UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

DEPARTAMENTO DE ELECTROTECNIA E INFORMÁTICA
INGENIERÍA EN INFORMÁTICA

Desarrollo de un Backend para la Optimización del Proceso de Retiro de Contenedores en el puerto de Valparaíso

Vicente Tomas Ramirez Gonzalez

vicente.ramirezg@usm.cl

Carlos Alten
Profesor Guía

Resumen: En los patios de contenedores, el proceso de retiro diario involucra a múltiples actores y una serie de tareas que deben realizarse de manera eficiente para garantizar el éxito del retiro. Sin embargo, problemas comunes como errores tipográficos, inconsistencias en los documentos y retrasos logísticos afectan negativamente el flujo del proceso, causando pérdidas de tiempo y recursos. Este trabajo propone una solución tecnológica que consiste en el desarrollo de una aplicación móvil y una interfaz web. Estas herramientas están diseñadas para agilizar y asegurar la gestión de documentos asociados al retiro de contenedores. Además, se incorporan algoritmos de machine learning para automatizar la revisión de documentos, minimizando errores humanos y detectando anomalías cuando se requiera de esta herramienta, contribuyendo al uso eficiente del espacio en los patios y aumentando la productividad operativa. Este sistema tiene el potencial de optimizar la logística y administración de contenedores, sirviendo como modelo para otras industrias con flujos documentales complejos

Palabras Clave: Gestión de documentos, Aplicación Móvil, Interfaz Web, Reducción de errores, Optimización logística.

1 Introducción

Este capítulo abarca el contexto general del proyecto, explicando los problemas actuales en el proceso de retiro de contenedores en los patios de carga, detallando como afectan a la logística portuaria y comercial.

Asimismo, se describe la propuesta de solución, basada en el desarrollo de una aplicación móvil y una plataforma web que optimiza la gestión documental, la comunicación entre actores y el seguimiento del estado de los tramites, planteando la relevancia del sistema propuesto para mejorar la eficiencia operativa, trazabilidad y la seguridad de los procesos logísticos.

1.1 Contexto y antecedentes

El proceso de retiro de contenedores en los patios de carga es una operación critica dentro de la logística portuaria y comercial. Este procedimiento garantiza que los contenedores con mercancías diversas puedan salir de los patios de carga de manera segura, eficiente y conforme a las normativas legales y operativas. Lo principal de este proceso es asegurar que cada contenedor y su contenido sean entregados al transportista correcto, evitando errores, retrasos o irregularidades que afecten a la cadena de suministro.

Este proceso involucra la colaboración de varios actores clave:

- **Agente de visado**: Encargado de revisar y validar los documentos emitidos por la aduana y los presentados por el solicitante. Este agente es responsable de garantizar que los datos sean correctos y cumplan con las normativas establecidas.
- **Personal de Aduanas**: Proporciona la documentación oficial que certifica la legalidad y el contenido del contenedor, incluyendo permisos de exportación o importación.
- **Agente Tramitador**: Gestiona los trámites administrativos necesarios para autorizar el retiro de la carga. Esto incluye la preparación de documentos y la comunicación con otros actores.
- **Chofer o Transportista**: Responsable de recoger físicamente el contenedor autorizado y transportarlo a su destino final.

La validación precisa y eficiente de los documentos es el elemento más importante de este proceso, debido a que el papeleo incluye información crítica como datos del exportador o importador, contenido de los contenedores, datos del transportista y permisos legales. Cualquier discrepancia en esta información puede generar retrasos, costos adicionales e incluso problemas legales.

El objetivo final de este proceso es optimizar la salida de contenedores, asegurando que las mercancías lleguen a su destino sin contratiempos. Esto implica minimizar errores humanos, reducir tiempos de espera y mejorar la trazabilidad de los tramites, contribuyendo así a una cadena logística más eficiente y segura.

Actualmente, se utilizan sistemas tecnológicos que facilitan la comunicación y la gestión de datos en este contexto. Sin embargo, estos sistemas deben ser altamente eficientes, rápidos, intuitivos y a prueba de fallos, ya que cualquier interrupción puede generar demoras significativas en la operación. Dentro de la empresa con la que se trabajó, se emplean plataformas unificadas para gestionar documentos sensibles, sistemas de apoyo para la verificación detallada de la documentación y software especializado para la administración de bases de datos y almacenamientos de archivos. Aunque estas herramientas han mejorado parcialmente la eficiencia, persisten desafíos relacionados con la escalabilidad, la automatización y la prevención de errores humanos.

1.2 Definición del problema.

Durante el análisis del proceso de entrega de cargas en los patios de contenedores, se han identificado los siguientes problemas que afectan significativamente la eficiencia del proceso:

Nº	PROBLEMA IDENTIFICADO	DESCRIPCIÓN DEL PROBLEMA
1	Diferencias en la documentación	Se presentan inconsistencias tipográficas o de información al comparar los documentos de aduana con los de la persona encargada del retiro.
2	Documentación física	A pesar de los avances tecnológicos, se sigue requiriendo documentación en formato físico, lo que implica la necesidad de escanear repetidamente los documentos para su lectura en sistemas digitales.
3	Desconocimiento del estado del proceso	No existe manera eficiente de conocer el estado actual del proceso, ni tampoco una forma efectiva de notificar cambios o necesidades en tiempo real.
4	Dependencia del personal	El proceso depende en gran medida del personal para tareas que podrían ser gestionadas a través de una plataforma, como la actualización de estado de los trámites pendientes, notificación de cambios en los procesos o el estado y recepción de pagos.

5	Transferencias bancarias	Actualmente se utilizan transferencias bancarias, lo que resulta menos eficiente que una plataforma de pago dedicada. Además, este método puede ocasionar errores y retrasos, sin mencionar las nuevas regulaciones que imponen un límite de transferencias recibidas por día.
6	Archivos no centralizados	Los documentos se almacenan de manera desorganizada en diferentes servidores, lo que complica su acceso y gestión eficiente de la documentación.
7	Sistema tradicional de usuarios	No se cuenta con un sistema de autenticación formal, lo que dificulta la gestión de usuarios y las solicitudes de trámites y la gestión de ellas, como también el recibo y manejo de documentos.
8	Gestión ineficaz de personas	La ausencia de un sistema de autenticación impide el control efectivo de las personas que interactúan dentro del proceso, lo que puede dar lugar a fraudes u otras actividades maliciosas.

Tabla 1-1. Lista de problemas identificados con sus descripciones

Fuente: Elaboración propia

1.3 Breve descripción sobre la propuesta de solución

La solución propuesta consiste en desarrollar una aplicación móvil y una plataforma web que optimicen el proceso de retiro de contenedores. La aplicación móvil permitirá a los agentes tramitadores gestionar pagos pendientes, subir documentos y revisar notificaciones, mientras que los conductores podrán consultar el estado del trámite, detalles de la carga y su ubicación. Por otro lado, la plataforma web estará dirigida al agente de visado, quien podrá administrar usuarios, asignar cargas, generar tramites y realizar comprobaciones automáticas de documentos utilizando plantillas predefinidas para verificar consistencias.

Desde el backend, se implementarán funcionalidades clave como la gestión de usuarios, autenticación segura y manejo de documentos mediante algoritmos de comprobación basados en machine learning. También se desarrollarán APIs para garantizar una integración eficiente entre las interfaces móviles y web, asegurando una comunicación segura y precisa.

Figura 1–1. Infografía de la solución propuesta

Fuente: Elaboración Propia

Límites del proyecto:
El alcance incluye el desarrollo de las funcionalidades mencionadas, excluyendo optimizaciones avanzadas como análisis predictivo o simulaciones logísticas, integración de sistema de notificaciones vía email e implementación de pasarela de pago.

1.4 Objetivos Generales y Específicos de la Tesina

Desarrollar un sistema backend eficiente que cumpla con estándares básicos de seguridad y permita gestionar los trámites en patios de contenedores mediante la integración de una plataforma web y una aplicación móvil, optimizando la administración de usuarios, documentos y notificaciones, garantizando la coherencia y precisión en los procesos

Objetivos específicos:

1. **Implementar un sistema de autenticación y autorización seguro** que permita a los usuarios acceder a las plataformas móvil y web según sus roles y funciones.
2. **Diseñar y desarrollar un módulo de gestión de usuarios**, que facilite la creación, edición, des habilitación, habilitación y asignación de roles en la plataforma web.
3. **Crear una funcionalidad para la carga y descarga de documentos** en un sistema de almacenamiento seguro, accesible desde las interfaces móvil y web.
4. **Implementar el manejo de tramites** dentro de un sistema para agilizar la gestión del proceso.
5. **Desarrollar un algoritmo basado en machine learning** para la lectura, análisis y comprobación automática de documentos PDF o imágenes subidas, utilizando plantillas predefinidas para verificar coherencia y detectar discrepancias.
6. **Implementar un sistema de notificaciones en tiempo real**, que permita informar a los agentes y conductores sobre el estado de los tramites y eventos relevantes.
7. **Diseñar APIs robustas y seguras** para garantizar una comunicación eficiente e integrada entre la aplicación móvil y plataforma web.
8. **Establecer mecanismos de validación y verificación de consistencia documental**, asegurando la integridad de la información procesada.

1.5 Justificación de proyecto

El desarrollo de este proyecto tiene como objetivo mejorar significativamente los tiempos y la eficiencia en la gestión de trámites relacionados con la entrega de cargas en patios de contenedores. La solución combina una aplicación móvil y una plataforma web para optimizar la entrega y recepción de documentos, facilitando el trabajo del agente tramitador como también del personal encargado de visado y conductor. Con ello, se busca reducir los errores humanos y agilizar procesos que tradicionalmente son lentos y propensos a inconsistencias.

Relevancia Técnica

Desde una perspectiva técnica, este proyecto destaca por el uso de tecnologías modernas que aportan robustez, escalabilidad y eficiencia al sistema backend. El backend está desarrollado en **Django**, un framework de alto nivel que permite implementar funcionalidades clave como la gestión de usuarios, roles y documentos, asegurando un diseño modular y escalable.

La base de datos utilizado es **PostgreSQL**, un sistema robusto y confiable que asegura el manejo eficiente de los datos relacionales. Al desplegar tanto el servidor Django como la base de datos PostgreSQL en contenedores Docker.

Además, se utiliza **PyTesseract** [1], una librería basada en Tesseract OCR [2], para la extracción de datos de documentos tanto PDF o imágenes transformadas a PDF. Esto mejora la precisión en el procesamiento de documentos y reduce significativamente los tiempos de revisión manual.

Beneficios

El proyecto ofrece beneficios clave tanto para los agentes tramitadores y conductores como también los agentes de visado:

1. **Eficiencia**: Reducción de los tiempos de espera mediante la digitalización y optimización de los procesos de tramitación de documentos.
2. **Escalabilidad**: Al aprovechar Docker y PostgreSQL, el sistema puede crecer fácilmente para manejar mayores volúmenes de datos y usuarios.
3. **Portabilidad y Flexibilidad**: El uso de contenedores permite desplegar el sistema en diferentes entornos sin necesidad de reconfiguración, facilitando pruebas y actualizaciones.

4. **Seguridad**: Gestión segura de datos y usuarios mediante autenticación robusta y mecanismos que protegen la integridad de la información procesada.
5. **Facilidad de mantenimiento**: Tecnologías ampliamente documentadas como Django, PostgreSQL y Docker hacen que el sistema sea más fácil de mantener y adaptar a nuevos requerimientos.

Innovación
La innovación del proyecto reside en su enfoque técnico al integrar tecnologías emergentes para resolver problemas prácticos. La utilización de contenedores Docker no solo asegura una mayor escalabilidad y facilidad de implementación, sino que también introduce una metodología moderna de despliegue que optimiza la gestión de recursos.

El uso de **PyTesseract** para la extracción de texto de documentos e imágenes es otro aspecto destacado, ya que automatiza parte del proceso de validación documental, reduciendo el margen de error y mejorando la eficiencia.

En resumen, el proyecto no aborda solo un problema practico de manejo de tramites en patio de contenedores, sino que también introduce una solución técnicamente relevante e innovadora, utilizando para este objetivo tecnologías que garantizan eficiencia, escalabilidad y robustez en la gestión de procesos críticos. La elección de Django, PostgreSQL y Docker asegura un sistema preparado para las necesidades actuales y futuras del sector.

1.6 Metodología

El enfoque que se empleará para el desarrollo será SCRUM, una metodología ágil que facilita la adaptación a cambios y permite un flujo constante de entregas de funcionalidades [1]. Para esto, se implementó un ciclo de trabajos cortos denominados sprints, donde la duración de cada uno es alrededor de una semana.

Dentro de los sprints, se establecerán metas claras y se priorizarán las funcionalidades más importantes a ser desarrolladas primero, para así lograr proporcionar incrementos de valor tangible al final de cada ciclo. Se llevará a cabo reuniones diarias para revisar el progreso, identificar obstáculos y asegurar que el equipo mantenga un flujo de trabajo optimo.

Con este enfoque se busca:

- Mejor adaptación a cambios de requisitos o prioridades del proyecto.
- Mantener una comunicación constante con los miembros del equipo y partes interesadas.
- Garantizar entregas continuas y de alta calidad a lo largo del ciclo de desarrollo.

Para lograr estos objetivos, se utilizarán herramientas colaborativas para la gestión de tareas, donde Teams será utilizado para lo antes mencionado, Git será utilizado para el control de versiones. Con esta metodología se busca optimizar los tiempos de desarrollo, mejorar la coordinación del equipo y asegurar que las funcionalidades se logren de manera incremental, de esta manera minimizando riesgos y mejorando la calidad del producto final.

1.7 Breve descripción de la organización del informe en capítulos

Dentro de este informe, se estructuran de manera lógica y progresiva con el objetivo de guiar al lector a través de la concepción, desarrollo y evaluación del proyecto. A continuación, se presenta una breve descripción de los capítulos:

1. **Introducción**: Visión general sobre el proyecto, su justificación, objetivos y metodología utilizada.
2. **Marco Teórico**: Base teórica que sustenta al proyecto. Se describen los fundamentos de backend, arquitecturas de software utilizadas, las tecnologías y frameworks seleccionados para la solución.
3. **Diseño e implementación**: Detalles del diseño del sistema. Se incluyen diagramas de arquitectura, modelos de datos y flujos de trabajo. Además, incluye las etapas de implementación del proyecto, especificando herramientas, lenguajes de programación y técnicas empleadas.
4. **Conclusiones**: Conclusiones derivadas del proyecto, evaluación personal del proyecto, limites identificados y aprendizajes a futuro.
5. **Agradecimientos**: Reconocimiento de las contribuciones de personas que apoyaron en el desarrollo del proyecto.
6. **Referencias**: Listado de fuentes bibliográficas, artículos y sitios web consultados durante el desarrollo del proyecto.
7. **Anexos**: Materiales adicionales que complementan al informe, como diagramas extendidos, listas relacionadas a las respuestas del sistema u otros documentos relevantes.

2 Marco Teórico

En este capítulo se establecen los fundamentos teóricos y técnicos que sustentan el desarrollo del proyecto, explicando los conceptos claves relacionados con la logística portuaria y el proceso de retiro de contenedores.

