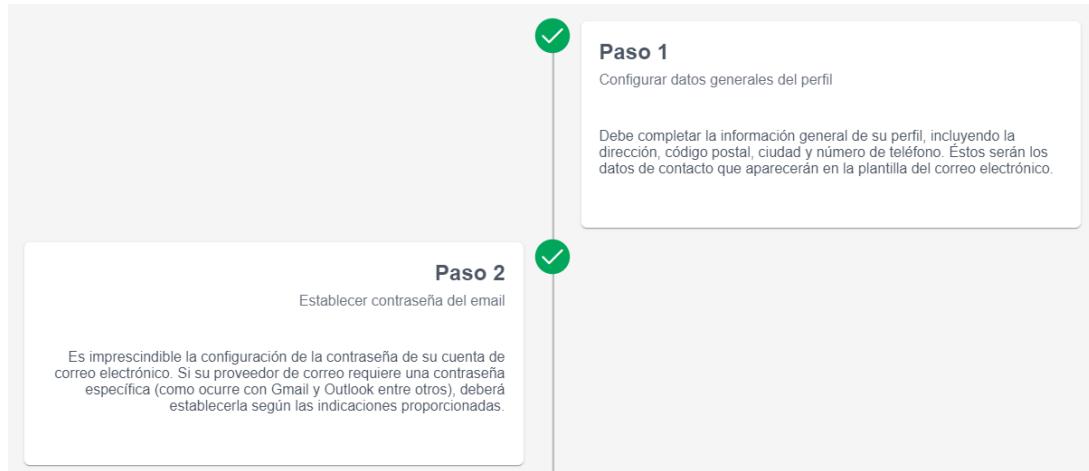


Figura 6.8: Instrucciones iniciales

- **InterestLinksComponent.jsx.** Esta sección presenta una serie de enlaces de interés organizados en tarjetas con información relevante para los procuradores, como se visualiza en la figura 6.9. Cada tarjeta incluye una imagen, una breve descripción y un botón para visitar el sitio web correspondiente.

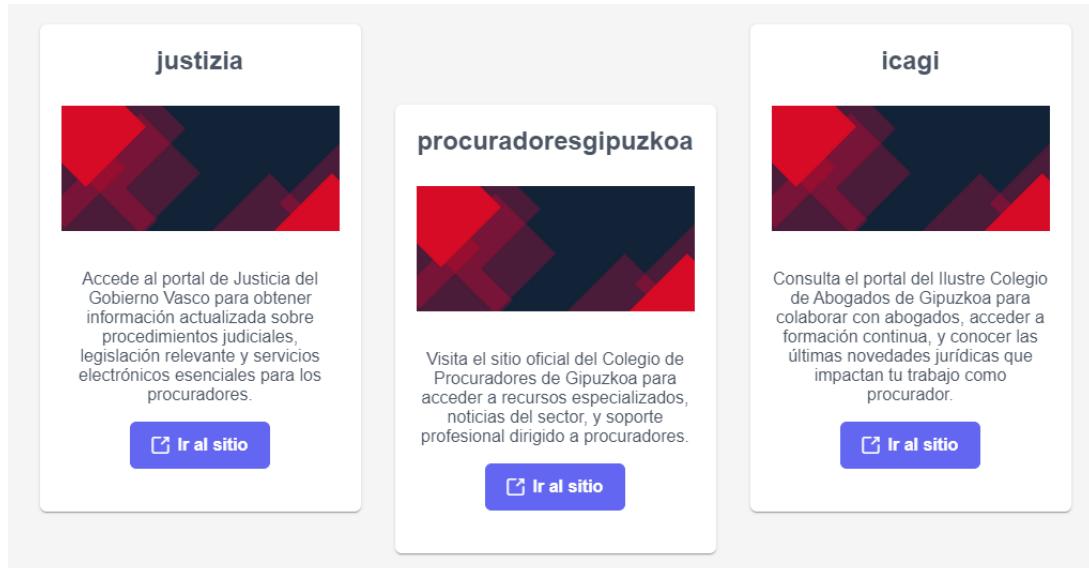


Figura 6.9: Enlaces de interés

### 6.3. Desarrollo del frontend

- **CourtsMapComponent.jsx.** Este componente implementa un mapa interactivo que muestra la ubicación de los principales juzgados del País Vasco, permitiendo a los usuarios explorar la información detallada de cada juzgado, como su nombre y dirección (ver figura 6.10).



Figura 6.10: Mapa de los juzgados

La integración del mapa se ha realizado utilizando la biblioteca @react-google-maps/api, una solución eficiente para incorporar y gestionar Google Maps en aplicaciones construidas con React.

- **ProfileSettingsPage.** Esta página permite a los usuarios visualizar y configurar los datos de su perfil. Ofrece la posibilidad tanto de completar como de actualizar la información personal, como se muestra en las figuras 6.11 y 6.12. Tras el registro, los usuarios deben seguir los pasos indicados en la página InfoPage.jsx, entre los que se detalla que deben completar los datos del perfil, como la contraseña del correo electrónico, el directorio de trabajo local y los datos de contacto y dirección.
  - **ProfileSettingsComponent.jsx.** Este es el componente principal que se encarga de gestionar la vista del perfil del usuario permitiendo tanto visualizar los datos como cambiar al modo edición para completarlos o actualizarlos.
    - **NameField.jsx** y **LastNamesField.jsx.** Campos destinados a recoger el nombre y los apellidos del usuario, asegurando que ambos campos permitan nombres compuestos y diferentes acentuaciones.
    - **EmailField.jsx.** Campo utilizado para recoger el correo electrónico del usuario, validando que siga un formato apropiado.
    - **PhoneField.jsx.** Permite al usuario introducir su número de teléfono.

## 6. DESARROLLO DE LA SOLUCIÓN

---

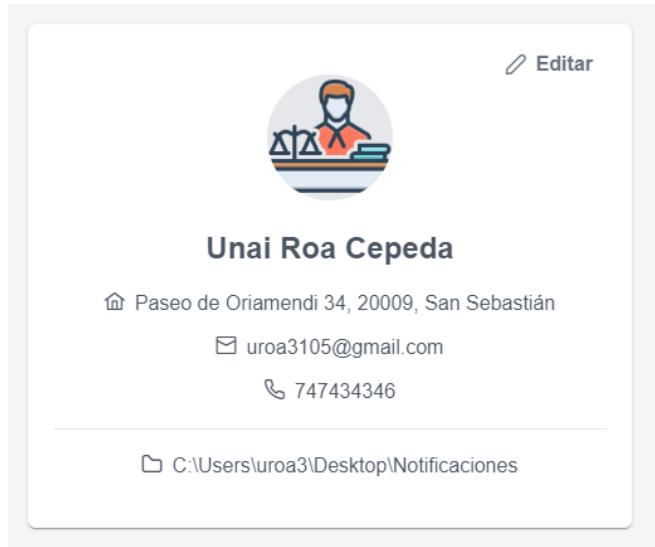


Figura 6.11: Configuración del perfil completada

- **AddressField.jsx, PostalCodeField.jsx, y CityField.jsx.** Campos destinados a recoger la dirección completa del usuario, incluyendo la dirección, el código postal y la ciudad.
- **LocalPathField.jsx.** Campo que permite al usuario especificar un directorio local de trabajo, donde almacenará los documentos PDFs relacionados con sus notificaciones.

A screenshot of a 'Editar Perfil' (Edit Profile) form. At the top left is a circular profile picture of the same person as in Figure 6.11. To the right is the 'Editar Perfil' button. Below the form are two green checkmarks and one red X button. The form consists of ten input fields, each with an icon and a label:

- User icon: Unai
- Address icon: Roa Cepeda
- Email icon: uroa3105@gmail.com
- Lock icon: Contraseña del email
- Phone icon: Teléfono
- House icon: Dirección
- Location pin icon: Código Postal
- Globe icon: Ciudad
- Folder icon: C:\Users\user\Desktop\exampleFolder

Figura 6.12: Actualización de los datos del perfil

- **EmailPasswordSetupDialog.jsx.** Este componente muestra un diálogo emergente (ver figura 6.13) que guía al usuario en la configuración de una contraseña específica para aplicaciones de terceros (como lo es AutoProc). Esta contraseña es esencial para el envío de correos a través de la plataforma. Las instrucciones son personalizadas según el proveedor de correo del usuario, ya que cada uno (como Gmail y Outlook) tiene pasos ligeramente diferentes. Además, es importante destacar que esta nueva contraseña generada no es la misma que el usuario utiliza para acceder a su cuenta de correo.



Figura 6.13: Diálogo de configuración de contraseña del correo

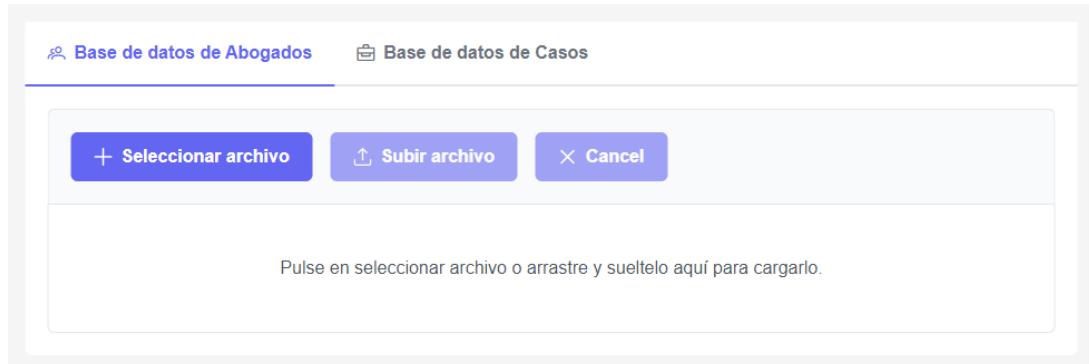
Los componentes mencionados utilizan los siguientes servicios para intercambiar los datos con el backend de forma segura, permitiendo visualizar la información actualizada del usuario en todo momento y crear la estructura de carpetas para la gestión de notificaciones.

- **GetUserService.jsx.** Servicio que se encarga de obtener los datos del perfil del usuario al cargar la página, enviando una solicitud al backend para recuperar la información almacenada.
- **UpdateUserService.jsx.** Sistema que envía al backend los datos del perfil del usuario que han sido previamente validados y formateados, permitiendo la actualización de la información almacenada
- **CreateFoldersService.jsx.** Servicio que se encarga de crear las carpetas en el directorio de trabajo local especificado por el usuario, asegurando una correcta organización de las notificaciones gestionadas por la plataforma.
- **DBUploadPage.** Esta página permite al usuario cargar las bases de datos que tuviera sobre los casos y abogados, como se ve en la figura 6.14. La interfaz está estructurada

## 6. DESARROLLO DE LA SOLUCIÓN

---

en dos pestañas que separan cada tipo de base de datos (la de casos y la de abogados), facilitando la carga de información específica de manera sencilla e intuitiva.



**Figura 6.14:** Vista de carga de bases de datos

- **FileUploadComponent.jsx.** Este componente permite al usuario seleccionar y cargar un archivo en formato `.json` o `.xlsx` para cargar la base de datos. Se reutiliza en ambas pestañas, adaptándose dinámicamente al tipo de base de datos que se esté gestionando, ya sea abogados o casos.
- **FileInfoComponent.jsx.** Este componente muestra la información del archivo seleccionado, incluyendo el nombre, tamaño y estado del archivo, además de ofrecer la opción de eliminarlo antes de proceder con el envío definitivo al servidor.
- **DataTableComponent.jsx.** Una vez seleccionado el archivo que se desea cargar, al mismo tiempo que `FileInfoComponent.jsx` muestra información relevante sobre el archivo, este componente presenta una previsualización en formato de tabla del contenido del archivo cargado, como se muestra en la figura 6.15. Esto permite al usuario revisar los datos que se van a cargar antes de enviarlos al backend.

Para gestionar y almacenar los datos de manera segura y eficiente, los componentes hacen uso de los siguientes servicios, los cuales procesan, validan y envían la información al backend.

- **FileProcessingService.jsx.** Este servicio se encarga de procesar el archivo seleccionado por el usuario y devolver un objeto JSON con los datos validados y formateados.

Cuando se trata de un archivo `.json`, el proceso es más sencillo, realizando únicamente la validación y formateo de los encabezados y datos para que cumplan con los formatos requeridos y se almacenen de manera coherente.

En el caso de tratarse de un archivo `.xlsx` (Excel), se utiliza la biblioteca `exceljs` para leer y transformar los datos a una estructura JSON, pudiendo aplicar nuevamente el proceso de validación y formateo de la información.

### 6.3. Desarrollo del frontend

The screenshot shows a web application interface. At the top, there are two tabs: 'Base de datos de Abogados' and 'Base de datos de Casos'. Below the tabs, there is a file upload section with three buttons: '+ Seleccionar archivo', 'Subir archivo', and 'Cancel'. A file named 'cases.json' is listed, showing its size as '0.62 KB' and status as 'Pendiente' (Pending). To the right of the file name is a red 'X' button. Below this, there is a data table with columns: 'cliente ↑↓', 'expediente ↑↓', 'letrado ↑↓', 'dado en fecha ↑↓', 'pago ↑↓', and 'nig ↑↓'. The table contains four rows of data. At the bottom of the table is a navigation bar with icons for '««', '<', '1', '>', and '»»'.

cliente ↑↓	expediente ↑↓	letrado ↑↓	dado en fecha ↑↓	pago ↑↓	nig ↑↓
Ana Rodríguez	CIVIL 4527/19	Lara Martínez	15/03/2021	completado	2006943220220006153
Luis Fernández	PENAL 1024/15	Jorge Pérez	10/07/2020	pendiente	2006943220210011185
María López	LABOR 3312/21	Raúl Sánchez	08/12/2022	completado	2006943220220008328

Figura 6.15: Previsualización tras la carga de las bases de datos

- **FileUploadService.jsx.** Este servicio se encarga de enviar al backend los datos procesados y validados, garantizando que éstos se almacenen correctamente en la base de datos privada correspondiente al usuario.
- **DBManagePage.** Esta página permite al usuario gestionar las bases de datos de casos y abogados cargadas previamente. La interfaz sigue la misma estructura que *DBUploadPage.jsx*, mostrando dos pestañas que separan cada tipo de base de datos, como se observa en la figura 6.16. En ellas, se visualizan los datos cargados y se ofrece la posibilidad de ordenar, editar y eliminar los registros de manera intuitiva.

A continuación se muestra el componente principal utilizado para la visualización y gestión de los datos.

- **EditableDataTableComponent.jsx.** Este componente muestra los datos de la pestaña seleccionada en una tabla editable, permitiendo al usuario visualizar todos los casos o abogados cargados previamente y ordenarlos según sus atributos. Además, ofrece la opción de seleccionar un elemento para modificar sus valores directamente en la tabla o eliminar el registro completo.

Para realizar las funciones que se han detallado en *EditableDataTableComponent.jsx* es imprescindible el uso de los siguientes servicios.

- **GetAllItemsService.jsx.** Se encarga de obtener los datos de la base de datos del backend, enviando una solicitud para recuperar todos los registros almacenados.

## 6. DESARROLLO DE LA SOLUCIÓN

Base de datos de Abogados		Base de datos de Casos	
	email ↑↓		name ↑↓
	delete edit jperez@gmail.com		name phone ↑↓
	delete edit ✓ × laumartinez@gmail.com	Laura Martínez	732671823
	delete edit rauls@gmail.com	Raúl Sánchez	627132183

Figura 6.16: Visualización de las bases de datos

- **ItemUpdateService.jsx.** Este servicio envía al servidor las actualizaciones de los datos cuando se edita un registro en la tabla. Envía la información validada y formateada al backend para mantener la coherencia de los datos almacenados.
- **ItemDeleteService.jsx.** Mediante esta función se establece comunicación con el backend, enviando el identificador único (`_id`) del registro junto con el nombre de la base de datos correspondiente. Una vez recibida la solicitud, el servidor localiza y elimina de manera segura el registro especificado.
- **ValidateItemService.jsx.** Este servicio realiza la validación de los datos de los registros antes de proceder con la actualización de los mismos. Se asegura de que los valores cumplan con los formatos establecidos.
- **NotificationsPage.** Esta página permite la gestión y el envío de los documentos PDF asociados a las notificaciones judiciales del procurador. La interfaz se estructura en dos pestañas, una para visualizar las notificaciones recibidas y otra para las enviadas, como se muestra en la figura 6.17. Estas pestañas reflejan la estructura de carpetas creada en el directorio local de trabajo especificado por el usuario, por lo que, por lo que los documentos mostrados corresponden con los archivos presentes en dichas carpetas locales.

Además, ofrece un sistema de envío de correos electrónicos que permite al usuario elegir entre el método automático o manual. Este mecanismo genera los correos utilizando una plantilla predefinida, realiza los envíos seleccionados y, tras completarlos, actualiza los directorios moviendo los documentos enviados a la carpeta correspondiente.
- **LocalFolderViewerComponent.jsx.** Este componente muestra el contenido de las carpetas locales de notificaciones recibidas y enviadas. Permite visualizar los

### 6.3. Desarrollo del frontend

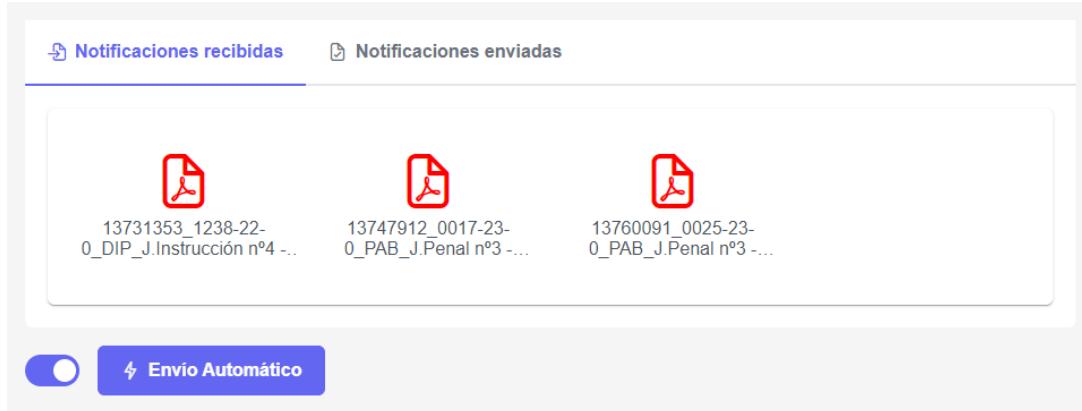


Figura 6.17: Vista de notificaciones recibidas

archivos PDF guardados en estas carpetas, con una actualización automática cada 5 segundos para reflejar los cambios más recientes.

- **Botón de envío.** Permite al usuario elegir entre realizar el envío de los correos electrónicos automático o manual. Al pulsarlo, se despliega un diálogo según la opción seleccionada, proporcionando las herramientas necesarias para completar el proceso.

A continuación se describen los distintos métodos de envío de los correos electrónicos.

- **Envío automático.** Esta opción permite al usuario la posibilidad de generar tantos correos electrónicos como notificaciones haya recibido, facilitando el envío simultáneo con solo unos clics. Al seleccionar este método, se genera automáticamente un correo electrónico para cada notificación, utilizando una plantilla predefinida (ver figura 6.18).

Antes de proceder con el envío, el usuario tiene la opción de previsualizar cada uno de los correos generados, a través de los botones situados al lado de cada notificación, como se observa en la figura 6.19. Gracias a este sistema, el usuario puede revisar lo que cada destinatario va a recibir.

- **Envío manual.** Este método proporciona al usuario un mayor control sobre el contenido de los correos electrónicos antes de enviarlos. A diferencia del envío automático, en esta opción el usuario selecciona una notificación específica para generar el correo correspondiente, como se muestra en la figura 6.20.

En ese momento, éste se crea utilizando la plantilla predefinida, pero con la posibilidad de ser editado por el usuario. Esto permite ajustar detalles del correo proporcionando una mayor flexibilidad, como se observa en la figura 6.21.

Todos los componentes y procesos mencionados funcionan gracias a la implementación de los principales servicios que se detallan a continuación.

## 6. DESARROLLO DE LA SOLUCIÓN

---



Figura 6.18: Envío automático de correos electrónicos

- **GetUserService.jsx.** Se encarga de obtener los datos del usuario a través de realizar una petición al backend. Estos datos serán utilizados en la plantilla predefinida para la elaboración de los correos electrónicos.
- **GetFileListFromLocalService.jsx.** Habiendo obtenido el directorio local de trabajo del usuario mediante *GetUserService.jsx*, se solicita nuevamente al backend la lista de archivos que se encuentran en cada carpeta de este directorio. Se utiliza para mostrar los archivos de notificaciones recibidas y enviadas en el componente *LocalFolderViewerComponent.jsx*.
- **GetNIGFromFileService.jsx.** Este servicio envía al backend la ruta de una notificación para que se realice la extracción del Número de Identificación General (NIG). Este identificador es fundamental para localizar el caso judicial asociado a la notificación, lo que permite incluir los datos correspondientes en el correo electrónico generado posteriormente.
- **GetCaseByNIGService.jsx.** Este servicio solicita al servidor los datos de un caso judicial a partir del Número de Identificación General (NIG) obtenido mediante *GetNIGFromFileService.jsx*.
- **GetLawyerByNameService.jsx.** Se encarga de solicitar al backend los datos de un abogado dado su nombre. Permite obtener la información del abogado correspondiente al caso para completar el correo electrónico.
- **SendEmailService.jsx.** Este servicio solicita al backend el envío de un correo electrónico, proporcionando toda la información necesaria, como el remitente, el destinatario, el asunto, el cuerpo del mensaje y la ruta del archivo adjunto.
- **MoveFileService.jsx.** Una vez enviado el correo electrónico, se utiliza este servicio para solicitar al backend mover la notificación enviada desde su ubicación actual a la carpeta de notificaciones enviadas.

### 6.3. Desarrollo del frontend



Figura 6.19: Previsualización de un correo automático



Figura 6.20: Envío manual de correos electrónicos

- **CalendarPage.** Ésta página permite a los usuarios visualizar un calendario donde los días con un juicio programados se destacan visualmente. Al seleccionar una fecha con un evento previsto, se muestra información detallada sobre éste, incluyendo la hora y la ubicación (ver figura 6.22).

A continuación se describe el componente principal empleado.

- **CustomCalendarComponent.jsx.** Este componente se encarga de mostrar el ca-

## 6. DESARROLLO DE LA SOLUCIÓN



**Figura 6.21:** Visualización de un correo manual

lendario y los eventos de manera interactiva. Para cargar los juicios correctamente, *CalendarPage.jsx* realiza una solicitud al backend para obtener los datos del usuario, incluyendo sus juicios programados. El componente recibe esta información y la proyecta en el calendario de forma dinámica.

### 6.3.5. Validación y formateo de datos

En este apartado se describen los servicios encargados de asegurar la corrección y coherencia de todos los datos proporcionados por el usuario, así como los registros de los casos y abogados o la información personal recogida en los formularios. Estos mecanismos son fundamentales para enviar al backend información válida y con un formato preciso, evitando posibles errores y garantizando la consistencia en los procesos de almacenamiento y envío de datos.

#### ■ Servicios de validación de datos

- **ValidateNameService.jsx.** Valida los nombres proporcionados mediante una expresión regular que permite una o más palabras con solo caracteres alfabéticos y ciertos caracteres especiales como apóstrofes y guiones.



**Figura 6.22:** Vista del calendario con un evento seleccionado

- **ValidatePhoneService.jsx.** Verifica los números de teléfono utilizando un patrón que acepta números españoles, incluyendo opcionalmente el prefijo +34. Asegura que los números tengan una longitud de 9 dígitos y comiencen con un dígito correspondiente a los rangos válidos para móviles (6 o 7) y para teléfonos fijos (8 o 9).
- **ValidateEmailService.jsx.** Comprueba los correos electrónicos utilizando una expresión regular que asegura que cumplan con una estructura adecuada. Ésta permite una combinación de letras, números y algunos caracteres especiales como puntos, guiones o guiones bajos antes del símbolo @. Seguido de esto, el correo debe contener un dominio de correo electrónico válido.
- **ValidateDateService.jsx.** Se encarga de validar fechas permitiendo múltiples formatos predefinidos como dd/MM/yyyy y yyyy-MM-dd. Analiza la fecha en cada formato mediante la biblioteca *date-fns* y lo transforma al siguiente formato, dd/MM/yyyy.
- **ValidateNIGService.jsx.** Valida el Número de Identificación General (NIG) mediante una expresión regular, asegurándose de que contenga 19 dígitos consecutivos sin caracteres adicionales.
- **ValidateExpedientService.jsx.** Verifica que la referencia del expediente cumpla con un patrón específico. Éste incluye una combinación de letras seguidas de un espacio, y terminado con otra combinación de números y una barra.
- **ValidatePayService.jsx.** Valida el estado de pago de los casos, asegurándose de que los valores permitidos sean *pagado*, *completado* o *pendiente*. Cualquier valor

## 6. DESARROLLO DE LA SOLUCIÓN

---

fueras de esta lista no será válido.