Además, se describen las tecnologías utilizadas, como APIs REST, Django, uso de Machine Learning para validación de documentos y las herramientas de infraestructura utilizadas, como lo son Docker y AWS. También se aborda el uso del patrón de diseño MVC y su aplicación en el sistema, junto con las ventajas de la metodología ágil SCRUM para la planificación y ejecución del proyecto.

Este capítulo proporciona el marco conceptual que conecta los fundamentos teóricos con las decisiones técnicas tomadas durante el desarrollo del sistema.

2.1 Fundamentos del desarrollo back-end

El desarrollo backend es una pieza esencial en la creación de sistemas y aplicaciones modernas, encargándose de la lógica del negocio, el procesamiento de datos y la interacción con bases de datos [1]. Desde la popularización de la web en la década de 1990, se hizo evidente la necesidad de un backend eficiente que soportara el creciente volumen de usuarios y la complejidad de las aplicaciones. En sus inicios, los servidores se centraban en tareas básicas como el manejo de bases de datos y la respuesta a solicitudes de los usuarios. Con el tiempo, estas funciones evolucionaron para incluir aspectos más avanzados como la escalabilidad, la seguridad y el mantenimiento del sistema.

El rol del backend en la Arquitectura de Software
Dentro de la arquitectura de software, el backend desempeña un rol critico al gestionar los procesos que no son visibles para el usuario, pero que son esenciales para el funcionamiento de una aplicación. Estos procesos incluyen:

- **Procesamiento de datos**: Recibir, procesar y devolver información, ya sea a usuarios finales o a sistemas externos, garantizando una comunicación eficiente entre los componentes del sistema.
- **Lógica de negocio**: Implementar reglas que aseguran que las operaciones cumplan con los requisitos funcionales de la aplicación.
- **Manejo de bases de datos**: Controlar el acceso a los datos, así como su almacenamiento, consulta y actualización en bases de datos relacionales o no relacionales.

El backend asegura que las aplicaciones funcionen de manera consistente y segura, incluso bajo una alta demanda de usuarios o solicitudes.

Tendencias de la Industria
En las últimas décadas, el desarrollo backend ha sido influenciado por diversas tendencias tecnológicas que han mejorado su desempeño y funcionalidad. Dentro de estas tendencias tecnologías se encuentran:

- **Microservicios**: Un enfoque arquitectónico que descompone aplicaciones en servicios pequeños e independientes, cada uno enfocado en una tarea específica [2].
- **Serverless computing**: Una forma de ejecución en la que el backend es gestionado por un proveedor en la nube, permitiendo a los desarrolladores centrarse en la lógica de negocio sin preocuparse por la infraestructura [3].
- **Contenedores**: Herramientas que permiten empaquetar aplicaciones y sus dependencias en un solo contenedor, facilitando el despliegue y asegurando consistencia entre entornos de desarrollo y producción [4].

Usos del Backend
El backend es indispensable en una amplia variedad de sistemas, incluyendo:

- Aplicaciones empresariales para la gestión de recursos humanos o logística.
- Plataformas de comercio electrónico, donde se maneja el inventario, pagos e interacción con usuarios.
- Sistemas de transacciones financieras, donde se requiere alta seguridad y procesamiento de datos en tiempo real.
- Aplicaciones móviles y web, que necesitan una comunicación constante con bases de datos y otros servicios externos.

Ventajas y Desventajas:

- **Ventajas**:
  - **Escalabilidad**: Permite que el sistema crezca para manejar mayores cargas trabajo.
  - **Separación de responsabilidades**: Facilita la organización del código al separar funciones de frontend y backend.
  - **Integración**: Posibilita la comunicación con servicios externos, como APIs de terceros o sistemas de autenticación.
- **Desventajas**:
  - **Complejidad**: Requiere diseñar sistemas robustos y bien estructurados, lo que puede incrementar la dificultad del desarrollo.
  - **Seguridad**: Es crucial implementar mecanismos solidos para proteger los datos sensibles y prevenir ataques cibernéticos.
  - **Manejo de errores**: La detección y resolución de errores en sistemas complejos puede ser un desafío importante.

2.2 Arquitectura de software

La arquitectura de software define como los componentes de un sistema se estructuran e interactúan entre sí. En este proyecto, se optó por realizar una arquitectura de capas basada en APIs, para que así se separe claramente las responsabilidades del sistema entre frontend, backend y capa de almacenamiento de datos.

Arquitecturas y patrones utilizados:

- **Arquitectura en Capas**: Esta arquitectura organiza el sistema en capas con distintas responsabilidades [5]. En este proyecto se utilizan tres capas:
  - **Presentación** (Frontend): Interfaz de usuario para móvil (Flutter) como web.
  - **Capa Lógica** (Backend): Procesamiento de documentos, gestión de usuario y gestión de tramites.
  - **Persistencia de datos:** Almacenamiento en bases de datos y almacenamiento de archivos.
- **Patrón MVC (Model-View-Controller)**: Implementado en Django, este patrón organiza las aplicaciones en modelos (datos), vistas (Interfaz) y controladores (procesamiento lógico) [6].
- **Estilo arquitectónico REST**: Las APIs REST se utilizan para permitir la comunicación eficiente entre las capas de presentación y lógica. Este estilo arquitectónico sigue principios como independencia cliente-servidor, sin estado y escalabilidad, asegurando la interacción robusta y modular entre los componentes [7].

Arquitecturas emergentes:

- **Microservicios**: Mientras que actualmente el proyecto está bajo una arquitectura monolítica con Django, la integración de microservicios podría considerarse en futuras iteraciones para modularizar diferentes servicios del sistema.

2.3 Tecnologías y frameworks utilizados

En este proyecto, se han seleccionado tecnologías y frameworks robustos con el objetivo de garantizar la eficiencia, seguridad y escalabilidad del sistema backend.

Lenguaje de programación

- **Python**: Seleccionado por la simplicidad, legibilidad y amplio ecosistema de librerías. Además, permite una rápida implementación de algoritmos de Machine Learning, utilizados en la comparación automatizada de documentos subidos al sistema.

Framework principal

- **Django**: Framework web de alto nivel basado en Python, que sigue el patrón Model-View-Template (MVT) [8]. Este framework proporciona mecanismos integrados para:
  - **Generación de APIs REST**: Gestión de usuarios, gestión de tramites, autenticación, manejo de documentos y notificaciones.
  - **Seguridad**: Incluye cifrado de contraseñas, protección contra CSRF, prevención de inyecciones SQL y medidas adicionales de seguridad en la autenticación.
  - **Escalabilidad y mantenimiento**: Soporte nativo para integración con bases de datos y desarrollo modular.

Base de datos

- **PostgreSQL**: Base de datos relacional robusta y flexible [9], utilizada para almacenar:
  - Información de usuarios y documentos.
  - Registro de pagos y transacciones.
  - Registro de trámites y cargas.

Autenticación y control de acceso

- **Firebase Authentication y autenticación local**
  - **Google Authentication**: Implementada para usuarios que prefieran un inicio de sesión rápido y seguro. Integrado con el servicio de Firebase, se utiliza el sistema de tokens JWT-like en conjunto a los tokens que genera Google, para tener un registro consistente entre tokens.
  - **Autenticación local basada en tokens (JWT-like)**: Implementada para los usuarios locales. Se utiliza un sistema de generación y validación de tokens para garantizar que cada acción este autenticada y protegida.

Manejo de documentos

- **AWS S3 Buckets**: Herramienta utilizada para el almacenamiento seguro y escalable de documentos relacionados con los trámites [11].

Pruebas y validación

- **Pruebas unitarias**
  - Uso del framework de pruebas de Django para verificar individualmente las funciones críticas, como autenticación, creación de usuario y procesamiento de documentos.
- **Pruebas de integración**
  - Validación de la comunicación entre frontend y backend a través de APIs REST.
- **Pruebas de carga**
  - Uso de herramientas como Locust [12] para analizar la capacidad del sistema bajo escenarios de alta concurrencia.

CI/CD y despliegue

- **GitLab CI/CD**: Configuración de pipelines para automatizar pruebas e integraciones.
- **Docker**: Uso de contenedores para garantizar entornos consistentes entre desarrollo y producción.

- **AWS**: Infraestructura de nube utilizada para alojar el backend, proporcionando escalabilidad y alta disponibilidad.

Seguridad

- **Cifrado de contraseñas**: Utilizando el sistema de hashing de Django (PBKDF2).
- **Tokenizacion**: Cada acción sensible requiere un token de autorización.
- **Validación y sanitización de datos**: Control estricto de entradas del usuario para prevenir inyecciones SQL y otros ataques como XSS.

Tecnologías emergentes

- **Machine Learning**: Integración de algoritmos para comparar documentos cargados por los usuarios con plantillas predefinidas, reduciendo errores humanos.

3 Diseño e Implementación

Este capítulo detalla el proceso de diseño y desarrollo del sistema propuesto para optimizar el proceso de retiro de contenedores. Se describe la arquitectura en capas utilizada, los componentes clave y las funcionalidades implementadas en cada módulo de backend. Además, se explica cómo se diseñaron e integraron las APIs REST para garantizar una comunicación eficiente entre la aplicación móvil, la plataforma web y el servidor backend.

También se incluye diagramas técnicos que ilustran la estructura del sistema, modelos de datos y flujos de interacción entre usuarios y el sistema. Por último, se aborda la estrategia de implementación, destacando el uso de contenedores Docker para el despliegue en AWS y las buenas prácticas adoptadas para garantizar la seguridad, escalabilidad y modularidad del sistema.

3.1 Diseño de Componentes

3.1.1 Arquitectura y estructura de los componentes.

Para que el desarrollo del proyecto se realice de manera eficiente y ordenada, se realizó una selección de arquitectura y componentes que serán fundamentales para que la arquitectura sea aplicada de manera correcta. Por esta razón, se describirá que arquitectura se eligió, porque y cómo funciona, también se describirán los componentes que conformar esta arquitectura, como se verá el despliegue y las entidades y relaciones de datos que componen al sistema.

Elección de la arquitectura:
Se opto por una arquitectura en capas, debido a su capacidad para separar las responsabilidades del sistema, lo que facilito el mantenimiento, la escalabilidad y el desarrollo paralelo. La solución consta de tres capas principales:

- **Capa de presentación (Frontend)**: Implementada en Flutter, esta capa incluye:
  - Una aplicación móvil para el agente tramitador y el conductor.
  - Una interfaz web para el agente de visado.
- **Capa de reglas de negocio (Backend)**: Construida en Django, esta capa se encarga de:
  - Procesamiento de la lógica de negocio.
  - Implementación de APIs REST para comunicar el frontend con el backend.
  - Autenticación de usuarios mediante Firebase Authentication y tokens locales.
- **Capa de datos**:
  - Bases de datos relacionales con PostgreSQL para gestionar información estructurada (usuarios, roles, pagos, registros).
  - Almacenamiento en AWS S3.

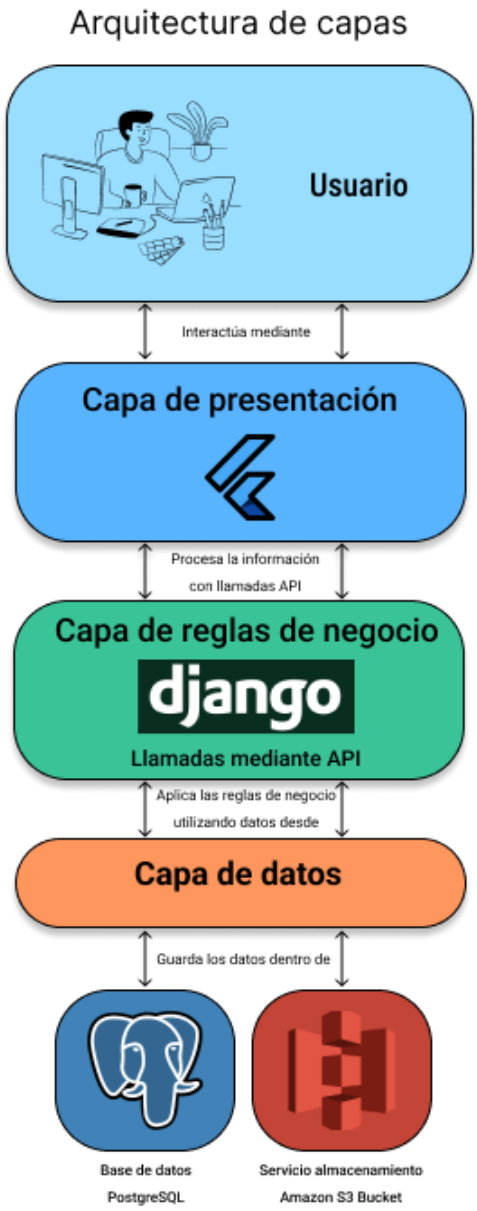

Figura 3–1. Diagrama de arquitectura de capas

Fuente: Elaboración propia

Utilizando esta lógica de arquitectura, se determinó que una buena arquitectura para este proyecto seria la siguiente, la cual fue representada mediante un diagrama de despliegue, el cual describe de qué manera es organizada la estructura de los elementos de software y hardware, ayudando a tener una mejor idea de la topología que se utiliza en el proyecto.

El siguiente diagrama describe la arquitectura en capas implementada en este proyecto. En la capa de presentación, se utilizan aplicaciones móviles y web desarrolladas con Flutter, que interactúan con el backend mediante APIs REST seguras. El backend, el cual es alojado en una instancia EC2 de AWS y ejecutado en contenedores Docker, utiliza Django para manejar la lógica de negocio, mientras que se utiliza Celery para administrar tareas intensivas como el reconocimiento óptico de caracteres (OCR)

mediante PyTesseract. La base de datos PostgreSQL almacena la información estructurada, mientras que los documentos se alojan en Amazon S3 para asegurar un almacenamiento escalable y confiable. Adicionalmente, Firebase Authentication proporciona mecanismos adicionales de autenticación, esto complementado a la autenticación basada en roles de conductor, tramitador y agente de visado, optimizando el flujo de trabajo.

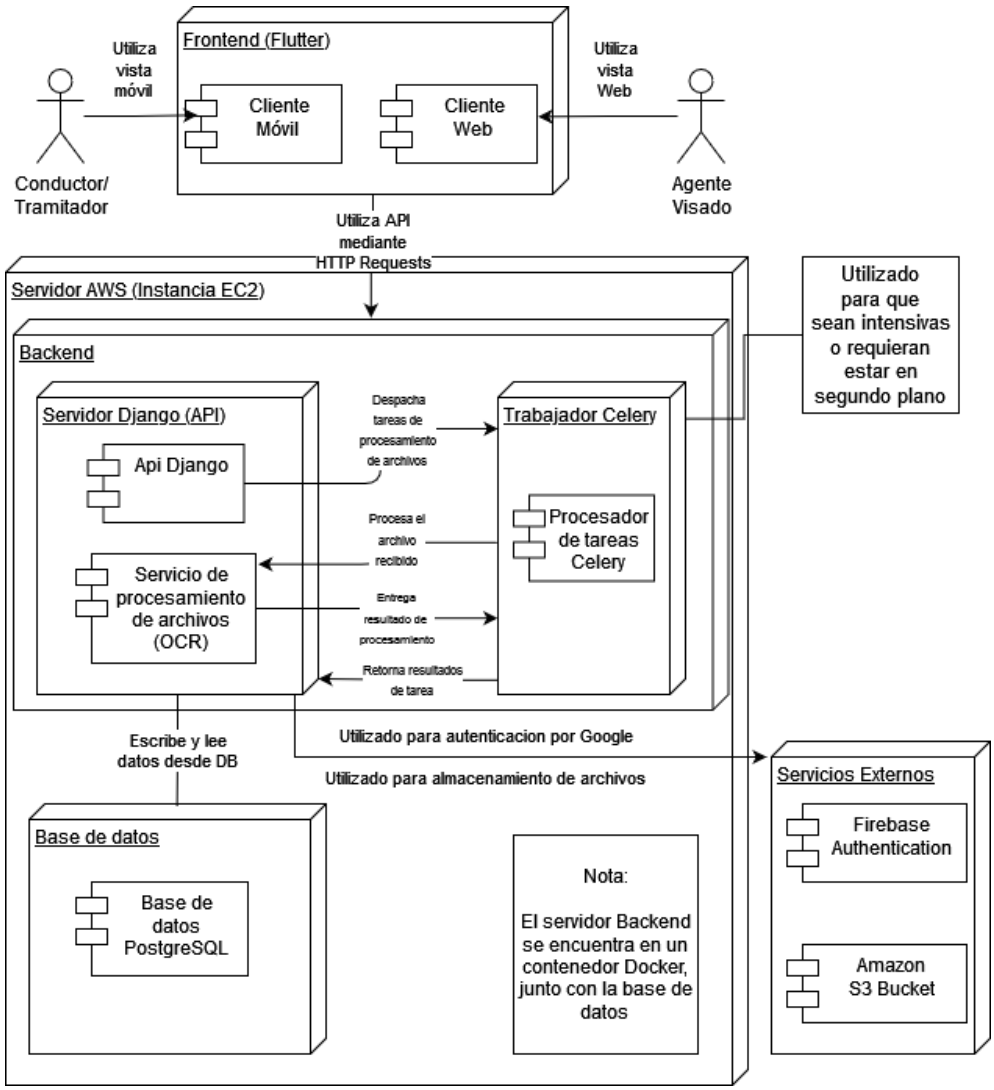

Figura 3–2. Diagrama de despliegue de la arquitectura utilizada

Fuente: Elaboración propia

Para comprender las interacciones dentro del sistema, se debe describir las entidades utilizadas por el sistema, debido a que es fundamental entender las relaciones que tienen para poder entender cómo funcionan los datos dentro de las interacciones.

A continuación, se presentarán dos figuras. La primera corresponde a los modelos de datos relacionales utilizados en el sistema, especificando que tipo de datos contiene cada entidad (representada como una tabla) y como se relacionan entre si mediante claves foráneas. Este modelo se enfoca en la organización y almacenamiento de los datos.

La segunda figura es el diagrama de clases, que representa el diseño del sistema desde una perspectiva orientada a objetos. Este diagrama describe cómo se relaciona varios datos y funciones dentro del sistema, no solo revelando la relación entre los datos presentes en la base de datos, sino también la

forma en que actúan las entidades y clases, a través de sus atributos y métodos. A diferencia de un modelo relacional, un diagrama de clases se centra en la lógica de la aplicación y en cómo interactúan las entidades para implementar la lógica de negocio definida en los requisitos del sistema.

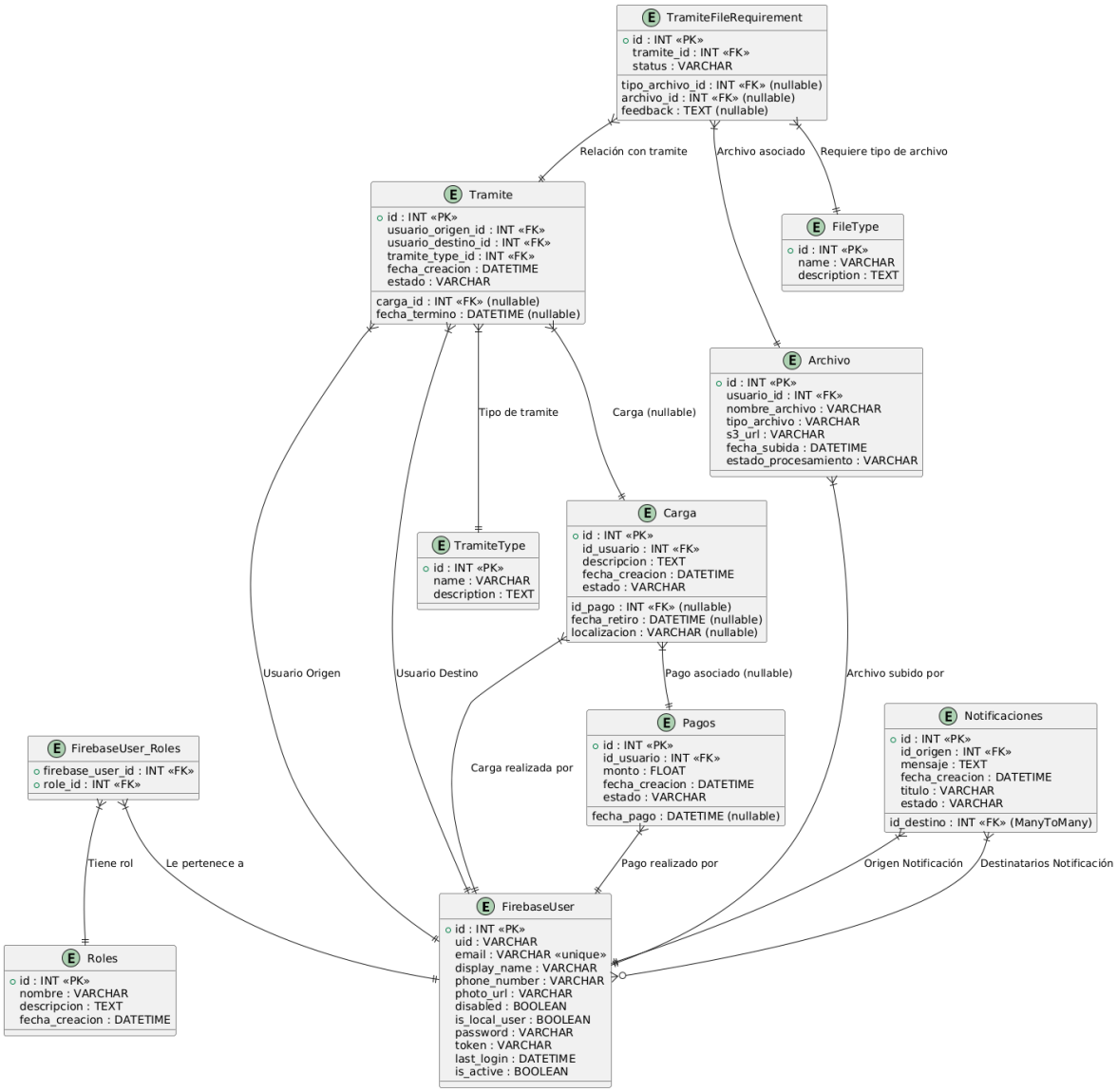

Figura 3–3. Modelo relacional de datos

Fuente: Elaboración propia

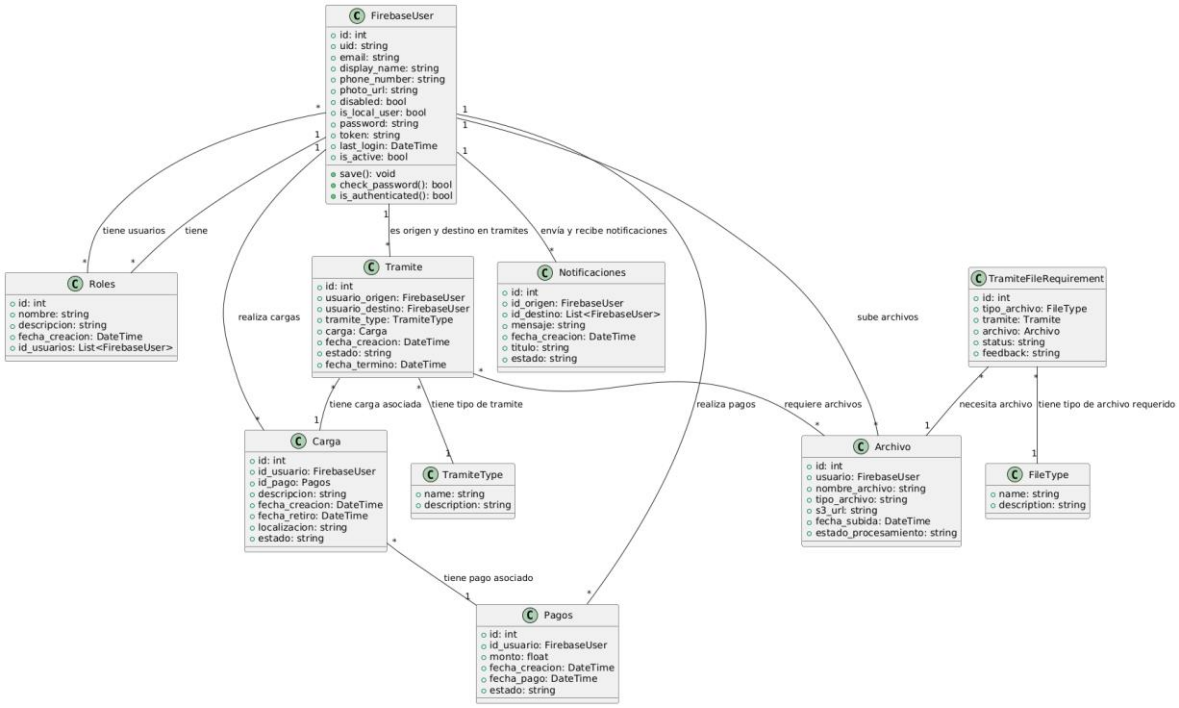

Figura 3–4. Diagrama de clases

Fuente: Elaboración propia

En términos de componentes de software, el sistema está compuesto por distintos módulos, los cuales son fundamentales para que la solución propuesta sea una eficiente y escalable, esta estructura logra realizar la separación de responsabilidades de manera modular, lo que permite la escalabilidad y eficiencia antes mencionada.

Módulo de autenticación y control de acceso:
Este módulo se encarga del proceso de autenticación y control de acceso basado en roles y tokens de acceso, donde se encuentran los siguientes roles para los usuarios:

- Conductor.
- Tramitador.
- Visado.

Esta división de roles facilita la gestión eficiente de permisos y accesos, lo que incluye que tipo de información y acciones requiere cada actor involucrado en el proceso del retiro de la carga, también agregando una capa de seguridad al sistema debido a que se puede exigir ciertos roles y permisos para acciones que traten con información sensible que debería ser vista y manipulada por cierto grupo de personas.

A continuación, se muestra el diagrama de componentes asociado a este módulo, que trata sobre cómo se ve la autenticación de usuario y los componentes que son utilizados para completar este proceso.

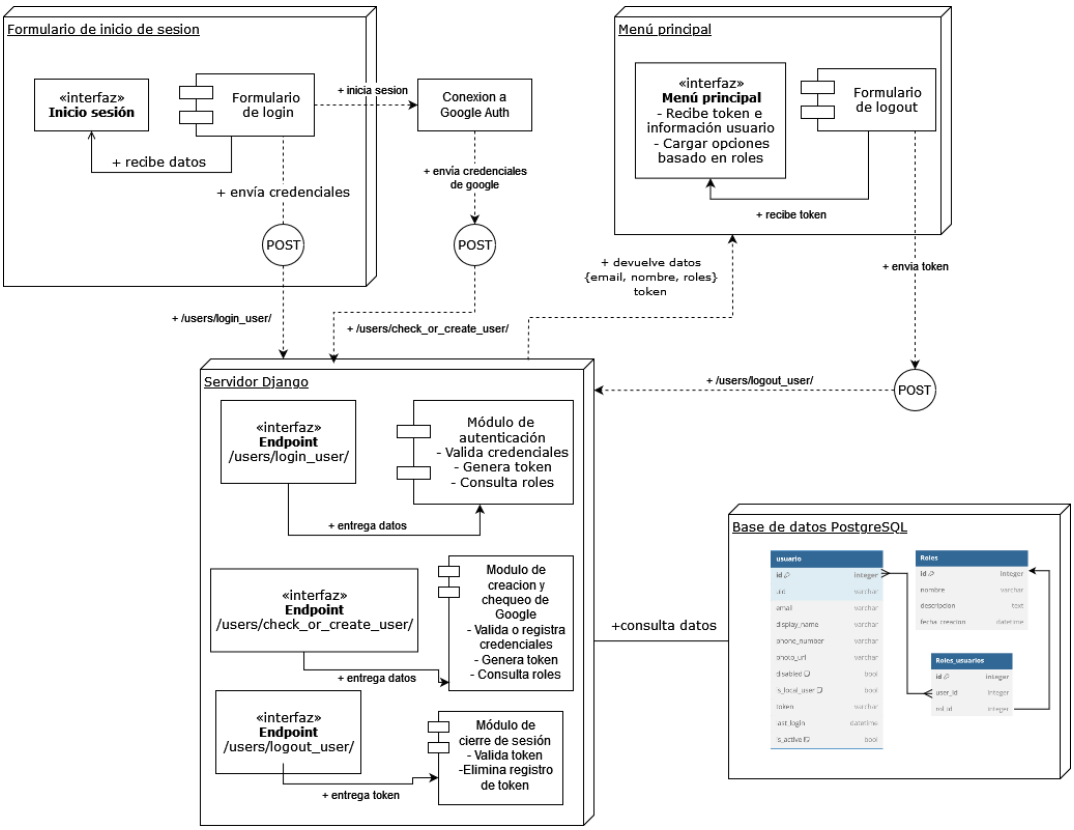

Figura 3–5. Diagrama de componentes del módulo de autenticación

Fuente: Elaboración propia

Dentro de este diagrama, se debe recalcar que, al permitirse dos métodos de autenticación, estos siendo vía Google y vía credenciales locales, se llaman a dos componentes distintos del servidor, es decir, el backend, esto nos permite tener mayor control sobre el tipo de autenticación que prefieren los usuarios y a la vez ofreciendo a los usuarios que tengan problemas con la creación de cuentas locales, autenticarse de una manera más simple. También se hace uso del token asignado al usuario para saber que interfaz mostrarle dependiendo de su rol.

Módulo de gestión de usuarios:
Dentro de este módulo, se encuentran los componentes de gestión de usuario, siendo específicamente:

- Registro de usuario:
  - Formulario de creación que envía los datos al endpoint `/users/check_or_create_user/`. Dentro del endpoint, se valida la información proporcionada, registra al usuario en la tabla `usuario` y asigna roles en la tabla `roles_usuarios`.
- Modificación de información de usuario:
  - Mediante una solicitud PUT, se permite actualizar ciertos datos de un usuario, permitiendo corregir errores o actualizar datos, como número telefónico o rol asignado.
- Activación de cuenta de usuario:
  - Las cuentas deshabilitadas pueden reactivarse enviando una solicitud PUT al endpoint `/users/enable_user/`, lo que actualiza el estado de la cuenta en la base de datos.
- Desactivación de cuenta de usuario:
  - Cuando se requiere restringir el acceso a un usuario, se envía una solicitud PUT al endpoint `/users/disable_user/`, marcando la cuenta como deshabilitada.