- **ValidateHeadersService.jsx.** Verifica que los encabezados de los datos proporcionados al cargar las bases de datos coincidan con los esperados para cada tipo de base de datos. Utiliza una lista de encabezados de referencia y compara estos con los del documento proporcionado, garantizando que no falten campos esenciales. Además, los campos no relevantes se omiten, evitando añadir información innecesaria.
- **ValidateCaseDataService.jsx.** Comprueba que todos los campos de un caso específico estén presentes utilizando un listado predefinido de datos requeridos. Además, cada campo es validado y formateado de manera individual mediante los demás servicios de validación y formateo descritos.
- **ValidateLawyerDataService.jsx.** Al igual que *ValidateCaseDataService.jsx*, este servicio valida los datos de un abogado asegurando que todos los campos requeridos estén presentes y tengan el formato correcto. Utiliza una lista predefinida de los datos necesarios y valida y formatea individualmente cada campo mediante los servicios descritos en este apartado.
- **ValidateItemService.jsx.** Este servicio determina si el registro a validar corresponde con un caso judicial o con un abogado. En función de que tipo de registro se trate, llama al servicio *ValidateCaseDataService.jsx* o a *ValidateLawyerDataService.jsx*.

### ■ Servicios de formateo de datos

- **FormatNameService.jsx.** Formatea los nombres eliminando espacios innecesarios y asegurando que cada palabra comience con una letra mayúscula.
- **FormatPhoneService.jsx.** Elimina el prefijo "-34" si está presente y quita todos los espacios que puedan haber entre los dígitos. Este formateo asegura que los números se guarden de manera uniforme en la base de datos.
- **FormatHeadersService.jsx.** Se encarga de normalizar los encabezados de los documentos proporcionados, asegurando que coincidan con una lista de encabezados esperados.

### 6.3.6. Manejo de errores y retroalimentación al usuario

El sistema implementa un enfoque integral para el manejo de errores y la retroalimentación al usuario, proporcionando una experiencia clara y detallada de las interacciones del usuario con la plataforma.

- **Sistema de retroalimentación.** La aplicación utiliza *ToastProvider.jsx* para mostrar notificaciones emergentes tras las interacciones del usuario con la plataforma. Éstas aparecen en la esquina superior derecha de la pantalla y proporcionan distintos tipos de mensajes:

- **Notificaciones simples.** Informan sobre éxitos, advertencias o errores mediante un breve mensaje, proporcionando la información esencial de manera concisa, como se muestra en la figura 6.23.

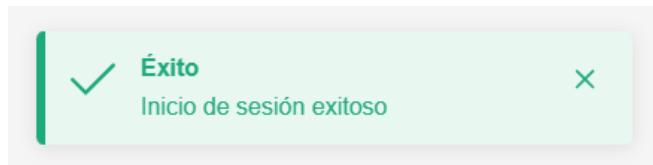


Figura 6.23: Ejemplo de retroalimentación de éxito

- **Notificaciones interactivas.** Además del mensaje informativo, incluyen un enlace que permite al usuario realizar una acción adicional, como se observa en la figura 6.24. En el caso de las advertencias sobre configuraciones incompletas, el enlace redirige al apartado correspondiente para que el usuario pueda completarlo.

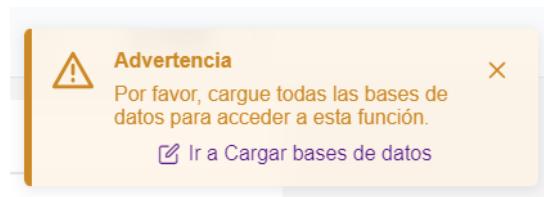


Figura 6.24: Ejemplo retroalimentación de advertencia con enlace

Por otro lado, cuando se trata de errores complejos, como fallos al cargar una base de datos, el enlace abre un diálogo emergente donde se muestran los errores detallados (ver figura 6.25).

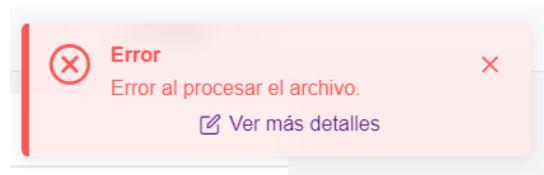


Figura 6.25: Ejemplo de retroalimentación de error con enlace

- **Retroalimentación en tiempo real en formularios.** Además de las notificaciones emergentes, la aplicación proporciona retroalimentación en tiempo real dentro de los formularios, ayudando al usuario a corregir cualquier error mientras completa cada campo.

Cuando se detecta un error en un campo, se marca visualmente y se muestra un mensaje de error junto al mismo. Al pasar el cursor sobre este mensaje, se despliega una breve descripción que indica cuál es el formato correcto del campo afectado, como se muestra en la figura 6.26.

## 6. DESARROLLO DE LA SOLUCIÓN



Figura 6.26: Retroalimentación del formulario de registro

Además, en el campo de la contraseña se incluye un indicador visual que muestra la fuerza de la contraseña, que va aumentando a medida que se cumplen los requisitos establecidos.

- **Manejo de errores.** Todos los componentes de la aplicación contemplan posibles errores y se han diseñado para gestionarlos de manera adecuada. Los errores que ocurren durante las operaciones se manejan mediante excepciones y se muestran al usuario mediante las notificaciones de *ToastProvider.jsx*. En el caso de errores complejos, la notificación incluye un enlace que abre un diálogo emergente donde se detallan los errores capturados por el componente o servicio correspondiente durante el procesamiento de los datos (ver figura 6.27). De esta manera, el usuario recibe información clara y específica sobre lo ocurrido.

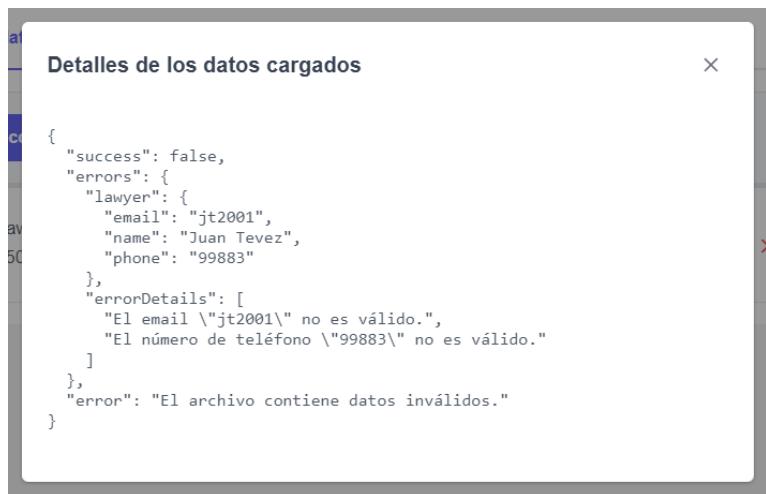


Figura 6.27: Diálogo con detalles de los errores

# CAPÍTULO 7

## Pruebas del sistema

En este capítulo se describen las pruebas realizadas y se muestran sus resultados para asegurar el correcto funcionamiento del sistema en sus distintas capas.

### 7.1. Pruebas del backend

Las pruebas del backend son fundamentales para verificar el correcto funcionamiento del mismo, y se han sido diseñadas para evaluar distintos aspectos clave del sistema, como la seguridad, la autenticación y las operaciones CRUD de las entidades.

#### 7.1.1. Pruebas mediante ThunderClient

En esta sección se detalla el proceso de pruebas de las funcionalidades del backend utilizando la extensión de *Visual Studio Code* llamada *ThunderClient*. Esta herramienta permite simular el envío de peticiones HTTP y verificar las respuestas del servidor de manera eficiente.

##### ▪ Pruebas de seguridad

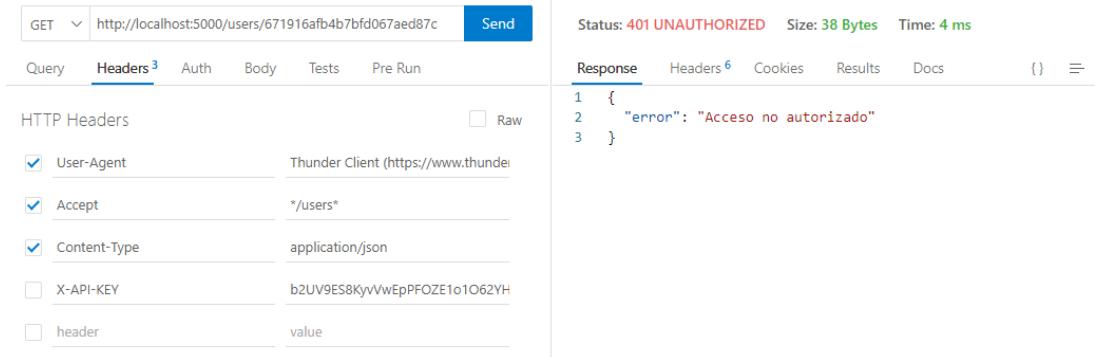
- **Pruebas de autorización mediante API Key.** Mediante estas pruebas se quiere verificar que el acceso a los servicios del backend esté protegido mediante una clave API, la cual debe ser proporcionada en las cabeceras de la petición.

En la primera prueba, se ha realizado una petición al backend para obtener los datos del usuario con `_id: 671916afb4b7bfd067aed87c`. Sin embargo, no se ha establecido la cabecera con la clave API, por lo que se espera recibir un error de autorización (ver figura 7.1).

En la segunda prueba, se ha realizado la misma petición, pero esta vez se ha añadido la cabecera con la clave API. Como resultado, se esperan recibir los datos del usuario correspondiente, como se muestra en la figura 7.2.

## 7. PRUEBAS DEL SISTEMA

---



The screenshot shows a Thunder Client interface with a 'GET' request to `http://localhost:5000/users/671916afb4b7bfd067aed87c`. The 'Headers' tab is selected, displaying the following headers:

- `User-Agent`: Thunder Client (https://www.thunderclient.com)
- `Accept`: \*/users\*
- `Content-Type`: application/json
- `X-API-KEY`: b2UV9ES8KyvWwEpPFOZE1o1O62YH
- `header`: value

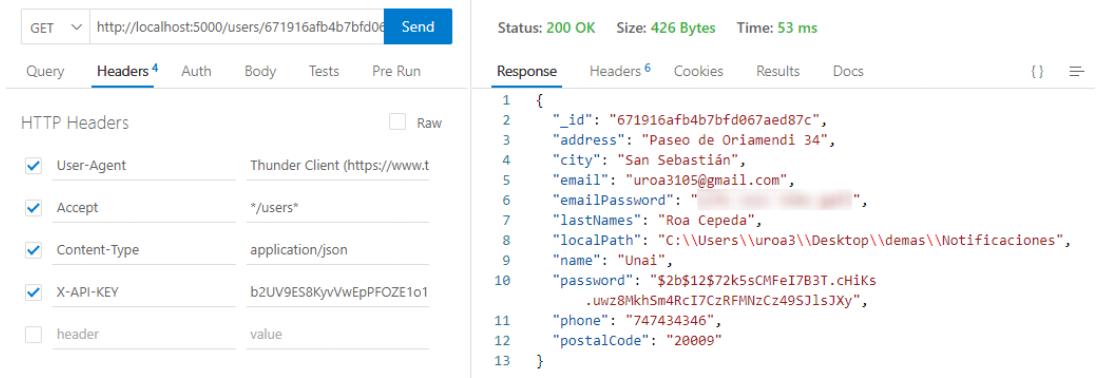
The response status is **401 UNAUTHORIZED**, size **38 Bytes**, and time **4 ms**. The response body is:

```

1  {
2    "error": "Acceso no autorizado"
3  }

```

Figura 7.1: Prueba de autorización mediante API Key - Error controlado



The screenshot shows a Thunder Client interface with a 'GET' request to `http://localhost:5000/users/671916afb4b7bfd067aed87c`. The 'Headers' tab is selected, displaying the same set of headers as in Figure 7.1.

The response status is **200 OK**, size **426 Bytes**, and time **53 ms**. The response body is a JSON object representing a user:

```

1  {
2    "_id": "671916afb4b7bfd067aed87c",
3    "address": "Paseo de Oriamendi 34",
4    "city": "San Sebastián",
5    "email": "uroa3105@gmail.com",
6    "emailPassword": "██████████",
7    "lastNames": "Roa Cepeda",
8    "localPath": "C:\\Users\\uroa3\\Desktop\\demas\\Notificaciones",
9    "name": "Unai",
10   "password": "$2b$12$72k5sCMFeI7B3T.cHiKs
11     .uwz8lkh5m4RcI7czRFMNzCz49SJlsJXy",
12   "phone": "747434346",
13   "postalCode": "20009"
14  }

```

Figura 7.2: Prueba de autorización mediante API Key - Éxito

- **Pruebas de autenticación mediante JWT.** Estas pruebas tienen como objetivo asegurar que las rutas protegidas del backend solo sean accesibles para usuarios autenticados mediante tokens JWT.

En la primera prueba, se ha realizado una petición al backend para obtener los datos del abogado con `_id: 671a8abad72b35c604dec0d5`. Sin embargo, no se ha establecido la cabecera con el token JWT, por lo que se espera recibir un error de autenticación, como se visualiza en la figura 7.3.