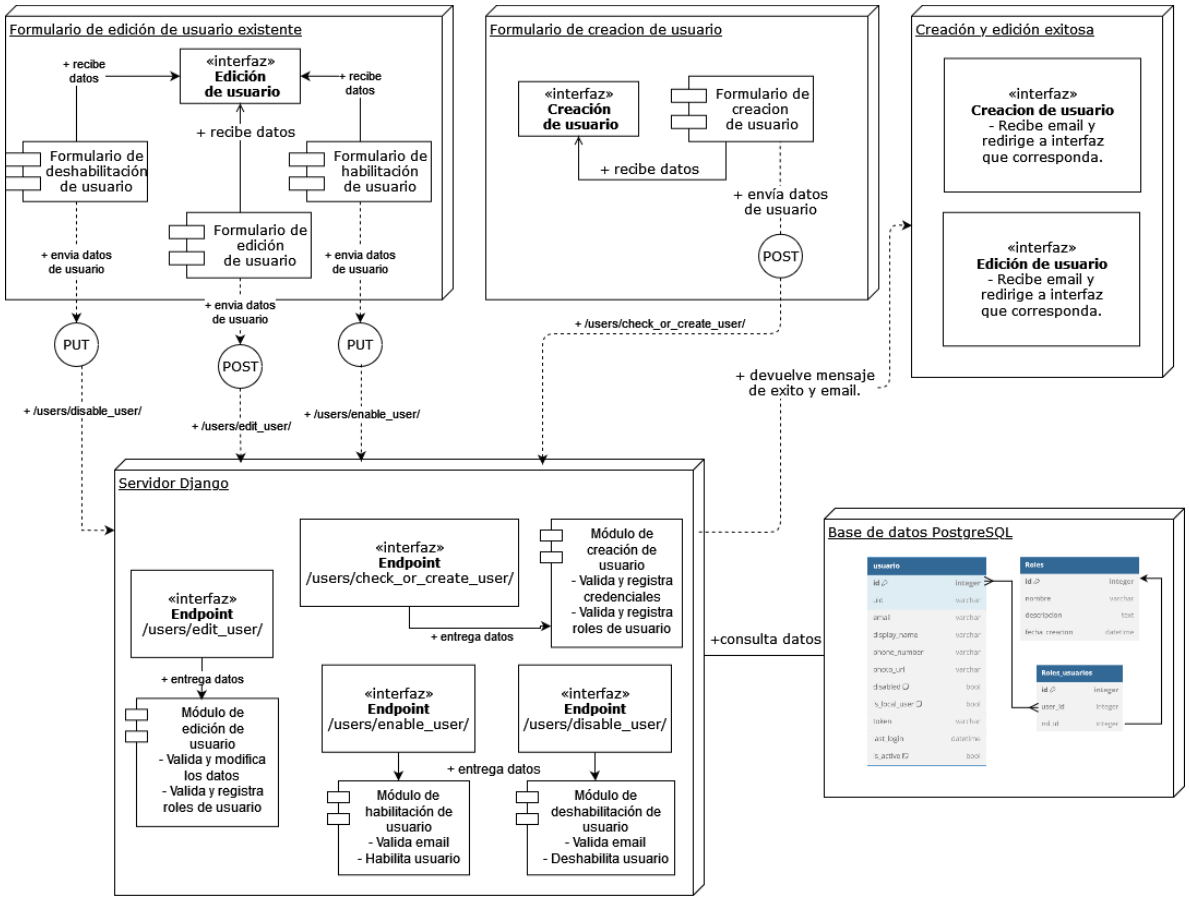

Figura 3–6. Diagrama de componentes del módulo de usuarios

Fuentes: Elaboración propia

Módulo de procesamiento de documentos:
El módulo de procesamiento de documentos trata sobre el uso de la implementación de algoritmos de machine learning, que utilizan el procesamiento óptico de caracteres (OCR) para poder comprobar que la información de los documentos entregados sean correctos y no hayan sido manipulados previo a la entrega, para esto se utilizan plantillas que ayudan a la comprobación de la información, esto no reemplaza la labor manual de revisión de documentos, si no, busca reducir significativamente el tiempo requerido para la validación de documentos mediante la automatización de verificaciones preliminares, dándoles una idea previa a la revisión si tiene errores detectados por el algoritmo.

Dentro del diagrama se encuentran tres componentes que funcionan para que su uso sea eficiente y no ocasione problemas con el resto del sistema, estos siendo:

- **Celery:**
  - Biblioteca de Python que permite el manejo de tareas asíncronas, evitando sobrecargas en el servidor principal
- **OCR (PyTesseract):**
  - Tecnología basada en el procesamiento óptico de caracteres, utilizada para extraer el texto de los documentos.
- **Plantillas de validación:**
  - Plantillas guardadas en el almacenamiento de S3, utilizada para verificar la coherencia de los datos extraídos.

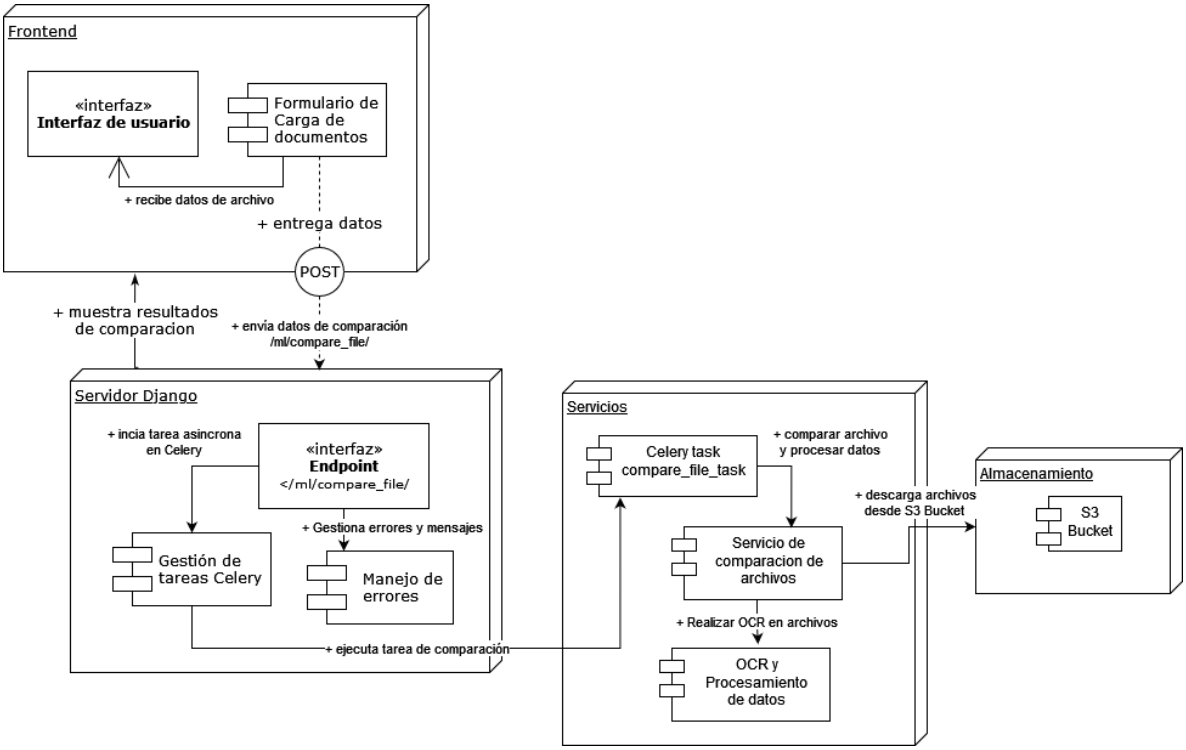

Figura 3–7. Diagrama de componentes del módulo de procesamiento de documentos

Fuente: Elaboración propia

Dentro del módulo, se consulta por la comprobación de un documento. Esta consulta es enviada al endpoint `/ml/compare_file/` en el servidor Django, donde se inicia una tarea asíncrona con Celery. La tarea descarga el archivo desde S3 Bucket, aplica el OCR y compara los datos extraídos con las plantillas prediseñadas. Los resultados de la validación son enviados de vuelta al cliente.

Módulo de gestión de cargas:
En cuanto al módulo de gestión de cargas, permite registrar, modificar y realizar seguimiento de las cargas dentro del sistema. Este módulo está diseñado para garantizar un manejo eficiente y seguro de la información asociada a las cargas, integrando validaciones basada en roles y autorizaciones. Las funcionalidades principales son las siguientes:

- Creación de cargas:
  - Los usuarios que tengan el rol de visado pueden registrar nuevas cargas mediante el formulario en la vista web. Los datos enviados al endpoint `/cargas/create_carga/` son validados por el backend, que comprueba los permisos del usuario antes de registrar la carga en la base de datos PostgreSQL.
- Edición de cargas:
  - Este módulo permite modificar los datos de una carga existente mediante el endpoint `/cargas/edit_carga/`. Se realizan validaciones para garantizar que solo usuarios que tengan el rol de visado puedan efectuar cambios.
- Consulta de cargas:
  - Los usuarios pueden consultar el estado de las cargas asignadas, pendientes o retiradas a través de formularios en la vista móvil y web. El backend procesa las solicitudes enviadas a los endpoints `/cargas/check_cargas_pendientes/` y `/cargas/check_cargas/retiradas/`, retornando la información correspondiente al frontend.

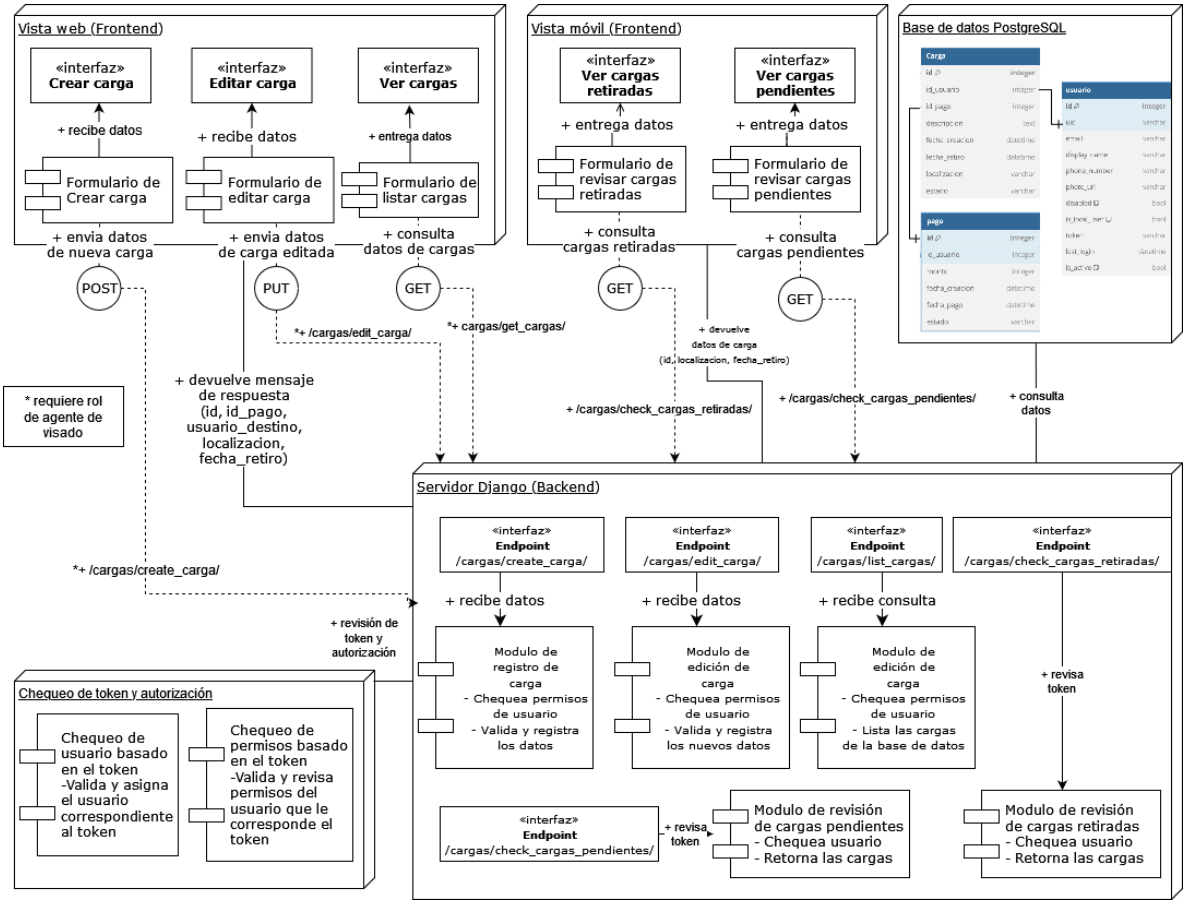

Figura 3–8. Diagrama de componente del módulo de cargas

Fuente: Elaboración propia

Este módulo asegura la gestión eficiente y segura de las cargas, permitiendo a los actores involucrados consultar y actualizar información en tiempo real, asegurando a la vez la seguridad de los datos utilizando los tokens de usuario, tanto para filtrar las cargas que tenga asignadas a su persona y permitiendo la manipulación de los datos solo por los usuarios autorizados por sus roles y permisos.

Módulo de notificaciones:
El módulo de notificaciones está diseñado para la gestionar la creación, consulta y actualización del estado de notificaciones en tiempo real. Este sistema permite enviar alertas a los usuarios que requieran una actualización por parte de los agentes de visado, asegurando que los actores involucrados estén informados sobre eventos relevantes en el sistema. En cuanto al funcionamiento de este módulo, tiene una serie de funciones principales que serán descritas a continuación.

- Creación de notificaciones:
  - Los agentes de visado pueden generar nuevas notificaciones a través del formulario de creación en la vista web. Los datos son enviados al endpoint `/notificaciones/create_notification/`, donde el backend valida los permisos que tiene el usuario que intento crear una nueva notificación, registra la notificación en la base de datos si es exitosa la validación y devuelve un mensaje de confirmación.
- Consulta de notificaciones:
  - Los usuarios pueden revisar notificaciones pendientes a través del formulario en la vista móvil. Las solicitudes GET al endpoint `/notificaciones/user_notifications/` recuperan una lista de notificaciones asociadas al usuario.

- Actualización del estado de la notificación:
    - Los usuarios pueden marcar una notificación como leída para que así se mantenga una constancia del estado de la notificación enviada. Esto se hace mediante una solicitud PUT al endpoint `/notificaciones/check_notificacion/<id_notificacion>/`, donde se actualiza el estado de la notificación en la base de datos, indicando que ya fue vista.

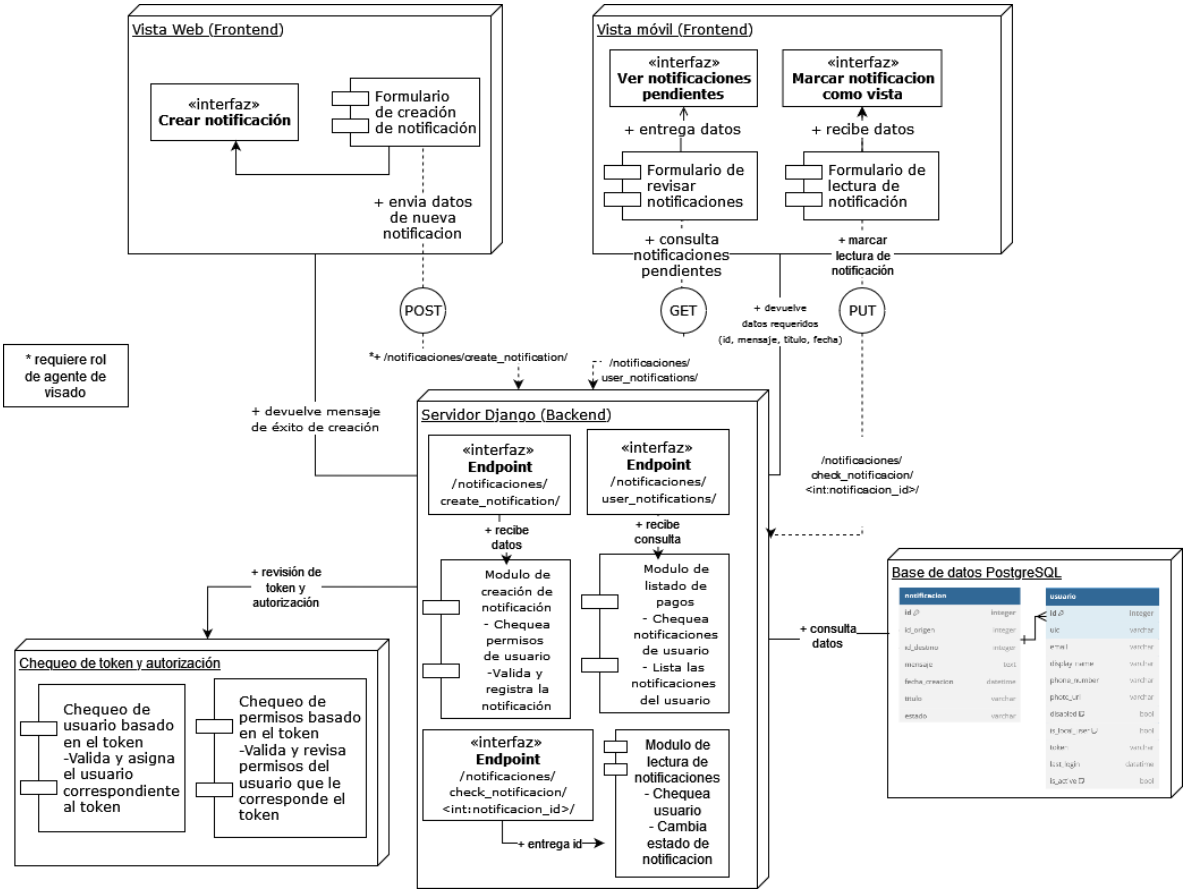

Figura 3–9. Diagrama de componente de módulo de notificaciones

Fuente: Elaboración propia

El módulo centraliza la gestión de alertas, mejorando la comunicación entre los actores involucrados. Esto no solo optimiza los flujos de trabajo, sino que también reduce los tiempos de espera e incrementa la productividad al proporcionar información relevante de manera inmediata.

Módulo de gestión de tramites:
Este módulo tiene como objetivo centralizar y automatizar la asignación de cargas, la creación de trámites y el seguimiento de su estado. El módulo asegura que cada actor involucrado en el proceso pueda interactuar con los tramites de manera eficiente y segura, utilizando roles y permisos claramente definidos. A continuación, se detallan las funcionalidades principales que se utilizan en este proceso.

- Creación de tramites:
    - Los agentes de visado pueden registrar nuevos tramites utilizando el formulario de creación en la vista web. Esta acción envía una solicitud POST al endpoint `/tramites/create_tramite/`, donde el backend valida los datos ingresados y los permisos del usuario antes de realizar el registro del trámite en la base de datos.

- Consulta de tramites:
  - Los usuarios pueden consultar tramites asociados a ellos a través de vistas especificas en móvil y web, es decir, todos los usuarios que tengan algún trámite asociado a él, esto se realiza mediante formularios de consulta, que envían solicitudes GET a endpoints como `/tramites/check_tramite/` y `/tramites/check_tramites_user/`, permitiendo revisar detalles del trámite, archivos relacionados a este y el estado del trámite.
- Gestión de archivos relacionados:
  - Los archivos asociados a un trámite pueden ser consultados y gestionados por los usuarios involucrados mediante el endpoint `/tramites/check_tramite_files/`.
- Seguimiento del estado de los tramites:
  - Este módulo permite revisar y actualizar el estado de los tramites, asegurando que los usuarios estén informados de su progreso.

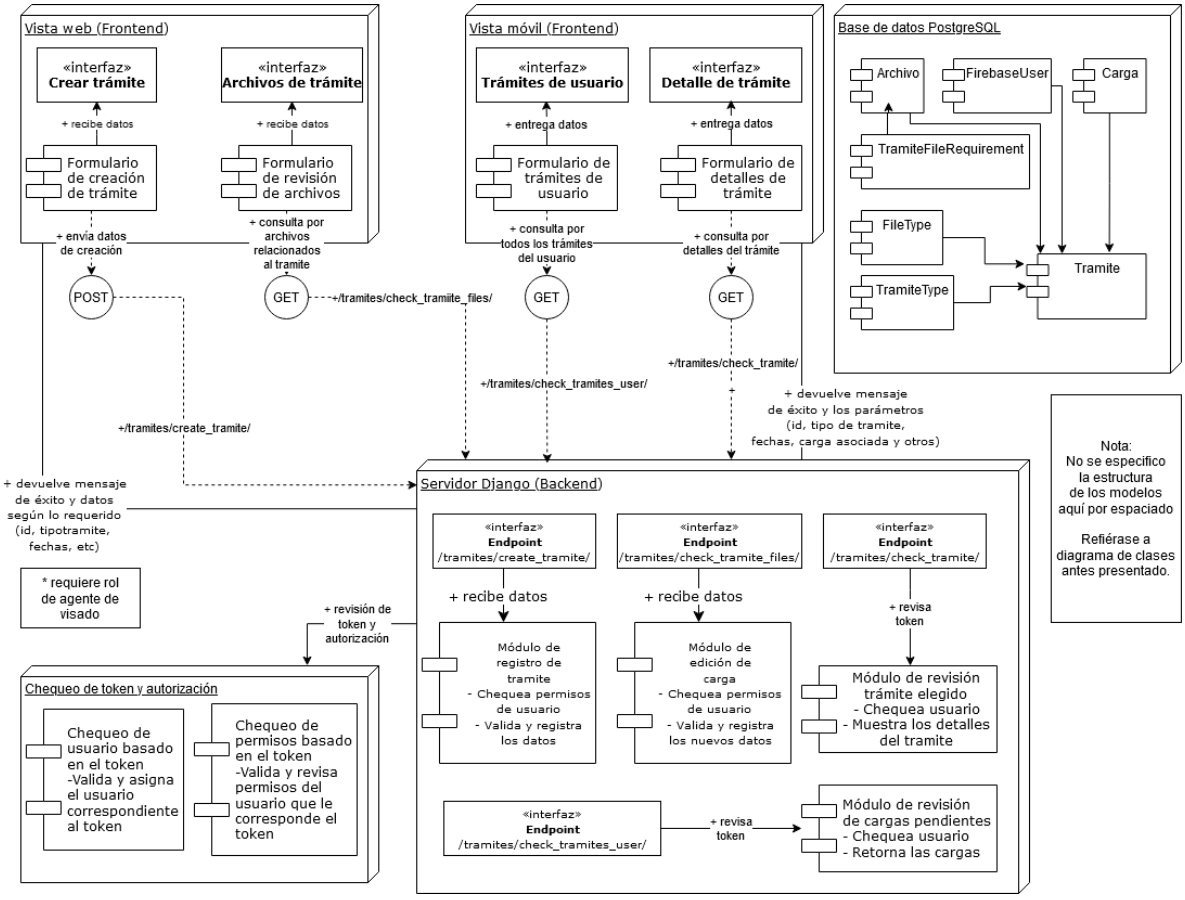

Figura 3–10. Diagrama de componentes del módulo de tramites

Fuente: Elaboración propia

El módulo de gestión de tramites proporciona un marco estructurado para manejar la creación, consulta y actualización de los tramites y los archivos que se vinculen a él, mejorando la colaboración entre los actores involucrados. Además, al integrar la validación basada en roles y tokens, garantiza la seguridad e integridad de los datos en cada solicitud.

Módulo de integración de pagos:
Este módulo trata sobre la integración de pagos, que es responsable de gestionar las solicitudes de creación y seguimientos de pagos realizados por los usuarios. Aunque no se implementaron

plataformas externas de pago, este módulo es fundamental tenerlo integrado ya que centraliza el manejo de registros de pagos en el sistema, asegurando una gestión eficiente y segura al momento de que se implemente la funcionalidad. Se describirán las funcionalidades que se requieren para que el módulo funcione de manera correcta y eficiente.

- Creación de pagos:
  - Los usuarios autorizados pueden registrar pagos asociados a un usuario a través de la interfaz web. Los datos del pago (monto y fecha de vencimiento) son enviados al endpoint `/pagos/create_pago/`, donde son validados y registrados en la base de datos.
- Consulta de pagos:
  - Los usuarios pueden consultar los pagos asociados a ellos mediante la interfaz móvil. Las solicitudes GET a los endpoints `/pagos/get_user_pagos/` y `/pagos/list_pagos/` permiten acceder a información como el monto, el estado y la fecha de creación de cada pago.
- Revisión del estado de pagos:
  - Permite a los usuarios y administradores verificar el estado actual de los pagos registrados, como “pendiente”, “completado”, o “fallido”, asegurando que el proceso tenga un nivel de transparencia adecuado.

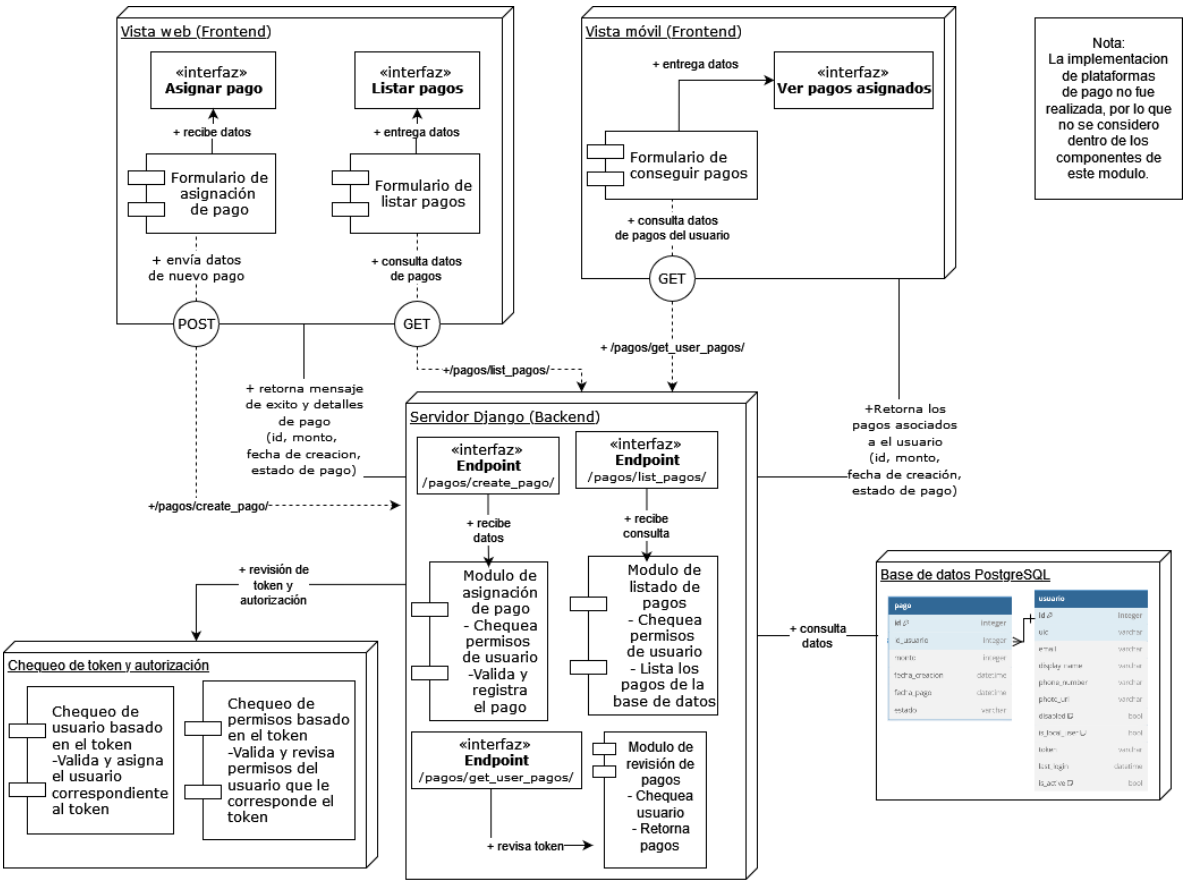

Figura 3–11. Diagrama de componentes del módulo de pago

Fuente: Elaboración propia

Aunque el módulo gestiona los registros de pago de manera eficiente, no se incluyó la integración con pasarelas externas. Esto puede ser considerado para futuras iteraciones, asegurando que los usuarios puedan realizar pagos directamente desde la plataforma.

Módulo de subida de archivos:
El módulo de subida de archivos permite a los usuarios cargar documentos necesarios para completar los tramites en el sistema. Este módulo asegura la gestión eficiente y segura de los archivos, integrándose con los servicios de almacenamiento en la nube (Amazon S3 Bucket) para garantizar la escalabilidad y accesibilidad. A continuación, están las funcionalidades involucradas en este módulo:

- Subida de archivo:
  - Los usuarios pueden cargar archivos para sus trámites a través del formulario correspondiente en la interfaz móvil o web. Los archivos son enviados al endpoint `/files/upload_file/`, donde el backend los valida antes de almacenarlos en un Bucket de Amazon S3.
- Consulta de archivos pendientes:
  - Los usuarios pueden consultar los documentos asociados a sus trámites mediante el endpoint `/files/get_user_assigned_files/`. Este proporciona una lista detallada de los archivos requeridos, incluyendo su estado (pendiente, aprobado, rechazado).
- Aprobación y rechazo de archivos:
  - Los agentes de visado pueden revisar los documentos cargados y aprobar o rechazar aquellos que no cumplan con los requisitos. Estas acciones se gestionan mediante los endpoints `/files/approve_file/<id_archivo>/` y `/files/reject_file/<id_archivo>`.