En la segunda prueba, se ha realizado la misma petición, pero esta vez se ha añadido la cabecera con el token JWT. Como resultado, se esperan recibir los datos del abogado correspondiente, como se muestra en la figura 7.4.

### ■ Pruebas de autenticación

- **Prueba de registro de nuevos usuarios.** A través de esta prueba se quiere comprobar que el sistema registre correctamente a un usuario nuevo, además de crear sus bases de datos privadas correspondientes y generar su token JWT para la sesión actual.

## 7.1. Pruebas del backend

The screenshot shows a GET request to `http://localhost:5000/lawyers/671a8abad72b35c604dec0d5`. The Headers tab is selected, showing the following configuration:

- User-Agent**: Thunder Client (https://www.thunderclient.com)
- Accept**: \*/lawyers\*
- Content-Type**: application/json
- Authorization**: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6Ij01NzE1MDMzOTkifQ.
- header**: value

The Response tab shows the following JSON output:

```

1 {
2   "msg": "Missing Authorization Header"
3 }

```

Figura 7.3: Prueba de autenticación mediante JWT - Error controlado

The screenshot shows a GET request to `http://localhost:5000/lawyers/671a8abad72b35c604dec0d5`. The Headers tab is selected, showing the following configuration:

- User-Agent**: Thunder Client (https://www.thunderclient.com)
- Accept**: \*/lawyers\*
- Content-Type**: application/json
- Authorization**: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6Ij01NzE1MDMzOTkifQ.
- header**: value

The Response tab shows the following JSON output:

```

1 {
2   "_id": "671a8abad72b35c604dec0d5",
3   "email": "miklorres@gmail.com",
4   "name": "Jorge Pérez",
5   "phone": "621472818"
6 }

```

Figura 7.4: Prueba de autenticación mediante JWT - Éxito

Para llevar a cabo esta prueba, se envía una petición POST al backend con unos datos de prueba válidos, puesto que el frontend es quien se encarga de verificarlos, como se muestra en la figura 7.5.

The screenshot shows a POST request to `http://localhost:5000/users/`. The Body tab is selected, showing the following JSON content:

```

1 {
2   "name": "Juan",
3   "lastNames": "Pérez Gómez",
4   "email": "juan.example@gmail.com",
5   "password": "Password123"
6 }

```

The Response tab shows the following JSON output:

```

1 {
2   "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJcmVmZacI6ZmFsc2UsImhdI6MTcyOTg1NzkzOCwianRpIjoiMDVkhjIzZtkEyjQ2ZS00M2QxLTg1NWeTNRKzDB1NWE1ZDE1IiwidH1wZSI6ImfjY2VzcycI5in1YiI6ijY3MjI40TkxDNg1NmVlZWlZTFKYzMwOSIsIm5iI6MTcyOTg1NzkzOCwiY3NyZii6ImFjYjQ1ODU1LT1hZTUthNDvhzs11NDAyLTRhYTQzMmM0OGY3ZiIsImV4ccI6MTcyOTg2NTEzOHO.eyJDUiafZ_sW4eSeUJFZdAzbQghf6ral1875_ArOQ2k",
3   "message": "Usuario 671b89914856eeeeee1dc309 creado correctamente"
4 }

```

Figura 7.5: Prueba de registro de usuario

Además de verificar que el usuario ha sido registrado exitosamente y se ha generado un token JWT, es fundamental comprobar que las bases de datos privadas

## 7. PRUEBAS DEL SISTEMA

del nuevo usuario han sido creadas correctamente. A continuación, se muestra una captura de pantalla de MongoDB Atlas, donde se observan las bases de datos privadas del nuevo usuario creadas (ver figura 7.6).

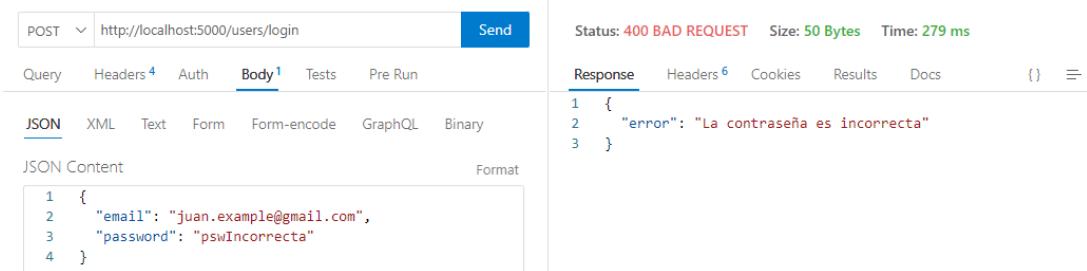


```
user_671b89914856eeeeee1...
cases
lawyers
```

Figura 7.6: Bases de datos privadas creadas al registrar usuario

- **Pruebas de inicio de sesión.** Esta prueba tiene como objetivo verificar que el sistema realice correctamente el proceso de inicio de sesión del usuario.

En la primera prueba, se ha realizado una petición POST con las credenciales del usuario incorrectas, concretamente una contraseña errónea. Por lo tanto, se espera recibir un error controlado con un mensaje indicando que la contraseña es incorrecta, como se observa en la figura 7.7.

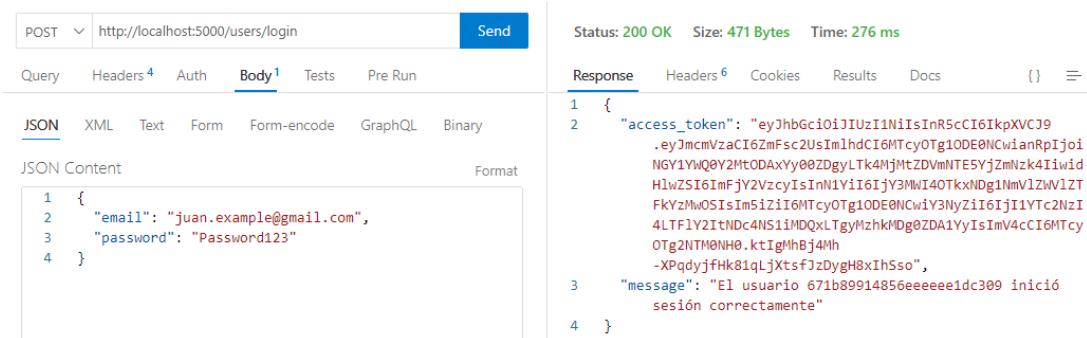


```
Status: 400 BAD REQUEST Size: 50 Bytes Time: 279 ms
Response Headers 6 Cookies Results Docs { } ≡
1 {
2   "error": "La contraseña es incorrecta"
3 }
```

```
POST http://localhost:5000/users/login Send
Query Headers 4 Auth Body 1 Tests Pre Run
JSON XML Text Form Form-encode GraphQL Binary
JSON Content Format
1 {
2   "email": "juan.example@gmail.com",
3   "password": "pswIncorrecta"
4 }
```

Figura 7.7: Prueba de inicio de sesión - Error controlado

En la segunda prueba, se ha realizado la misma petición, pero esta con las credenciales correctas. Por lo tanto, se espera recibir un mensaje de éxito junto con el token JWT de la sesión generado (ver figura 7.8).



```
Status: 200 OK Size: 471 Bytes Time: 276 ms
Response Headers 6 Cookies Results Docs { } ≡
1 {
2   "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
3 .eyJncmVzaCI6ZmFsc2UsImlhCI6MTcyOTg1ODE0NCwianRpIjoi
4 NGY1YnQ0Y2ltODAxYy00ZDgyLTk4MjMtZDVmNTESYjZmNzk4Iiwid
5 HlwZSI6ImFjY2VzcycIsInN1Yi6IjY3MWI40TkxNDg1NmVlZWVlZT
6 FKY2MwOSIsIm51Zi6MTcyOTg1ODE0NCwiy3NyZi6IjI1YTc2NzI
7 4LTf1Y2ItNDc4NS1iMDQxLTgyMzhkMDg0ZDA1YyIsImV4CCI6MTcy
8 OTg2NTM0NH0.ktIgMBj4Mh
9 -XPqdyjfHk81qlJxtsfJzDygh8xIhsso",
10 3   "message": "El usuario 671b89914856eeeeee1dc309 inició
11 4   sesión correctamente"
12 }
```

```
POST http://localhost:5000/users/login Send
Query Headers 4 Auth Body 1 Tests Pre Run
JSON XML Text Form Form-encode GraphQL Binary
JSON Content Format
1 {
2   "email": "juan.example@gmail.com",
3   "password": "Password123"
4 }
```

Figura 7.8: Prueba de inicio de sesión - Éxito

## 7.1. Pruebas del backend

- **Pruebas sobre las operaciones CRUD de las entidades.** Estas pruebas son muy similares para cada una de las entidades por lo que se ha tomado la entidad del abogado como ejemplo y se han realizado las pruebas correspondientes a cada operación CRUD del abogado.
  - **Prueba de creación de un abogado.** Se ha enviado una petición POST al backend para crear un nuevo abogado proporcionando los datos necesarios. Se espera recibir un mensaje de éxito tras almacenar el abogado en la base de datos, como se muestra en la figura 7.9.

The screenshot shows a Postman interface with the following details:

- Method: POST
- URL: http://localhost:5000/lawyers/
- Body tab is selected, showing JSON content:

```
1 {  
2   "name": "Ane Etxeberria",  
3   "email": "ane.etxeberria@outlook.com",  
4   "phone": "77777777"  
5 }
```
- Response tab shows the result:

Status: 201 CREATED Size: 73 Bytes Time: 99 ms

```
1 {  
2   "message": "Abogado 671b8b014856eeeeee1dc30a creado  
correctamente"  
3 }
```

Figura 7.9: Prueba de creación de abogado - Éxito

- **Pruebas de lectura de un abogado.** En primer lugar, se ha realizado una prueba enviando una petición GET proporcionando un `_id` erróneo al backend. Se espera recibir un error controlado mediante un mensaje indicando el `_id` incorrecto, como se visualiza en la figura 7.10.

The screenshot shows a Postman interface with the following details:

- Method: GET
- URL: http://localhost:5000/lawyers/8881b8b014856eeacf1dc30a
- Body tab is selected, showing JSON content:

```
1
```
- Response tab shows the result:

Status: 404 NOT FOUND Size: 64 Bytes Time: 44 ms

```
1 {  
2   "error": "Abogado 8881b8b014856eeacf1dc30a no encontrado"  
3 }
```

Figura 7.10: Prueba de lectura de abogado - Error controlado

En esta segunda prueba, se ha realizado la misma solicitud pero proporcionando un `_id` existente, por lo que se espera obtener los datos del abogado asociado (ver figura 7.11).

- **Prueba de actualización de un abogado.** Se ha enviado una petición PUT al servidor indicando el `_id` del abogado que se desea actualizar y los datos actualizados. En este caso, probamos a enviar el correo actualizado, esperando obtener un mensaje de éxito indicando la correcta actualización (ver figura 7.12).

Tal y como está definido el servidor al actualizar una entidad se han de facilitar todos los campos, incluyendo tanto los datos actualizados como los que se van a mantener igual.

## 7. PRUEBAS DEL SISTEMA

The screenshot shows a Postman interface with a GET request to `http://localhost:5000/lawyers/671b8b014856eeeeee1dc30a`. The response status is 200 OK, size is 133 Bytes, and time is 48 ms. The response body is a JSON object:

```

1 {
2   "_id": "671b8b014856eeeeee1dc30a",
3   "email": "ane.etxeberria@outlook.com",
4   "name": "Ane Etxeberria",
5   "phone": "77777777"
6 }

```

**Figura 7.11:** Prueba de lectura de abogado - Éxito

The screenshot shows a Postman interface with a PUT request to `http://localhost:5000/lawyers/671b8b014856eeeeee1dc30a`. The response status is 400 BAD REQUEST, size is 42 Bytes, and time is 5 ms. The response body is a JSON object:

```

1 {
2   "error": "Faltan campos requeridos"
3 }

```

**Figura 7.12:** Prueba de actualización de abogado - Error no previsto

En esta segunda prueba se realiza la misma solicitud, pero además del correo actualizado, se proporcionan los demás campos con los datos que ya tenía el abogado correspondiente (ver figura 7.13).

The screenshot shows a Postman interface with a PUT request to `http://localhost:5000/lawyers/671b8b014856eeeeee1dc30a`. The response status is 200 OK, size is 78 Bytes, and time is 66 ms. The response body is a JSON object:

```

1 {
2   "message": "Abogado 671b8b014856eeeeee1dc30a actualizado correctamente"
3 }

```

**Figura 7.13:** Prueba de actualización de abogado - Éxito

- **Prueba de eliminación de un abogado.** Se ha enviado una petición DELETE para eliminar un abogado dado su `_id`. Si el abogado con dicho identificador existe, se espera obtener un mensaje de éxito, como se muestra en la figura 7.14.

## 7.2. Pruebas del frontend

Las pruebas de validación y de formateo de datos tienen como objetivo verificar que los datos introducidos por el usuario cumplan con los formatos definidos. Para cada servicio de validación y de formateo, se han definido casos de prueba que abarcan tanto datos correctos como incorrectos.

## 7.2. Pruebas del frontend

The screenshot shows a Postman interface with the following details:

- Method: DELETE
- URL: http://localhost:5000/lawyers/671b8b014856eeeeee1dc30a
- Status: 200 OK
- Size: 62 Bytes
- Time: 49 ms
- Response body:

```
1 {  
2   "message": "Abogado 671b8b014856eeeeee1dc30a eliminado"  
3 }
```

Figura 7.14: Prueba de eliminación de abogado - Éxito

Se han diseñado pruebas unitarias para verificar que las funciones de validación y de formateo identifican correctamente errores en los datos, como números de teléfono con longitud incorrecta o correos electrónicos con formato incorrecto.

### 7.2.1. Pruebas de validación de datos

A continuación, se listan los archivos de pruebas de las validaciones implementados, mostrando algún ejemplo de los casos de prueba planteados para cada uno.

- **ValidateEmailService.test.jsx.** Se verifican distintos formatos de correo electrónico, desde los más simples hasta aquellos con dominios y caracteres especiales, asegurando que el servicio identifique correctamente los correos válidos e inválidos (ver figura 7.15).

```
4  describe('ValidateEmailService', () => {  
5    it('debe devolver true para formatos de correo electrónico válidos', () => {  
6      expect(ValidateEmailService('nombre@dominio.com')).toBe(true);  
7      expect(ValidateEmailService('nombre.apellido@dominio.com')).toBe(true);  
8      expect(ValidateEmailService('nombre-apellido@dominio.com')).toBe(true);  
9    });  
10   it('debe devolver false para formatos de correo electrónico incorrectos', () => {  
11     expect(ValidateEmailService('nombre@dominio')).toBe(false); // No termina en . algo  
12     expect(ValidateEmailService('nombre@.com')).toBe(false); // Entre el @ y el . no hay nada  
13     expect(ValidateEmailService('nombre@@dominio.com')).toBe(false); // Tiene dos @  
14   });  
15 }));  
16 };
```

Figura 7.15: Casos de prueba de ValidateEmailService.test.jsx

- **ValidateNIGService.test.jsx.** Se prueban los NIG (Número de Identificación General) asegurando que tengan el formato numérico adecuado y la longitud requerida, como se muestra en la figura 7.16.
- **ValidatePayService.test.jsx.** Prueba las diferentes categorías de estado de pago aceptadas por el sistema, validando que se reconozcan correctamente los estados como “pendiente” o “completado”, como se visualiza en la figura 7.17.
- **ValidatePhoneService.test.jsx.** Verifica que los números de teléfono introducidos sigan el formato establecido, incluyendo la longitud y la combinación de los primeros dígitos, como se muestra en la figura 7.18.

## 7. PRUEBAS DEL SISTEMA

---

```
5  describe('ValidateNIGService', () => {
6    it('debe devolver true para un NIG de 19 dígitos', () => {
7      expect(ValidateNIGService('1234567890123456789')).toBe(true);
8      expect(ValidateNIGService('9876543210123456789')).toBe(true);
9      expect(ValidateNIGService('2006943220220006153')).toBe(true);
10   });
11
12  it('debe devolver false para NIGs incorrectos', () => {
13    expect(ValidateNIGService('123456789012378')).toBe(false); // Longitud menor a 19
14    expect(ValidateNIGService('12345678901234562137890')).toBe(false); // Longitud mayor a 19
15    expect(ValidateNIGService('hsgggdt273jshhdy167')).toBe(false); // Contiene letra
16    expect(ValidateNIGService('12345678 0123456789')).toBe(false); // Contiene espacio
17  });
18});
```

Figura 7.16: Casos de prueba de ValidateNIGService.test.jsx

```
5  describe('ValidatePayService', () => {
6    it('debe devolver true para estados de pago válidos', () => {
7      expect(ValidatePayService('pagado')).toBe(true);
8      expect(ValidatePayService('completado')).toBe(true);
9      expect(ValidatePayService('pendiente')).toBe(true);
10   });
11
12  it('debe devolver false para estados de pago no válidos', () => {
13    expect(ValidatePayService('cancelado')).toBe(false); // Estado no válido
14    expect(ValidatePayService('')).toBe(false); // Cadena vacía
15    expect(ValidatePayService('Pagado')).toBe(false); // Sensible a mayúsculas/minúsculas
16    expect(ValidatePayService('PAGADO')).toBe(false); // Sensible a mayúsculas/minúsculas
17    expect(ValidatePayService(null)).toBe(false); // Valor nulo
18    expect(ValidatePayService(undefined)).toBe(false); // Valor indefinido
19  });
20});
```

Figura 7.17: Casos de prueba de ValidatePayService.test.jsx

```
5  describe('ValidatePayService', () => {
6    it('debe devolver true para estados de pago válidos', () => {
7      expect(ValidatePayService('pagado')).toBe(true);
8      expect(ValidatePayService('completado')).toBe(true);
9      expect(ValidatePayService('pendiente')).toBe(true);
10   });
11
12  it('debe devolver false para estados de pago no válidos', () => {
13    expect(ValidatePayService('cancelado')).toBe(false); // Estado no válido
14    expect(ValidatePayService('')).toBe(false); // Cadena vacía
15    expect(ValidatePayService('Pagado')).toBe(false); // Sensible a mayúsculas/minúsculas
16    expect(ValidatePayService('PAGADO')).toBe(false); // Sensible a mayúsculas/minúsculas
17    expect(ValidatePayService(null)).toBe(false); // Valor nulo
18    expect(ValidatePayService(undefined)).toBe(false); // Valor indefinido
19  });
20});
```

Figura 7.18: Casos de prueba de ValidatePhoneService.test.jsx

- **ValidateNameService.test.jsx.** Este archivo prueba los nombres y apellidos, comprobando que se sigan las reglas de formato definidas, incluyendo caracteres permitidos, como se observa en la figura 7.19.
- **ValidateDateService.test.jsx.** Se prueban diferentes formatos de fecha aceptados, así como fechas con errores comunes como separadores incorrectos o días y meses fuera de rango, asegurando que el servicio maneje correctamente cada caso (ver figura 7.20).

## 7.2. Pruebas del frontend

```
5  describe('ValidateNameService', () => {
6    it('debe devolver true para nombres válidos', () => {
7      expect(ValidateNameService('Juan Pérez')).toBe(true); // Nombre simple con espacio y tilde
8      expect(ValidateNameService('Ana María López-García')).toBe(true); // Nombres compuestos y apellidos
9      expect(ValidateNameService("O'Connor")).toBe(true); // Apóstrofe en nombre
10     expect(ValidateNameService('José Luis')).toBe(true); // Nombre compuesto con tilde
11   });
12
13  it('debe devolver false para nombres inválidos', () => {
14    expect(ValidateNameService('1234')).toBe(false); // Solo números
15    expect(ValidateNameService('Juan_Pérez')).toBe(false); // Guion bajo no permitido
16    expect(ValidateNameService('Juan@Pérez')).toBe(false); // Simbolo no permitido
17    expect(ValidateNameService('')).toBe(false); // Cadena vacía
18    expect(ValidateNameService(' ')).toBe(false); // Solo espacios
19    expect(ValidateNameService('Juan Pérez!!')).toBe(false); // Caracteres especiales no permitidos
20  });
21});
```

Figura 7.19: Casos de pruebas de ValidateNameService.test.jsx

```
4  describe('ValidateDateService', () => {
5    it('debe devolver true y formatear la fecha cuando el formato es dd/MM/yyyy', () => {
6      const result = ValidateDateService('15/03/2021');
7      expect(result).toEqual({ isValid: true, formattedDate: '15/03/2021' });
8    });
9
10   it('debe devolver true y formatear la fecha cuando el formato es dd-MM-yyyy', () => {
11     const result = ValidateDateService('15-03-2021');
12     expect(result).toEqual({ isValid: true, formattedDate: '15/03/2021' });
13   });
14
15   it('debe devolver true y formatear la fecha cuando el formato es yyyy-MM-dd', () => {
16     const result = ValidateDateService('2021-03-15');
17     expect(result).toEqual({ isValid: true, formattedDate: '15/03/2021' });
18   });
19});
```

Figura 7.20: Ejemplos de casos de prueba de ValidateDateService.test.jsx

- **ValidateExpedientService.test.jsx.** Se realizan pruebas sobre el formato de los expedientes, validando tanto las secuencias de caracteres permitidos como el patrón específico utilizado, como se muestra en la figura 7.21.

```
5  describe('ValidateExpedientService', () => {
6    it('debe devolver true para formatos de expediente válidos', () => {
7      expect(ValidateExpedientService('PENAL 1024/15')).toBe(true); // Letras seguidas de números con barra
8      expect(ValidateExpedientService('CIVIL 2048/20')).toBe(true); // Letras y números en otro tipo
9      expect(ValidateExpedientService('MERCA 450/22')).toBe(true); // Ejemplo de expediente con otro nombre
10     expect(ValidateExpedientService('LABOR 9084/18')).toBe(true); // Otro caso
11   });
12
13  it('debe devolver false para formatos de expediente incorrectos', () => {
14    expect(ValidateExpedientService('1024/15')).toBe(false); // Sin categoría al inicio
15    expect(ValidateExpedientService('PENAL 1024/')).toBe(false); // Barra al final sin número de año
16    expect(ValidateExpedientService('PENAL /15')).toBe(false); // Sin número antes de la barra
17    expect(ValidateExpedientService('penal 1024/15')).toBe(false); // Sin mayúsculas
18    expect(ValidateExpedientService('PENAL 1024 15')).toBe(false); // Sin barra entre números
19  });
20});
```

Figura 7.21: Casos de prueba de ValidateExpedientService.test.jsx

- **ValidateHeadersService.test.jsx.** Este archivo se enfoca en la validación de los encabezados requeridos al cargar archivos, comprobando que todos los campos críticos

## 7. PRUEBAS DEL SISTEMA

---

estén presentes (ver figura 7.22).

```
30 | it('debe devolver error cuando faltan encabezados críticos de abogados', () => {
31 |   const headers = ["name", "phone"];
32 |   const result = ValidateHeadersService(headers, expectedLawyersHeaders, expectedCasesHeaders, 'lawyers');
33 |
34 |   expect(result).toEqual({
35 |     success: false,
36 |     error: {
37 |       message: 'Falta algún campo crítico en el documento.',
38 |       missingHeaders: ["email"]
39 |     }
40 |   });
41 | });

Figura 7.22: Ejemplo de caso de prueba de ValidateHeadersService.test.jsx
```

- **ValidateLawyerDataService.test.jsx.** Comprueba los datos relacionados con los abogados, como nombre, correo electrónico y número de teléfono. Se han definido casos de prueba para datos válidos e inválidos, asegurando que se detecten errores de formato (ver figura 7.23).

```
75 | it('debe devolver dos errores cuando dos campos son incorrectos', () => {
76 |   const lawyerItem = {
77 |     name: '123 Ana',
78 |     email: 'ana.lopez@correo',
79 |     phone: '+34 612 345 678'
80 |   };
81 |   const result = ValidateLawyerDataService(lawyerItem);
82 |
83 |   expect(result).toEqual({
84 |     success: false,
85 |     errors: {
86 |       lawyer: lawyerItem,
87 |       errorDetails: [
88 |         'El nombre "123 Ana" no es válido.',
89 |         'El email "ana.lopez@correo" no es válido.'
90 |       ]
91 |     }
92 |   });
93 | });

Figura 7.23: Ejemplo de caso de prueba de ValidateLawyerDataService.test.jsx
```

- **ValidateCaseDataService.test.jsx.** Verifica los datos relacionados con los casos, como el nombre del cliente, el letrado, el formato del expediente, las fechas y el NIG. Se han definido casos de prueba para datos válidos e inválidos, asegurando que se detecten errores de formato (ver figura 7.24).