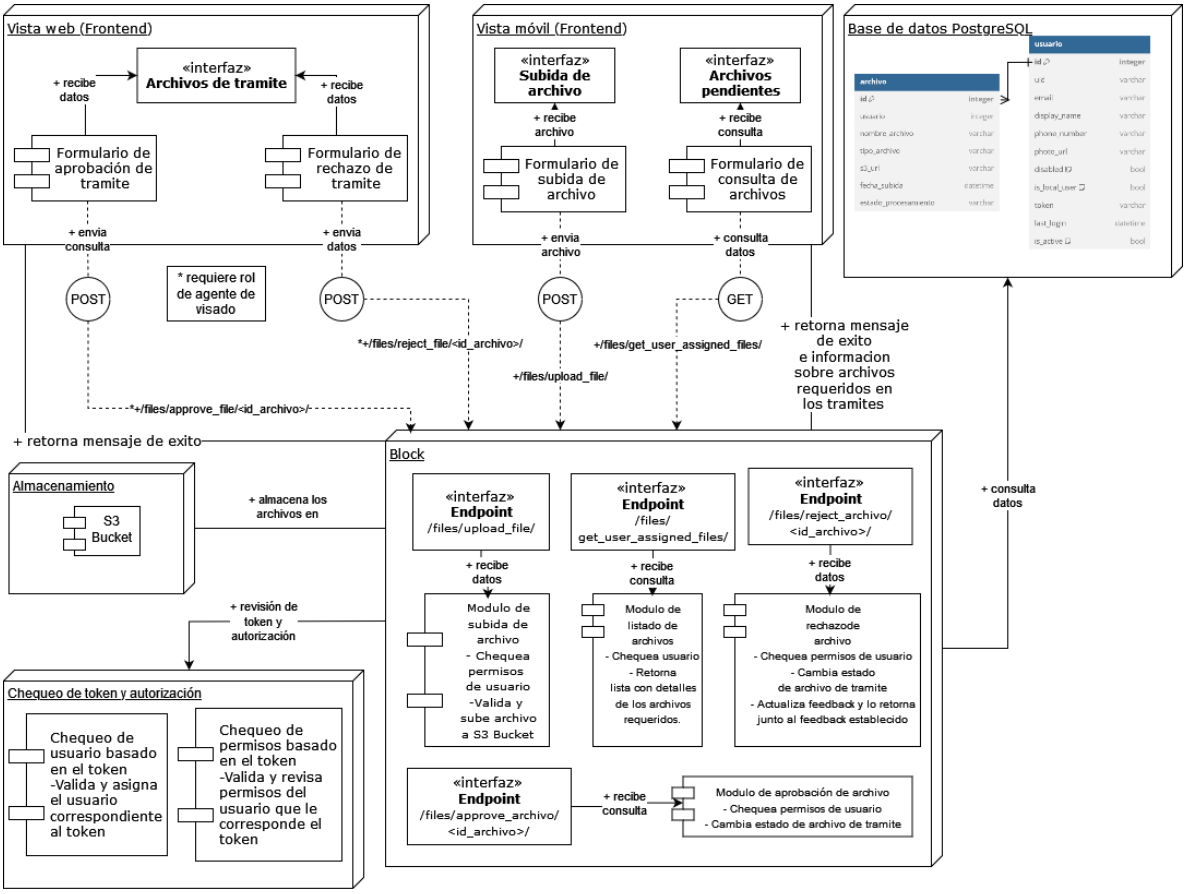

Figura 3–12. Diagrama de componentes del módulo de subida de archivos

Fuente: Elaboración propia

Este módulo de subida de archivos automatiza y asegura el manejo de documentos requeridos para tramites, mejorando la eficiencia del proceso y reduciendo errores manuales. Su diseño modular

permite integrar funcionalidades adicionales, como validaciones automáticas basadas en OCR, en futuras iteraciones.

Estrategia de implementación:
La implementación del sistema se llevará a cabo mediante un enfoque iterativo y modular que asegura la validación y funcionalidad de cada componente antes de su integración a los ambientes de producción. Esto incluye el desarrollo basado en APIs REST y el uso de tecnologías modernas como Docker y AWS para garantizar un entorno escalable y consistente.

A continuación, se explicará los tres aspectos mencionados, junto con sus detalles técnicos y beneficios que traen al proyecto.

Desarrollo iterativo:
El enfoque iterativo garantiza que cada módulo del sistema se desarrolle, pruebe y valide de manera independiente antes de su integración en el sistema completo.

Dentro de los ciclos iterativos, se encuentra las siguientes fases de ciclo:

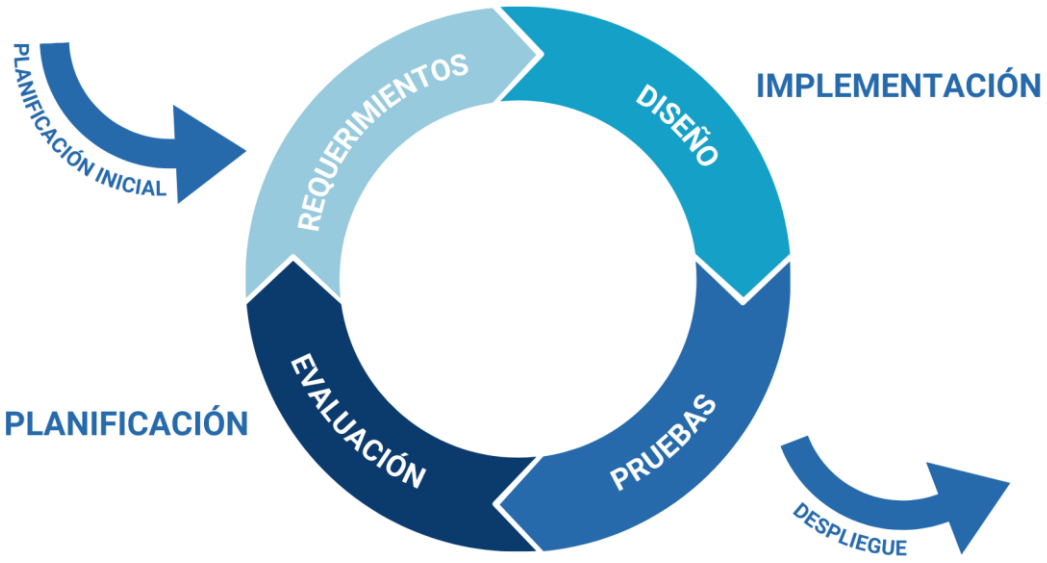

Figura 3–13. Ilustración del desarrollo iterativo

Fuente: Elaboración propia

Las fases del ciclo se definen como:

1. **Planeación:**
   - Se investiga que se necesita por parte de los usuarios finales y clientes.
2. **Requerimientos:**
   - Se definen los requerimientos de software que necesite los usuarios finales y clientes.
3. **Diseño:**
   - Se establece la arquitectura de software donde se realizará la implementación, tomando en cuenta el diseño general del sistema, servicios requeridos y otros.
4. **Implementación:**
   - Se desarrolla el código para implementar la lógica de negocio del requerimiento, siguiendo el diseño antes establecido

5. **Pruebas:**
    - El código es probado por potenciales bugs y errores, se realizan pruebas unitarias y de integración para verificar el funcionamiento correcto.
6. **Evaluación:**
    - El equipo de proyecto se junta para evaluar los avances realizados en la iteración, indicando que se pudo lograr y que falto por desarrollar.

Este enfoque de desarrollo nos permite una menor cantidad de errores al momento de realizar la integración, nos facilita que el sistema se mantenga modular y permite realizar ajustes al sistema basado en los comentarios del equipo y cliente en cada entrega.

Uso de APIs REST:
La comunicación entre el frontend y backend se realiza mediante APIs REST, lo que asegura una arquitectura modular y desacoplada.

En cuanto al diseño técnico de las APIs del sistema, se encuentra lo siguiente:

- Cada módulo del backend expone endpoints específicos que representan funcionalidades.
- Los endpoints siguen estándares RESTful, asegurando la uniformidad y escalabilidad.
- Ejemplos de estructura de endpoints en el sistema:
    - Autenticación: `POST /users/login_user/`
    - Trámites: `GET /tramites/check_tramite_files/`
    - Pagos: `POST /pagos/create_pago/`
- El frontend se comunica mediante solicitudes HTTP (GET, POST, PUT) a los endpoints correspondientes.
- Los datos se transfieren en formato JSON, facilitando el procesamiento tanto en el backend como en el frontend.
- En términos de pruebas realizadas para las APIs, se utilizaron las siguientes pruebas:
    - Pruebas de integración utilizando Postman.
    - Pruebas utilizando el frontend.

El uso de APIs REST permiten la compatibilidad por múltiples clientes (e.g., móvil y web) sin hacer modificaciones al backend y permite la reusabilidad de las APIs en otros sistemas.

Despliegue y entorno:
El sistema es desplegado en AWS, utilizando contenedores Docker para garantizar la consistencia del entorno y escalabilidad del sistema.

Como se vio en el diagrama de despliegue (Figura 3–2), se utilizó una instancia de AWS EC2 para alojar el contenedor de Docker, en el cual, cada módulo del sistema se ejecuta en contenedores Docker independientes, garantizando que las configuraciones básicas del entorno de desarrollo sean idénticas al de producción, eliminando problemas de configuración.

La instancia EC2 nos permite tener flexibilidad y escalabilidad del sistema, en caso de que se requiera mayor cantidad de recursos o se necesite implementar otros servicios.

También se utiliza Amazon S3, lo que nos permite tener un almacenamiento seguro de los documentos y archivos subido por los usuarios.

Este despliegue nos permite tener una consistencia con los sistemas debido al uso de contenedores Docker, haciendo que los requerimientos de hardware se mantengan consistentes, y al utilizar las instancias de AWS EC2, se pueden aumentar los recursos de hardware bajo demanda y a la vez monitorear estos cambios de carga utilizando herramientas que provee AWS (e.g., CloudWatch).

3.1.2 Uso de buenas prácticas y patrones de diseño.

En este proyecto, se utilizaron dos patrones clave para el diseño del sistema:

- **Patrón MVC (Model-View-Controller).**
- **Estilo arquitectónico REST.**

Aunque Django utiliza internamente una variación del patrón MVC [7] conocida como MVT (Model-View-Template), en este proyecto se adoptó el patrón MVC, ya que el backend funciona exclusivamente como API y el frontend se maneja de forma externa. A continuación, se explica cómo se estructuran ambos patrones y la razón detrás de la elección.

Patrón MVC (Model-View-Controller):
El patrón MVC organiza la lógica del sistema en tres componentes principales:

- **Modelos**: Gestionan la lógica de negocio y definen los datos que se encuentran en la base de datos (e.g., Usuario).
- **Vistas**: Se encargan del diseño y presentación de los datos, es decir, como los datos deben ser representados, esto comúnmente es mediante vistas HTML o respuestas JSON.
- **Controladores**: Procesas las solicitudes HTTP y enruta a los modelos y vistas correspondientes. (e.g., APIs REST).

Patrón MVT (Model-View-Template):
Utilizado únicamente por el framework de Django, separa la lógica del sistema en:

- **Modelos:** Al igual que el patrón MVC, gestiona la lógica de negocio y define los datos que se encuentra en la base de datos.
- **Vistas**: Se encarga de recibir las solicitudes mediante entradas del usuario, las procesa y seleccionan un **template** para renderizar una respuesta.
- **Template**: Se encarga de generar la salida visual (HTML) que el usuario verá. Este enfoque combina el backend y el frontend en un mismo Framework.

En este proyecto, se utilizó el patrón MVC debido a las siguientes razones:

1. **Frontend separado:**
    - El frontend está desarrollado en Flutter y no utiliza el sistema de plantillas de Django, por lo que la funcionalidad del backend se limita a proveer datos a través de APIs REST
    - Por este motivo, se permite una separación entre el cliente (frontend) y el servidor (backend), alineándose con los principios de REST.
2. **Foco en APIs REST:**
    - Las vistas en Django se utilizan para procesar la lógica de negocio y retornar respuestas en formato JSON, en lugar de renderizar templates HTML.
    - Esto simplifica la implementación y hace que el backend sea más modular y reutilizable.
3. **Escalabilidad y mantenibilidad:**
    - Debido a que se sigue el patrón MVC, el backend puede desarrollarse y escalarse de forma independiente del frontend, facilitando futuras actualizaciones y mejoras.

En resumen, Django utiliza internamente el patrón MVT para combinar la lógica de negocio y el renderizado de frontend, pero en este proyecto se adoptó el patrón MVC para aprovechar la modularidad y flexibilidad que ofrece, permitiendo construir un backend desacoplado y optimizado para trabajar como API REST y servir a un frontend externo.

Estilo arquitectónico REST:
En este proyecto, el estilo arquitectónico REST se utilizó para la comunicación entre el frontend y el backend. Este enfoque ofrece varias características que optimizan la interacción entre ambos sistemas.

1. **Independencia Cliente-Servidor:**
    - El frontend y backend esta desacoplado, permitiendo que ambos se desarrollen, escalen y se mantengan de forma independiente.
2. **Sin estado:**
    - Cada solicitud incluye toda la información necesaria para ser procesada, eliminando dependencias entre solicitudes, aportando a la facilidad de la escalabilidad del sistema.
3. **Componentes clave de REST:**
    - **Recurso:** Todo se trata como recurso, identificado por una URL única (e.g., /usuarios/, /tramites/).
    - **Métodos HTTP:** Se utilizan métodos estándar como GET, POST, PUT para operar sobre los recursos
    - **Formato de Datos:** JSON es el formato principal utilizado para la transferencia de datos entre cliente y servidor.
4. **Ventajas técnicas:**
    - **Escalabilidad:** Al ser sin estado, facilita escalar el backend.
    - **Interoperabilidad:** REST es compatible con la mayoría de frameworks y lenguajes, facilitando integraciones.
    - **Pruebas de integración sencillas:** Las APIs pueden probarse fácilmente con herramientas como Postman.

REST es un estándar ampliamente adoptado que facilita la interoperabilidad, escalabilidad y modularidad. Su implementación en este proyecto permitió desarrollar un backend eficiente que proporciona datos al frontend de manera independiente y flexible.

Buenas prácticas de desarrollo

- **Estructura modular del código**:
    - El sistema está dividido en módulos claros (archivos, gestión de cargas, gestión de usuarios, procesamiento de documentos, etc.), lo que facilita el desarrollo colaborativo y la reutilización del código.
- **Control de versiones con Git**:
    - Se utilizo Git para gestionar el desarrollo del sistema, con ramas separadas para cada funcionalidad, garantizando un historial claro y organizo de cambios.
- **Pruebas unitarias y de integración**:
    - Implementadas para las funcionalidades críticas, como la autenticación y el procesamiento de documentos, utilizando el framework de pruebas integrado de Django.
- **Gestión de errores**:
    - Uso de bloques `try-except` para capturar y manejar excepciones de manera controlada, proporcionando mensajes de error claros a los usuarios.
    - Implementación de respuestas HTTP estandarizadas (códigos 200, 400, 401, 500) para facilitar el manejo de errores desde el frontend.
- **Buenas prácticas de seguridad**:
    - Cifrado de contraseñas con algoritmos estándar (PBKDF2).
    - Validaciones estrictas de entradas del usuario para prevenir inyecciones SQL, XSS y otros ataques.
    - Uso de tokens de autenticación para proteger rutas sensibles.
    - Configuración de protección CSRF para las solicitudes web.
- **Automatización y despliegue**:
    - Uso de Docker para garantizar entornos consistentes entre desarrollo y producción.
    - Planificación de integración continua (CI) con GitLab para ejecutar pruebas automáticamente en cada push al repositorio.