### 7.2.2. Pruebas de formateo de datos

A continuación, se listan los archivos de pruebas de los formateos implementados, mostrando algún ejemplo de los casos de prueba planteados para cada uno.

```

31  it('debe devolver error cuando falta un campo requerido', () => {
32    const caseItem = {
33      expediente: 'PENAL 1024/15',
34      letrado: 'Jorge Pérez',
35      'dado en fecha': '10/07/2020',
36      pago: 'pendiente',
37      nig: '2006943220210011185',
38    };
39
40    const result = ValidateCaseDataService(caseItem);
41
42    expect(result).toEqual({
43      success: false,
44      errors: {
45        case: caseItem,
46        errorDetails: ['El campo "cliente" es obligatorio y no puede estar vacío.'],
47      },
48    });
49  });

```

Figura 7.24: Ejemplo de caso de prueba de ValidateCaseDataService.test.jsx

- **FormatStringService.test.jsx.** Se prueban diferentes cadenas de texto para asegurar que se eliminen algunos caracteres como acentos y tildes, y se conviertan todas las letras a minúsculas, manteniendo un formato uniforme (ver figura 7.25).

```

5  describe('FormatStringService', () => {
6    it('debe eliminar acentos y convertir a minúsculas', () => {
7      expect(FormatStringService('Árbol')).toBe('arbol');
8      expect(FormatStringService('MÚSICA')).toBe('musica');
9      expect(FormatStringService('PÍNGÜINO')).toBe('pinguino');
10     });
11   });

```

Figura 7.25: Casos de prueba de FormatStringService.test.jsx

- **FormatNameService.test.jsx.** Este archivo se enfoca en el formateo de nombres y apellidos, asegurando que la primera letra de cada palabra esté en mayúscula y que se eliminen los espacios en blanco no deseados, manteniendo un formato coherente (ver figura 7.26).
- **FormatPhoneService.test.jsx.** Prueba el formateo de números de teléfono, eliminando prefijos de país y espacios innecesarios, y dejándolos en el formato definido, como se muestra en la figura 7.27.
- **FormatHeadersService.test.jsx.** Se realizan pruebas para asegurar que los encabezados proporcionados por los usuarios se normalicen correctamente, como se observa en la figura 7.28.

## 7. PRUEBAS DEL SISTEMA

---

```
5  describe('FormatNameService', () => {
6    it('debe formatear correctamente los nombres', () => {
7      expect(FormatNameService('juan perez')).toBe('Juan Perez');
8      expect(FormatNameService('MARIA GARCIA')).toBe('Maria Garcia');
9      expect(FormatNameService('jósé luis o\'Connor')).toBe('José Luis O'Connor');
10     expect(FormatNameService('  juan perez  ')).toBe('Juan Perez');
11     expect(FormatNameService('ana-maria')).toBe('Ana-maria');
12   });
13});
```

Figura 7.26: Casos de prueba de FormatNameService.test.jsx

```
5  describe('FormatPhoneService', () => {
6    it('debe eliminar el prefijo +34 y los espacios', () => {
7      expect(FormatPhoneService('+34 747 434 346')).toBe('747434346');
8      expect(FormatPhoneService('+34 612345678')).toBe('612345678');
9    });
10
11   it('debe mantener el número si no tiene el prefijo +34', () => {
12     expect(FormatPhoneService('747434346')).toBe('747434346');
13     expect(FormatPhoneService('612 345 678')).toBe('612345678');
14   });
15});
```

Figura 7.27: Casos de prueba de FormatPhoneService.test.jsx

```
27  const normalizedHeadersLawyers = ["email", "name", "phone"];
28
29  it('debe omitir las claves que no coincidan con los encabezados esperados', () => {
30    const item = {
31      "EMAIL": "miklorres@gmail.com",
32      "name": "Lara Martínez",
33      "PHONE": "732 671 823",
34      "Edad": 45
35    };
36
37    const result = FormatHeadersService(item, normalizedHeadersLawyers);
38    const expected = {
39      "email": "miklorres@gmail.com",
40      "name": "Lara Martínez",
41      "phone": "732 671 823"
42    };
43
44    expect(result).toEqual(expected);
45  });
46});
```

Figura 7.28: Ejemplos de casos de prueba de FormatHeadersService.test.jsx

### 7.3. Pruebas de integración

Las pruebas de integración se centran en verificar que los distintos componentes del sistema funcionen correctamente en conjunto. El objetivo principal es verificar el correcto almacenamiento de las bases de datos en MongoDB y el funcionamiento de los procesos más complejos como el envío de correos electrónicos

## 7.4. Resumen de resultados

### 7.3.1. Pruebas de carga de las bases de datos

En estas pruebas, se han utilizado los mismos datos empleados en las pruebas del backend, pero esta vez realizando la carga directamente desde la interfaz de la aplicación, simulando así el comportamiento real del sistema. Además de pruebas individuales, como la carga de un único abogado, se han llevado a cabo pruebas con archivos que contienen múltiples registros para verificar la correcta gestión y almacenamiento de datos.

### 7.3.2. Pruebas de envío de correos electrónicos

El sistema de envío de correos electrónicos ha sido probado directamente desde la aplicación, tanto mediante la función manual, como la automática. En todas las pruebas, se ha verificado que los correos lleguen al destinatario correspondiente, que estén correctamente estructurados y contengan toda la información necesaria, incluyendo los datos del usuario y los detalles del caso. Además, se ha comprobado que cada correo tenga el archivo PDF de la notificación correctamente adjunto, asegurando que el contenido enviado cumpla con los requisitos establecidos.

A continuación, se muestra una imagen donde se visualiza la bandeja de entrada del correo *miklorres@gmail.com* tras el envío automático de 3 notificaciones (ver figura 7.29). En este caso, se ha utilizado este correo de prueba como destinatario para todos los abogados asociados a los casos de las notificaciones enviadas. De esta manera, se asegura que todas las notificaciones lleguen al mismo correo, facilitando la comprobación de los resultados.

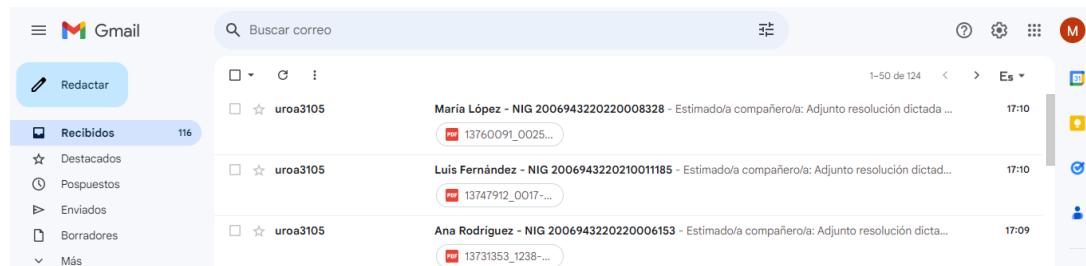


Figura 7.29: Prueba de envío automático de 3 notificaciones

En la siguiente imagen se muestra los detalles del contenido de uno de los correos recibidos (ver figura 7.30).

## 7.4. Resumen de resultados

Todas las pruebas de validación, formateo, integración y seguridad han sido finalmente exitosas. Durante el proceso de desarrollo y pruebas, se identificaron y corrigieron algunos fallos menores, principalmente relacionados con detalles de formato o casos específicos no previstos inicialmente. Estas pruebas han permitido ajustar esos errores leves, asegurando que todos los componentes funcionen correctamente.

## 7. PRUEBAS DEL SISTEMA

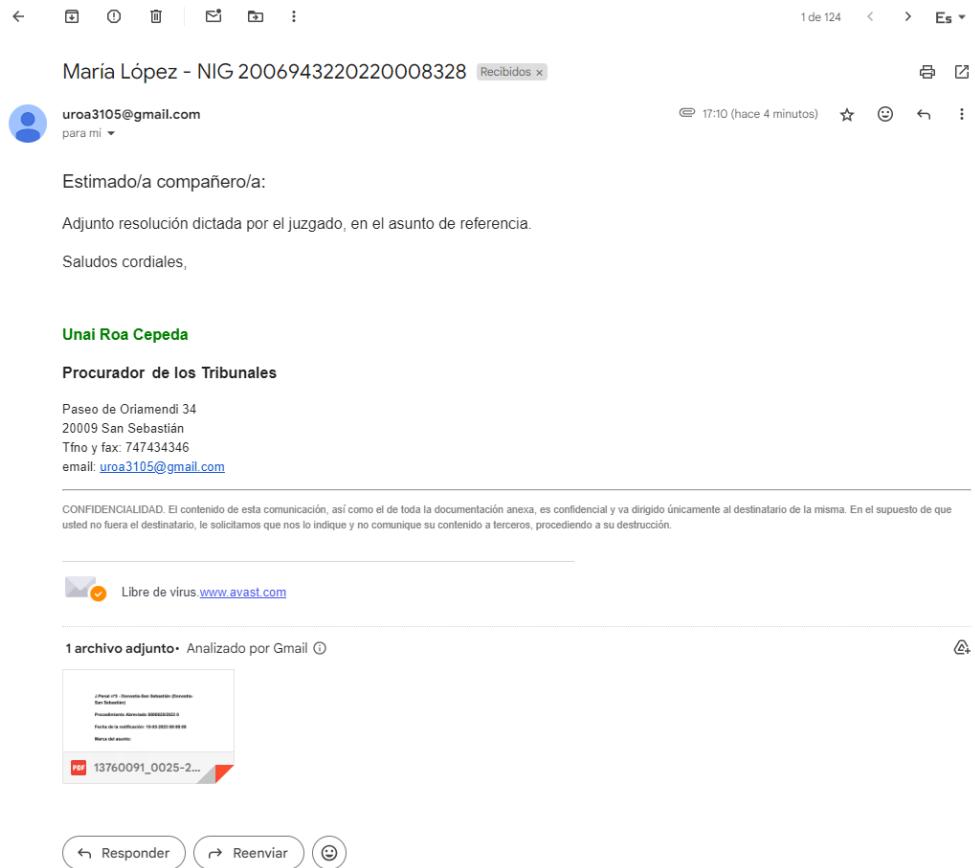


Figura 7.30: Prueba de estructura de correo recibido

A continuación, se muestra una imagen con los resultados de la ejecución de las pruebas de validación y formateo de datos (ver figura 7.31).

```
✓ src/tests/validationTests/ValidateExpedientService.test.jsx (2)
✓ src/tests/validationTests/ValidateHeadersService.test.jsx (4)
✓ src/tests/validationTests/ValidateLawyerDataService.test.jsx (5)
✓ src/tests/validationTests/ValidateNameService.test.jsx (2)
✓ src/tests/validationTests/ValidateNIGService.test.jsx (2)
✓ src/tests/validationTests/ValidatePayService.test.jsx (2)
✓ src/tests/validationTests/ValidatePhoneService.test.jsx (2)

Test Files 14 passed (14)
Tests 41 passed (41)
Start at 17:23:59
Duration 5.02s (transform 1.18s, setup 0ms, collect 5.13s, tests 187ms, environment 11ms, prepare 5.35s)
```

Figura 7.31: Ejecución de las pruebas de validación y formateo de datos

# CAPÍTULO 8

## Seguimiento y control

En este capítulo se detallan las actividades de seguimiento y control realizadas durante el proyecto, abarcando reuniones, gestión de retrasos, cambios en la planificación y comparación del tiempo invertido.

### 8.1. Reuniones y actas

Durante el desarrollo del proyecto, se han realizado reuniones periódicas con los tutores y el cliente. Éstas han sido fundamentales para orientar el trabajo y asegurar el cumplimiento de los objetivos establecidos, además de garantizar la satisfacción del cliente. Se ha recogido toda la información de estas comunicaciones en actas de reuniones, incluyendo la fecha, temas abordados y decisiones adoptadas. Las actas completas de cada reunión se encuentran en el anexo de las [Actas de reuniones](#).

### 8.2. Retrasos significativos

Inicialmente, el proyecto estaba planificado para ser entregado el 14/09/2024, sin embargo, tras evaluar la viabilidad de cumplir con este plazo durante la reunión 8, se determinó que no era posible completar todas las tareas previstas a tiempo. Por ello, se decidió posponer la entrega hasta la siguiente convocatoria, programada para el 27/10/2024.

#### 8.2.1. Causas del retraso

El retraso fue causado principalmente por la necesidad de dedicar más tiempo del previsto a ciertas tareas del proyecto, así como a la incorporación de nuevas funcionalidades solicitadas durante las reuniones con el cliente.

### 8.2.2. Impacto del retraso

El retraso generó una desviación en los plazos inicialmente establecidos, lo que afectó a la planificación general del proyecto. Además, influyó en la disponibilidad de recursos, ya que fue necesario asignar tiempo adicional para implementar las funcionalidades añadidas.

### 8.2.3. Acciones adoptadas

Para gestionar este retraso, se realizó una replanificación del diagrama de Gantt, ajustando los plazos de entrega y añadiendo una nueva iteración en el desarrollo del proyecto para abordar las nuevas funcionalidades.

## 8.3. Cambios en la planificación

En consecuencia de los ajustes realizados debido al retraso y a la incorporación de nuevas funcionalidades, se llevaron a cabo modificaciones en la planificación del proyecto. Estas modificaciones se reflejan en los siguientes diagramas actualizados.

### 8.3.1. Diagrama de Estructura de Desglose del Trabajo (EDT) extendido

Se han añadido nuevas tareas para los siguientes paquetes de trabajo.

- **Paquete de Diseño.** Se ha incorporado el diseño del calendario e instrucciones del usuario.
- **Paquete de Desarrollo.** En la iteración *IT2* se ha añadido la tarea de ordenación de las tablas de casos y abogados por atributos, propuesta por el cliente en la reunión [5](#). La iteración *IT3* se ha actualizado con la incorporación del envío manual de correos electrónicos, decidido durante la reunión [6](#). Además, se ha creado la iteración *IT5*, que se centra en el desarrollo del calendario de juicios y en las instrucciones para el usuario, abarcando la extracción de fechas de juicio mediante OCR, la visualización del calendario y la creación de instrucciones, conforme a lo acordado en la reunión [7](#).
- **Paquete de Pruebas.** Se ha actualizado para contemplar las pruebas correspondientes a las funcionalidades implementadas en la iteración 5.

En la figura [8.1](#) se muestra la representación visual de la Estructura de Desglose de Trabajo.

### 8.3.2. Diagrama de Gantt actualizado

Se han añadido las nuevas tareas incluidas en el EDT y se han ajustado los tiempos para garantizar la entrega final del proyecto en la fecha establecida, el 27/10/2024. En la figura [8.2](#) se muestra la planificación actualizada mediante el diagrama Gantt.

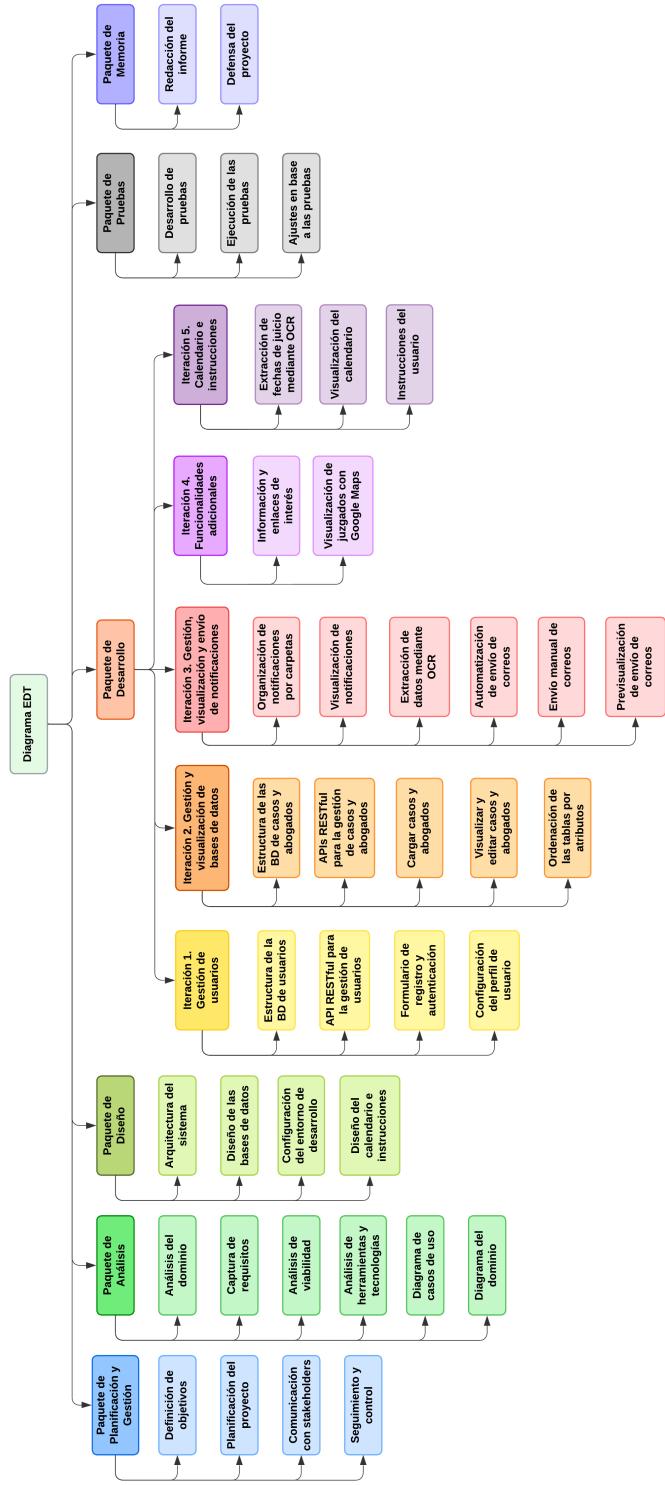
### 8.3.3. Diagrama de casos de uso extendido

Se han incluido los casos de uso relacionados con las nuevas funcionalidades implementadas durante el desarrollo del proyecto. En la figura 8.3 se muestra el diagrama de casos de uso extendido.

## 8.4. Seguimiento del tiempo invertido

Durante el desarrollo del proyecto, se ha realizado un seguimiento detallado del tiempo invertido en cada tarea. Al comparar las horas de dedicación previstas con las horas realmente empleadas, se observa una desviación significativa en algunos paquetes de trabajo. Concretamente, se ha requerido más tiempo del inicialmente estimado en el Paquete de Desarrollo y en el Paquete de Memoria, debido a la incorporación de nuevas funcionalidades, la complejidad de implementar algunas tareas y el esfuerzo adicional dedicado a la redacción del informe. En la figura 8.4 se muestra una comparación detallada entre las horas previstas y las horas efectivamente dedicadas a cada tarea del proyecto.

## 8. SEGUIMIENTO Y CONTROL



**Figura 8.1:** Diagrama de Estructura de Desglose de Trabajo extendido

## 8.4. Seguimiento del tiempo invertido

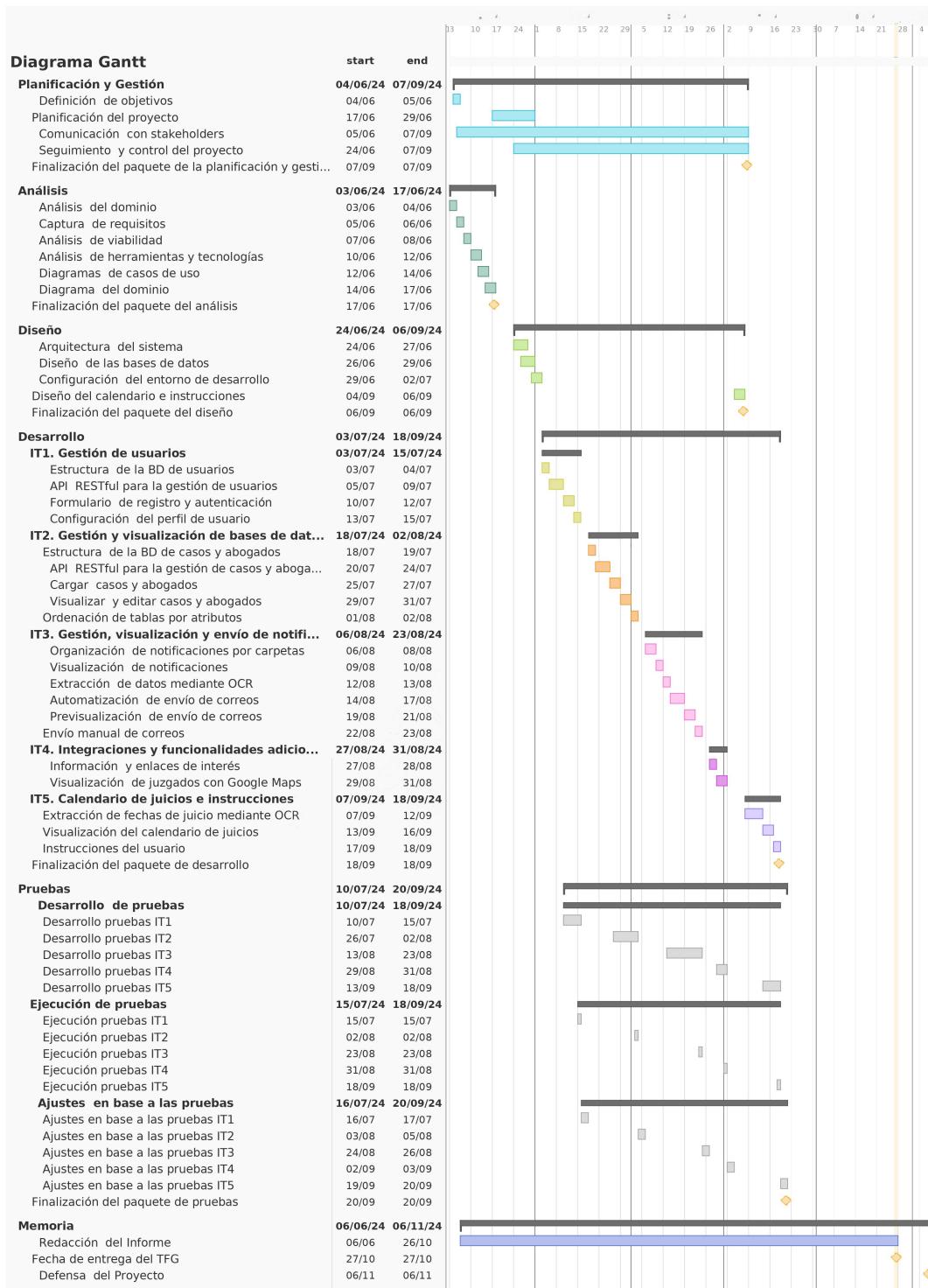
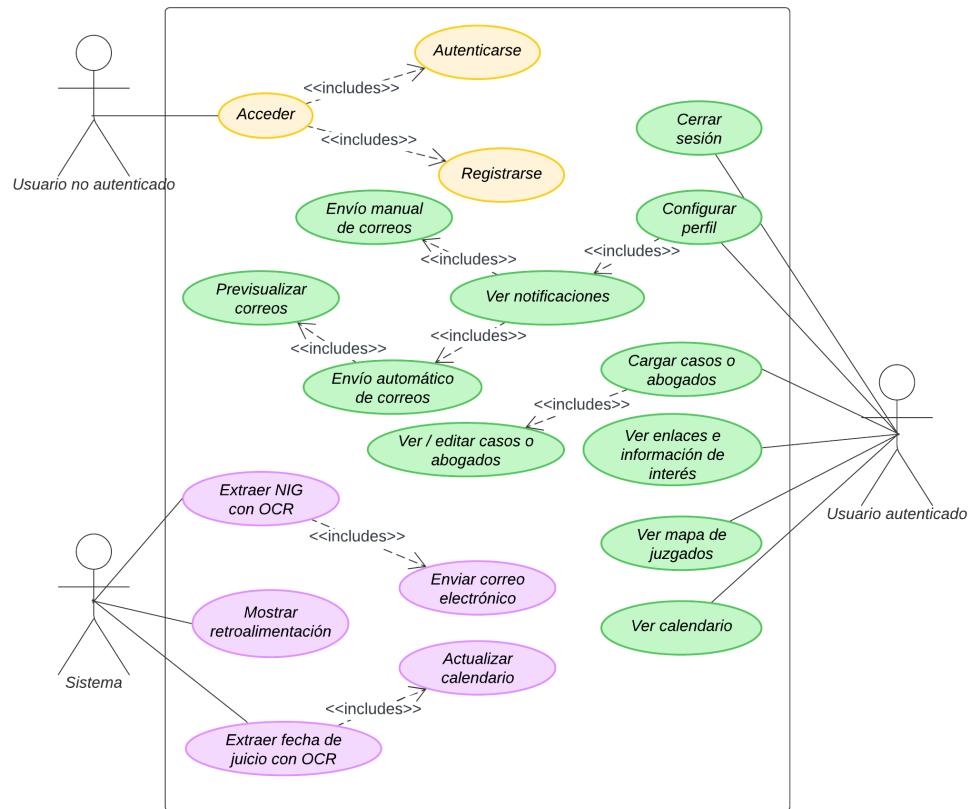


Figura 8.2: Diagrama Gantt actualizado

## 8. SEGUIMIENTO Y CONTROL

---



**Figura 8.3:** Diagrama de casos de uso extendido

#### 8.4. Seguimiento del tiempo invertido

Paquete de Trabajo / Iteraciones	Tareas	Estimación prevista (h)	Dedicación real (h)
Paquete de trabajo Planificación y Gestión	Definición de objetivos	4	4
	Planificación del proyecto	20	20
	Comunicación con stakeholders	10	12
	Seguimiento y control	10	10
Paquete de trabajo Análisis	Análisis del dominio	4	4
	Captura de requisitos	4	4
	Análisis de viabilidad	2	2
	Análisis de herramientas y tecnologías	8	8
	Diagrama de casos de uso	8	6
	Diagrama del dominio	8	6
Paquete de trabajo Diseño	Arquitectura del sistema	10	10
	Diseño de las bases de datos	8	6
	Configuración del entorno de desarrollo	5	6
	Diseño del calendario e instrucciones	-	4
Paquete de trabajo Desarrollo	IT1. Gestión de usuarios	Estructura de la BD de usuarios	4
		API RESTful para la gestión de usuarios	12
		Formulario de registro y autenticación	8
		Configuración del perfil de usuario	6
	IT2. Gestión y visualización de bases de datos	Estructura de la BD de casos y abogados	6
		API RESTful para la gestión de casos y abogados	12
		Cargar casos y abogados	8
		Visualizar y editar casos y abogados	8
		Ordenación de las tablas por atributos	-
	IT3. Gestión, visualización y envío de notificaciones	Organización de notificaciones por carpetas	8
		Visualización de notificaciones	6
		Extracción de datos mediante OCR	6
		Automatización de envío de correos	12
		Envío manual de correos	-
		Previsualización de envío de correos	8
	IT4. Integraciones y funcionalidades	Información y enlaces de interés	6
		Visualización de juzgados con Google Maps	8
	IT5. Calendario e instrucciones	Extracción de fechas de juicio mediante OCR	-
		Visualización del calendario	-
		Instrucciones del usuario	-
Paquete de trabajo Pruebas	Desarrollo de pruebas	10	10
	Ejecución de pruebas	4	2
	Ajustes en base a las pruebas	10	8
Paquete de trabajo Memoria	Redacción del Informe	60	70
	Defensa del Proyecto	6	6
<b>Horas totales</b>		<b>309</b>	<b>342</b>

**Figura 8.4:** Comparación entre dedicación prevista y real



# CAPÍTULO 9

## Conclusión y trabajo a futuro

En este capítulo se presentan los resultados alcanzados durante el desarrollo del proyecto, destacando el cumplimiento de los objetivos, las lecciones aprendidas y las oportunidades de mejora y expansión de la plataforma.