3.1.3 Integración con el sistema general.

La integración del backend con el resto del sistema, específicamente con el frontend, se realiza mediante HTTP Requests y respuestas en formato JSON. Este método permite una comunicación eficiente, estandarizada y desacoplada entre los componentes. El backend expone una serie de llamados mediante APIs REST que gestionan la lógica de negocio y la interacción con la base de datos, mientras que el frontend consume estas APIs para mostrar datos y procesar acciones de los usuarios.

Método de integración

1. **APIs REST**:
   - Cada funcionalidad clave del sistema está asociada a un conjunto de endpoints definidos.
   - Las respuestas son estructuradas en JSON, proporcionando datos en un formato estándar que puede ser interpretado fácilmente por cualquier cliente.
2. **Mensajes esperados**:
   - El frontend envía solicitudes HTTP (GET, POST, PUT, DELETE) al backend con los parámetros requeridos.
   - El backend procesa estas solicitudes y responde con un objeto JSON que incluye:
     - Código de estado HTTP: Indica el resultado de la llamada (e.g., 200 OK, 400 Bad Request, 401 Unauthorized, 500 Internal Server Error).
     - Mensaje de error o éxito: Detalla el resultado de la operación.
     - Datos: Información solicitada o procesada (e.g., datos de usuario, detalles de carga).
3. **Seguridad en las solicitudes**:
   - Las rutas protegidas requieren un token de autenticación en los encabezados HTTP:
     - `X-Auth-Token: token`
   - La comunicación entre frontend y backend se realiza mediante HTTPS para garantizar la seguridad de los datos transmitidos.

Documentación de las llamadas esperadas

A continuación, se presentan los endpoints principales utilizados en el sistema, organizados por módulos clave. La colección de todos los endpoints se encuentra en el anexo correspondiente (7).

1. **Llamados relacionados a usuario.**

DESCRIPCIÓN	ENDPOINT	MÉTODO	PARÁMETROS	¿REQUIERE ADMINISTRADOR?
INICIAR SESION CON CREDENCIALES	/users/login_user/	POST	EMAIL CONTRASEÑA	NO
CREAR USUARIO PARA INGRESO	/users/check_or_create_user/	POST	EMAIL NOMBRE CONTRASEÑA NUMERO TELEFONICO --OPCIONALES— ROL	NO* *SE REQUIERE PARA ASIGNACIÓN DE ROL A UN USUARIO

Tabla 3-1: Lista de endpoints relacionados al módulo de usuario

Fuente: Elaboración propia

2. Llamadas relacionadas a pagos

DESCRIPCIÓN	ENDPOINT	MÉTODO	PARÁMETROS	¿REQUIERE ADMINISTRADOR?
GENERAR PAGO A USUARIO	/pagos/create_pago/	POST	EMAIL MONTO	SI

Tabla 3-2. Lista de endpoints relacionados al módulo de pagos

Fuente: Elaboración propia

3. Llamadas relacionadas a las cargas

DESCRIPCIÓN	ENDPOINT	MÉTODO	PARÁMETROS	¿REQUIERE ADMINISTRADOR?
GENERAR CARGA PARA UN USUARIO	/cargas/create_carga/	POST	EMAIL DESCRIPCION ID_PAGO LOCALIZACION	SI
REVISAR CARGAS DE USUARIO PENDIENTES	/cargas/check_cargas_pendientes/	GET	N/A	NO

Tabla 3-3. Lista de endpoints relacionados al módulo de cargas

Fuente: Elaboración propia

4. Llamadas relacionadas a los tramites

DESCRIPCIÓN	ENDPOINT	MÉTODO	PARÁMETROS	¿REQUIERE ADMINISTRADOR?
GENERAR NUEVO TRAMITE	/tramites/create_tramite/	POST	TIPO TRAMITE USUARIO DESTINO ID DE CARGA LISTA DE ARCHIVOS REQUERIDOS	SI
REVISAR DETALLES DE TRAMITE	/tramites/check_tramite/	GET	ID DE TRAMITE	NO
REVISAR TRAMITES DE UN USUARIO	/tramites/check_tramites_user/	GET	N/A	NO

Tabla 3-4. Lista de endpoints relacionados al módulo de tramites

Fuente: Elaboración propia

5. Llamadas relacionadas a las notificaciones

DESCRIPCIÓN	ENDPOINT	MÉTODO	PARÁMETROS	¿REQUIERE ADMINISTRADOR?
CREAR NUEVA NOTIFICACION	/notificaciones/create_notification/	POST	TITULO MENSAJE DESTINATARIOS	SI
REVISAR NOTIFICACIONES DE USUARIO	/notificaciones/user_notifications/	GET	N/A	NO

Tabla 3-5. Lista de endpoints relacionados al módulo de notificaciones

Fuente: Elaboración propia

6. Llamadas relacionadas a los archivos

DESCRIPCIÓN	ENDPOINT	MÉTODO	PARÁMETROS	¿REQUIERE ADMINISTRADOR?
SUBIR UN ARCHIVO	/files/upload_file/	POST	ID DE TRAMITE ID DE TIPO DE ARCHIVO ARCHIVO	NO
VER ARCHIVOS REQUERIDOS DE USUARIO	/files/get_user_assigned_files/	GET	N/A	NO

Tabla 3-6. Lista de endpoints relacionados al módulo de archivos

Fuente: Elaboración propia

7. Llamadas relacionadas a utilización de algoritmo de comprobación de archivos

DESCRIPCIÓN	ENDPOINT	MÉTODO	PARÁMETROS	¿REQUIERE ADMINISTRADOR?
VALIDAR Y COMPROBAR ARCHIVO CON PLANTILLA	/ml/compare_file/	POST	URL DE ARCHIVO S3 NOMBRE PLANTILLA	SI

Tabla 3-7 Lista de endpoints relacionados al módulo de comprobación de archivos

Fuente: Elaboración propia

3.2 Detalles de la codificación y desarrollo.

Metodología de desarrollo

El proyecto fue realizado utilizando **SCRUM** como metodología ágil, dividiendo el desarrollo en ciclos iterativos llamados sprints. Cada sprint tuvo objetivos claros y entregables específicos, priorizando las funcionalidades del backend para garantizar el flujo de trabajo incremental y adaptable a los cambios de requerimientos.

El proyecto fue planificado para 3 sprints, de dos semanas cada uno, con el ultimo sprint teniendo una duración de una semana. Esta planificación permitió abordar tareas complejas de backend con mayor profundidad y realizar pruebas más detalladas antes de mostrarle los resultados al cliente.

Detalle de los sprints.

Sprint 1: Diseño y organización del proyecto
Objetivos principales:
- Establecer las bases del sistema backend en términos de diseño y estructura.
- Configurar herramientas y entornos para facilitar el desarrollo.

Actividades técnicas:
- Diseño de Arquitectura:
  - Definición de la arquitectura en capas con enfoque MVC.
  - Identificación de módulos backend clave (autenticación, tramites, pagos).
- Configuraciones iniciales:
  - Configuración de Django como framework backend.
  - Instalación de PostgreSQL y creación de tablas principales (usuarios, roles, tramites).
  - Configuración de contenedores Docker para garantizar consistencia entre entornos.

Sprint 2: Desarrollo inicial de funcionalidades backend
Objetivos principales:
- Implementar módulos principales del backend.
- Crear APIs REST para comunicación con el frontend.

Actividades técnicas:
- Implementación de modelos:
  - Usuarios, Roles, Tramites y pagos.
- Creación de Endpoints iniciales:
  - Autenticación de usuarios (local y Google).
  - Registro y consulta de usuarios.
  - Creación y consulta de tramites.
- Pruebas de validación:
  - Integración con GitLab CI/CD.
  - Pruebas unitarias automatizadas.
  - Pruebas de integración continua.

Sprint 3: Optimización y despliegue
Objetivos principales:
- Finalizar funcionalidades avanzadas del backend.
- Optimizar el rendimiento del sistema y desplegarlo en producción.

Actividades técnicas:
- **Funciones avanzadas:**
  - Notificaciones personalizadas.
  - Validación de documentos con PyTesseract y Celery.
- **Pruebas finales:**
  - Pruebas de rendimiento con Locust.
- **Despliegue en AWS:**
  - Configuración de VPC y contenedores Docker.

Historias de usuario

Las historias de usuario son descripciones breves de las funcionalidades necesitadas en el sistema, descritas desde la perspectiva del usuario final, detallando que necesita, para que y que valor aporta. A continuación, se describirán las historias de usuario principales que se definieron para las funcionalidades más críticas del sistema.

- Como tramitador, quiero subir documentos requeridos para los tramites, para evitar imprimir papeles constantemente.
- Como tramitador, quiero realizar los pagos de manera más fácil, para evitar errores de transferencias o problemas con dinero efectivo.
- Como tramitador, quiero recibir actualizaciones sobre mi tramite, para poder saber que está pasando con los documentos entregados.
- Como tramitador, quiero tener acceso a un historial de tramites realizados para tener constancia de lo que se ha realizado.
- Como conductor, quiero saber el estado de los tramites asociados a mi carga, para saber cuándo ir a buscar la carga.
- Como conductor, quiero recibir notificaciones sobre actualizaciones de los tramites de la carga a tiempo real, para no preguntar por el estado constantemente.
- Como conductor, quiero saber dónde está mi carga, para poder ir a retirarla.
- Como agente de visado, quiero revisar y aprobar o rechazar los documentos subidos más fácil para poder revisar más documentos.
- Como agente de visado, quiero asignar tramites a las personas más rápido, para poder recibir y revisar más tramites por día.
- Como agente de visado, quiero realizar búsquedas de trámites pendientes más rápido para terminar más tramites.
- Como agente de visado, quiero notificar a los usuarios de actualizaciones relacionados al proceso de tramites, para mantener informados a los usuarios.

A partir de las historias de usuario, los casos de uso detallan las interacciones especificas entre los actores y el sistema. Estos se centran en describir que hace el sistema para cumplir con cada historia de usuario, incluyendo el flujo principal y las posibles excepciones. En esta sección se presenta los casos de uso que se utilizan en el sistema, detallando los casos de uso más relevantes, mientras que la descripción completa de todos los casos de uso puede encontrarse en el anexo (7).

Para complementar la descripción de los casos de uso, se incluyen diagramas de secuencia que ilustran las interacciones entre actores, componentes internos del sistema y servicios externos durante la ejecución de cada caso de uso.

A continuación, se presenta el diagrama de caso de uso, donde se representa gráficamente las interacciones principales entre los actores del sistema y las funcionalidades que este ofrece.

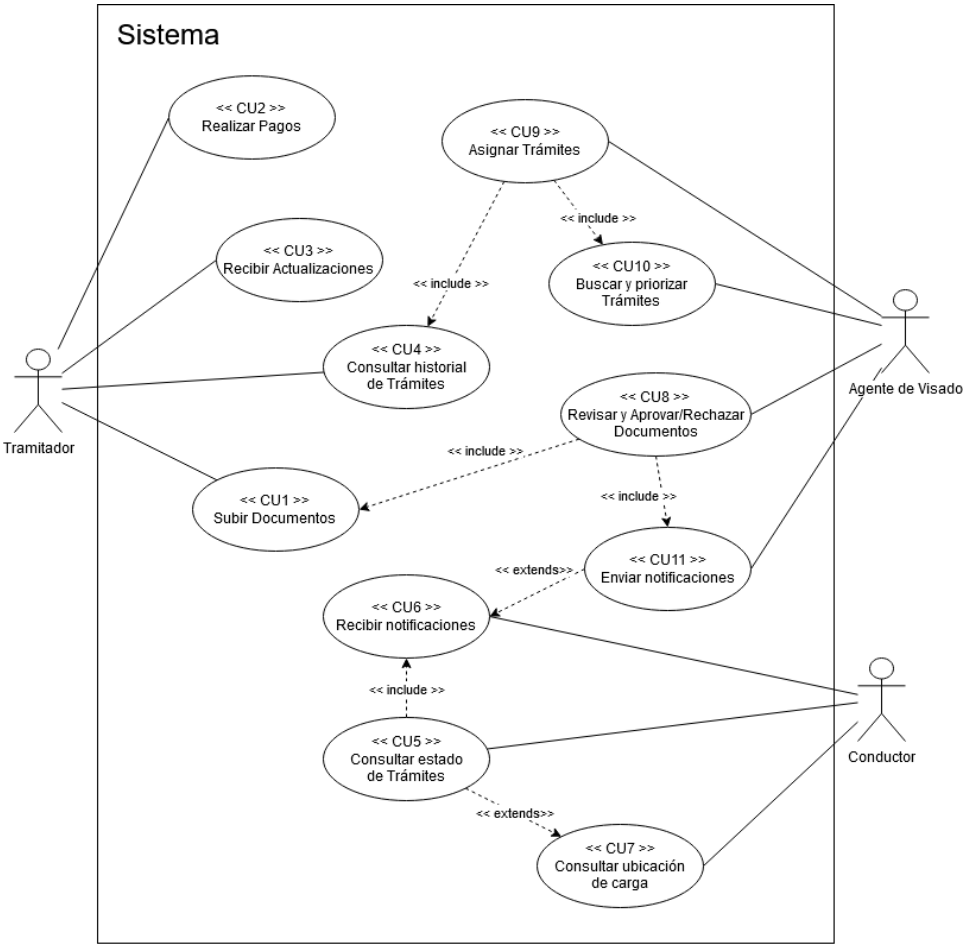

Figura 3–14 Diagrama de casos de uso

Fuente: Elaboración Propia

Caso de uso 1:
Subir documentos requeridos para los tramites.

ACTOR	**TRAMITADOR**
PROPOSITO	Permitir al tramitador cargar documentos relacionados con un trámite sin necesidad de imprimirlos.
FLUJO PRINCIPAL	1. Tramitador ingresa a la aplicación móvil e inicia sesión. 2. El sistema valida sus credenciales. 3. El tramitador ingresa a la pestaña de documentos pendientes. 4. El tramitador selecciona un documento que tenga pendiente y sube un archivo. 5. El sistema valida el formato y tamaño del archivo. 6. El archivo se almacena en AWS S3 y se registra en la base de datos. 7. El sistema confirma que el archivo se ha subido correctamente.
EXCEPCIONES	2ª. Si las credenciales son incorrectas, retorna un error. 5ª. En caso de que se encuentre un error, no se sube el archivo y se retorna un error.
PRECONDICIONES	El tramitador debe estar registrado en el sistema y tener una cuenta activa. El sistema debe tener documentos pendientes asociados al tramitador. La conexión a internet debe estar activa para realizar la subida de archivos.

POSTCONDICIONES	El archivo subido se registró en la base de datos al trámite correspondiente. El archivo quedo almacenado de forma segura en AWS S3. El tramitador recibió una confirmación que el documento fue subido correctamente.