### 9.1. Cumplimiento de los Objetivos

A lo largo del desarrollo de este proyecto, se ha logrado cumplir con los objetivos establecidos inicialmente, tanto principales como secundarios. La plataforma desarrollada proporciona a los procuradores una herramienta robusta para gestionar las notificaciones junto a los casos y abogados relacionados. Además, se han añadido algunas mejoras solicitadas por el cliente, adaptando el sistema a sus necesidades específicas.

### 9.2. Lecciones aprendidas

Durante el desarrollo del proyecto, una de las lecciones más importantes ha sido la gestión de imprevistos y cambios en los plazos. Aunque se había identificado previamente el riesgo de retrasos en la planificación, la implementación de nuevas funcionalidades y la subestimación del tiempo requerido, especialmente en la fase de desarrollo, llevaron a una desviación significativa en los plazos inicialmente establecidos. Es por ello que se ha aprendido la importancia de realizar estimaciones más realistas y de incorporar márgenes de tiempo para abordar posibles imprevistos. Además, la planificación debe ser ciertamente flexible para ajustarse según surjan nuevas necesidades.

Otra lección clave ha sido mantener una comunicación constante con el cliente para recibir su feedback de manera continua. Este enfoque ha permitido realizar ajustes en etapas tempranas, evitando así tener que deshacer trabajo ya completado.

### **9.3. Trabajo a futuro**

A pesar de los logros alcanzados, se han identificado varios aspectos de mejora y expansión que permitirán optimizar el sistema y adaptarlo a nuevas necesidades.

#### **9.3.1. Despliegue y mantenimiento de la aplicación**

En el futuro, se podría llevar a cabo la implementación y despliegue del sistema en un entorno de producción. Además, se puede establecer un plan de mantenimiento para gestionar actualizaciones, detectar problemas y mejorar el rendimiento.

#### **9.3.2. Expansión a nivel nacional**

Aunque la plataforma está actualmente diseñada para los procuradores del País Vasco, se propone extender su alcance a otras comunidades autónomas de España. Esta expansión implicará adaptar el sistema a los procedimientos específicos de cada región, así como la integración con otras plataformas de gestión judicial utilizadas fuera del País Vasco.

#### **9.3.3. Integración de nuevas funcionalidades**

Se pueden explorar mejoras adicionales para ampliar las prestaciones de la plataforma. Por ejemplo, se podría ampliar la gestión de pagos actual, añadiendo automatización de recordatorios y generación de facturas.

#### **9.3.4. Monitoreo de la aplicación**

Se puede implementar un sistema de monitoreo continuo para supervisar el rendimiento, detectar posibles errores y garantizar un funcionamiento óptimo. Esto permitirá actuar rápidamente ante problemas, mejorando la experiencia de los usuarios.

### **9.4. Conclusiones**

Finalmente, el proyecto ha alcanzado satisfactoriamente los objetivos establecidos, proporcionando una plataforma robusta para los procuradores en el País Vasco. A pesar de los desafíos encontrados, como retrasos y ajustes en los plazos, se han cumplido las expectativas del cliente, adaptando el sistema a sus necesidades.

Además, desarrollar una aplicación completa ha implicado afrontar y resolver retos constantemente, como la complejidad de la extracción de datos de archivos PDF. Superar estos desafíos ha enriquecido la experiencia adquirida y ha fortalecido las habilidades en el desarrollo de software. Las lecciones aprendidas permitirán afrontar futuros proyectos con una visión más realista y una planificación más eficiente.

En definitiva, este proyecto ha sentado las bases para mejoras futuras, consolidando una solución tecnológica que puede seguir evolucionando para adaptarse a nuevas exigencias del sector.

# Apéndice



# Actas de reuniones

## Acta UROA-TFG-AR-TUTORES-15/01/2024

<b>Fecha</b>	15/01/2024
<b>Medio de comunicación</b>	Videoconferencia
<b>Participantes</b>	Idoia Berges, Ekaitz Jauregi, Unai Roa
<b>Temas abordados</b>	Propuesta de idea del TFG
<b>Decisiones adoptadas:</b>	
- Definir mejor el alcance y los objetivos principales. - Establecer las funcionalidades a implementar para evaluar la viabilidad del proyecto.	

Tabla 1: UROA-TFG-AR-TUTORES-15/01/2024

## Acta UROA-TFG-AR-CLIENTE-04/06/2024

<b>Fecha</b>	04/06/2024
<b>Medio de comunicación</b>	Presencial
<b>Participantes</b>	David Juan Alfonso, Unai Roa
<b>Temas abordados</b>	Requisitos y funcionalidades principales de la aplicación y estimación del tiempo de desarrollo.
<b>Decisiones adoptadas:</b>	
- Objetivos y funcionalidades principales de la aplicación definidos. - Priorizar la automatización de los correos electrónicos y la gestión de los casos y abogados. - Establecer como fecha estimada de finalización el 1/09/2024.	

Tabla 2: UROA-TFG-AR-CLIENTE-04/06/2024

**Acta UROA-TFG-AR-CLIENTE-29/06/2024**

<b>Fecha</b>	29/06/2024
<b>Medio de comunicación</b>	Presencial
<b>Participantes</b>	David Juan Alfonso, Unai Roa
<b>Temas abordados</b>	Presentación del análisis y diseño propuesto.
<b>Decisiones adoptadas:</b>	
<ul style="list-style-type: none"> <li>- Aprobación del análisis y diseño propuesto.</li> <li>- Sugerencia de que cada usuario tenga sus propias bases de datos privadas.</li> <li>- Comenzar con el desarrollo de la gestión de usuarios.</li> </ul>	

**Tabla 3:** UROA-TFG-AR-CLIENTE-29/06/2024**Acta UROA-TFG-AR-CLIENTE-15/07/2024**

<b>Fecha</b>	15/07/2024
<b>Medio de comunicación</b>	Presencial
<b>Participantes</b>	David Juan Alfonso, Unai Roa
<b>Temas abordados</b>	Gestión de usuarios.
<b>Decisiones adoptadas:</b>	
<ul style="list-style-type: none"> <li>- Resolución de dudas.</li> <li>- Aprobación del desarrollo propuesto.</li> </ul>	

**Tabla 4:** UROA-TFG-AR-CLIENTE-15/07/2024**Acta UROA-TFG-AR-CLIENTE-30/07/2024**

<b>Fecha</b>	30/07/2024
<b>Medio de comunicación</b>	Presencial
<b>Participantes</b>	David Juan Alfonso, Unai Roa
<b>Temas abordados</b>	Gestión y visualización de bases de datos.
<b>Decisiones adoptadas:</b>	
<ul style="list-style-type: none"> <li>- Propuesta de añadir métodos de ordenación en las tablas.</li> <li>- Aprobación del desarrollo propuesto.</li> </ul>	

**Tabla 5:** UROA-TFG-AR-CLIENTE-30/07/2024

## Acta UROA-TFG-AR-CLIENTE-20/08/2024

<b>Fecha</b>	20/08/2024
<b>Medio de comunicación</b>	Presencial
<b>Participantes</b>	David Juan Alfonso, Unai Roa
<b>Temas abordados</b>	Gestión, visualización y envío de notificaciones.
<b>Decisiones adoptadas:</b>	
<ul style="list-style-type: none"> <li>- Propuesta de añadir envío manual de los correos electrónicos.</li> <li>- Aprobación del desarrollo propuesto.</li> <li>- Implementar el nuevo caso de uso y comenzar con la iteración 3.</li> </ul>	

Tabla 6: UROA-TFG-AR-CLIENTE-20/08/2024

## Acta UROA-TFG-AR-CLIENTE-30/08/2024

<b>Fecha</b>	30/08/2024
<b>Medio de comunicación</b>	Presencial
<b>Participantes</b>	David Juan Alfonso, Unai Roa
<b>Temas abordados</b>	Integraciones y funcionalidades adicionales.
<b>Decisiones adoptadas:</b>	
<ul style="list-style-type: none"> <li>- Propuesta de añadir un apartado con instrucciones.</li> <li>- Propuesta de añadir un calendario de juicios.</li> <li>- Aprobación del desarrollo propuesto.</li> <li>- Implementar los nuevos casos de uso</li> </ul>	

Tabla 7: UROA-TFG-AR-CLIENTE-30/08/2024

## Acta UROA-TFG-AR-TUTORES-04/09/2024

<b>Fecha</b>	04/09/2024
<b>Medio de comunicación</b>	Presencial
<b>Participantes</b>	Idoia Berges, Ekaitz Jauregi, Unai Roa
<b>Temas abordados</b>	Presentación de los progresos realizados en la parte del desarrollo del proyecto y evaluación de la viabilidad de presentar el proyecto para la fecha prevista.
<b>Decisiones adoptadas:</b>	
<ul style="list-style-type: none"> <li>- No realizar la matrícula para la convocatoria prevista.</li> <li>- Prolongar el proyecto hasta la siguiente convocatoria, con fecha de entrega el 27/10.</li> </ul>	

Tabla 8: UROA-TFG-AR-TUTORES-04/09/2024

**Acta UROA-TFG-AR-CLIENTE-20/09/2024**

<b>Fecha</b>	20/09/2024
<b>Medio de comunicación</b>	Presencial
<b>Participantes</b>	David Juan Alfonso, Unai Roa
<b>Temas abordados</b>	Ultimos detalles de la aplicacion.
<b>Decisiones adoptadas:</b>	
<ul style="list-style-type: none"> <li>- Muestra de ultimas pruebas.</li> <li>- Cliente conforme con los resultados obtenidos.</li> </ul>	

**Tabla 9:** UROA-TFG-AR-CLIENTE-20/09/2024**Acta UROA-TFG-AR-TUTORES-26/10/2024**

<b>Fecha</b>	26/10/2024
<b>Medio de comunicación</b>	Videoconferencia
<b>Participantes</b>	Ekaitz Jauregi, Unai Roa
<b>Temas abordados</b>	Revisión de los últimos detalles del informe final.
<b>Decisiones adoptadas:</b>	
<ul style="list-style-type: none"> <li>- Realizar ajustes finales en el informe.</li> <li>- Preparar la entrega para la fecha acordada.</li> </ul>	

**Tabla 10:** UROA-TFG-AR-TUTORES-26/10/2024

# Bibliografía

- [1] Licencia creative common. <https://creativecommons.org/licenses/by-nc-nd/4.0/legalcode.es>, 2024. Ver página 11.
- [2] Java vs javascript vs python. <https://stackshare.io/stackups/java-vs-javascript-vs-python>, 2024. Ver página 32.
- [3] Mohamed IDBRAHIM. Javascript vs typescript. <https://www.blog.brightcoding.dev/2023/09/27/javascript-vs-typescript-a-comprehensive-comparison/>, 2023. Ver página 34.
- [4] Dan Ochoa. React vs angular. <https://www.halfnine.com/blog/post/react-vs-angular>, 2024. Ver página 35.
- [5] Sarthak Niranjan. Webpack vs vite. <https://codeparrot.ai/blogs/webpack-vs-vite-a-detailed-comparison-for-modern-web-development>, 2024. Ver página 36.
- [6] Referencia figura mvc. <https://www.linkedin.com/pulse/modelo-vista-controlador-precognis>, 2022. Ver página 39.
- [7] Rick Anderson, Theano Petersen, Scott Addie, Nick Schonning, Andy Pasic, and Tom Dyksstra. Modelo vista controlador. <https://learn.microsoft.com/es-es/aspnet/mvc/overview/older-versions-1/overview/asp-net-mvc-overview>, 2024. Ver página 39.