Tabla 3-8. Diagrama de caso de uso 1

Fuente: Elaboración propia

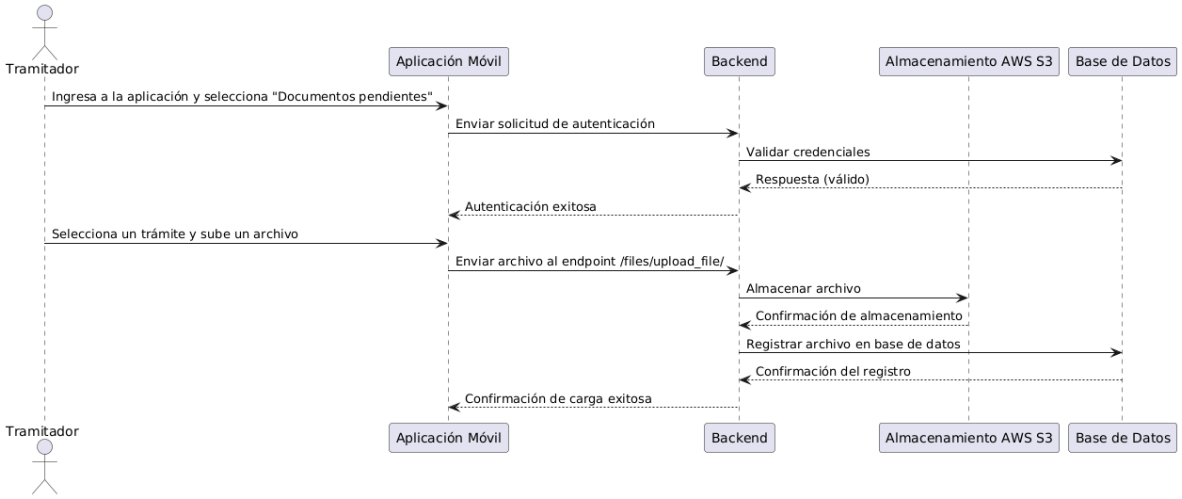

Figura 3–15. Diagrama de secuencia del caso de uso 1

Fuente: Elaboración propia

Caso de uso 7:
Consultar ubicación de la carga:

ACTOR	**CONDUCTOR**
PROPOSITO	Proveer al conductor la ubicación de su carga para planificar su retiro.
FLUJO PRINCIPAL	1. Tramitador ingresa a la aplicación móvil e inicia sesión. 2. El sistema valida sus credenciales. 3. El conductor accede a la sección de ubicación de la carga. 4. El sistema consulta la base de datos para obtener la ubicación asociada. 5. La aplicación muestra la ubicación en un mapa interactivo.
EXCEPCIONES	2ª. Si las credenciales son incorrectas, retorna un error. 4ª. En caso de que no haya alguna carga asignada, el sistema informa mediante un mensaje que no hay información disponible.
PRECONDICIONES	El conductor debe estar registrado en el sistema y tener una cuenta activa. El sistema debe asignadas cargas al conductor. La base de datos debe contener información actualizada sobre la ubicación de la carga. La conexión a internet debe estar activa para permitir la consulta y mostrar el mapa interactivo.
POSTCONDICIONES	El conductor visualizó la ubicación actual de su carga en el mapa interactivo. El sistema registró la consulta en el historial de acciones del conductor.

Tabla 3-9. Diagrama de caso de uso 7

Fuente: Elaboración propia

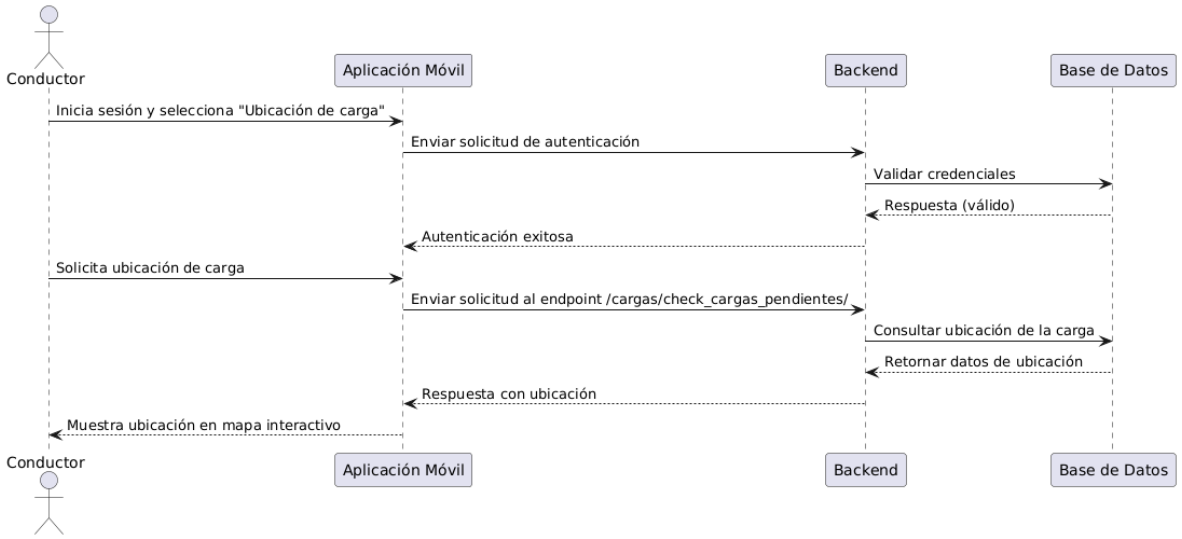

Figura 3–16. Diagrama de secuencia del caso de uso 7

Fuente: Elaboración propia

Caso de uso 8:
Permitir al agente de visado revisar documentos de manera eficiente.

ACTOR	**AGENTE DE VISADO**
PROPOSITO	Permitir al agente de visado revisar documentos de manera eficiente.
FLUJO PRINCIPAL	1. Tramitador ingresa a la aplicación web e inicia sesión. 2. El sistema valida sus credenciales. 3. El agente accede al listado de tramites realizados. 4. El sistema consulta la base de datos y devuelve un listado con los tramites. 5. El agente selecciona un trámite y visualiza los documentos subidos. 6. El sistema retorna los documentos relacionados al trámite seleccionado. 7. El agente revisa cada documento y elige si aprobar o rechazar el documento. 8. El sistema actualiza el estado del documento en la base de datos.
EXCEPCIONES	2ª. Si las credenciales son incorrectas, retorna un error. 4ª. En caso de que no haya tramites realizados en el sistema, se informa que no hay información disponible. 6ª. Si se han aprobado todos los documentos del trámite, se informa que ya se han aprobado los documentos.
PRECONDICIONES	El agente de visado debe tener una cuenta activa en el sistema y tener una cuenta activa. Debe haber tramites realizados con documentos pendientes de revisión registrados en la base de datos. El sistema debe estar conectado a la base de datos y en funcionamiento.
POSTCONDICIONES	El estado de los documentos revisados se actualizo en la base de datos. El agente de visado recibió una confirmación de las acciones realizadas (aprobación o rechazo)

Tabla 3-10. Diagrama de caso de uso 8

Fuente: Elaboración propia

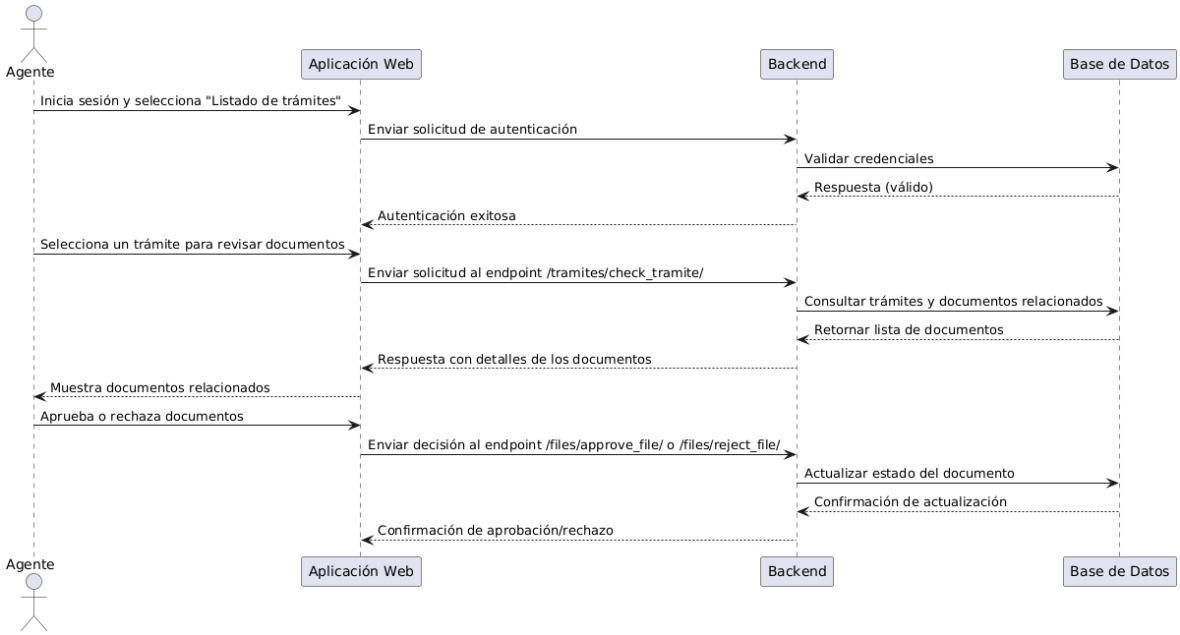

Figura 3–17. Diagrama de secuencia del caso de uso 8

Fuente: Elaboración propia

Control de versiones

Se utilizo **Git** como sistema de control de versiones, siguiendo una estructura de ramas organizada:

- `Main`: Versión estable del proyecto.
- `Dev`: Rama de desarrollo donde se integran las nuevas funcionalidades antes de ser fusionadas con main.
- `Dev-server`: Rama dedicada al desarrollo de funcionalidades experimentales que puede o no ser utilizadas dentro del proyecto.

Mecanismos de CI/CD

Se implementaron mecanismos de integración continua (CI), con un enfoque en la automatización del proceso de pruebas y la preparación para el despliegue. El despliegue en AWS se realiza de manera manual, permitiendo evaluar y preparar versiones listas para que así no se afecte el entorno de producción.

1. **Pruebas automatizadas en GitLab CI/CD:**
   - Configuración de un pipeline que ejecuta prueba unitarias e integración de forma automática al realizar un `commit` o `merge request`.
   - Garantiza que los cambios sean validados antes de pasar a producción.
2. **Despliegue en AWS:**
   - Realizado de manera manual, copiando la versión validada desde el entorno local hacia el servidor remoto.
   - Permite mayor control en que versión es utilizada en producción, garantizando el correcto funcionamiento con los otros sistemas.

3.3 Pruebas y Validación

Para garantizar que el backend cumple con los requerimientos funcionales y no funcionales, se implementó un enfoque integral de pruebas y validación. Este proceso abarca desde la verificación de componentes individuales hasta la validación de flujos completos en entornos controlados.

1. **Automatización de pruebas:**
   - Configuración de pipelines en GitLab CI/CD para ejecutar pruebas unitarias e integración automáticamente en cada commit.
   - Validación continua para detectar errores de forma temprana.
2. **Pruebas manuales:**
   - Realización de pruebas funcionales en casos de uso específicos que simulan el comportamiento real del sistema.
   - Realización de pruebas funcionales dentro del sistema real una vez verificado el funcionamiento del módulo correspondiente.
3. **Validación de resultados:**
   - Comparación de los resultados con los criterios de aceptación definidos.
   - Documentación de las respuestas obtenidas, tanto de error como de éxito en cada llamada.

3.3.1 Estrategias de testing aplicadas a los componentes.

Para garantizar la calidad y el correcto funcionamiento del backend, se han implementado diversas estrategias de Testing enfocadas en validar los componentes individuales, su integración y su rendimiento. A continuación, se describen las estrategias aplicadas.

1. **Pruebas Unitarias:**
   - **Objetivo**: Validar que los componentes individuales, como modelos, vistas y funciones, funcionan correctamente de manera aislada.
   - **Componentes probados**:
     - Modelos: Validación de restricciones, relaciones y métodos personalizados.
       - Ejemplo: Verificar que la creación de usuarios con roles predeterminados se realice correctamente.
     - Funciones: Comprobación de lógica de negocio, como generación de tokens y validación de credenciales.
   - **Herramientas utilizadas**:
     - Modulo integrado de pruebas en Django (`unittest`).
   - **Integración en GitLab CI/CD**:
     - Las pruebas serán ejecutadas automáticamente en cada commit dentro del pipeline previamente configurado en GitLab.

2. **Pruebas de Integración:**
   - **Objetivo**: Validar la interacción entre los distintos componentes del sistema.
   - **Componentes probados**:
     - Interacción entre backend y la base de datos.
     - Flujo de autenticación y autorización.
     - Validación del comportamiento de los endpoints REST y su lógica de negocios.
   - **Herramientas utilizadas**:
     - Postman: Herramienta utilizada para probar manualmente las solicitudes HTTP y verificar respuestas esperadas de los endpoints.
   - **Metodología**:
     - Cada endpoint API cuenta con documentación de los escenarios probados, indicando:
       - **Códigos de estado HTTP** (e.g., 200 OK, 400 Bad Request, 401 Unauthorized, etc.)
       - **Formato de la Respuesta**: Descripción y ejemplos de los datos esperados (atributos JSON)

3. **Pruebas funcionales:**
    - **Objetivo**: validar que las funcionalidades completas cumplen con los requerimientos definidos.
    - **Componentes probados**:
        - Flujo de creación de usuario y asignación de roles.
        - Flujo de retiro de carga como conductor.
        - Manejo de documentos y validaciones.
        - Gestión de notificaciones.
    - **Métodos utilizados**:
        - Pruebas manuales realizadas para simular casos de uso reales.
        - Validación de restricciones de acceso según roles definidos.
        - Ejemplos:
            - Un usuario con rol de conductor quiere revisar en qué estado esta su carga y donde tiene que ir a retirarla.
            - Un administrador crea una cuenta de usuario con rol de Visado para nuevo personal encargado del proceso de asignación de tramites.
            - Un usuario con rol de tramitador quiere subir un archivo que se le requirió previamente.
            - Un usuario revisa las notificaciones que tiene.
            - Un usuario con rol de Tramites o Conductor no puede acceder a funciones restringidas.

4. **Pruebas de regresión:**
    - **Objetivo**: Garantizar que los cambios que se hagan en el sistema no introduzcan errores en funcionalidades existentes.
    - **Componentes Probados**:
        - Funcionalidades críticas (usuarios, tramites, pagos, archivos, cargas y notificaciones)
        - Flujo completo de la creación de un trámite con todos sus componentes, siendo la creación de un pago, carga asignada al pago, creación de un usuario tramitador y conductor y la creación del trámite en sí.
    - **Métodos utilizados**:
        - Automatización de pruebas unitarias y pruebas manuales de integración, ejecutadas repetidamente mediante pipelines CI/CD y manualmente.

5. **Pruebas de rendimiento:**
    - Evaluar la capacidad del backend para manejar múltiples peticiones simultaneas bajo diferentes condiciones de carga.
    - **Componentes Probados:**
        - Autenticación de usuarios.
        - Conseguir detalles del usuario.
        - Creación de usuario.
        - Creación de tramites.
    - **Herramientas utilizadas:**
        - Se utilizo Locust para simular escenarios de alto tráfico y determinar métricas como:
            - Tiempos de respuestas promedio.
            - Tasa de errores bajo carga.
    - **Metodología utilizada:**
        - Generación de reportes detallados para identificar puntos críticos en el rendimiento.
        - Configuración de escenarios de prueba con diferentes niveles de usuarios simultáneos.

- Se incrementa la cantidad en 2 hasta llegar a los 100 usuarios simultáneos, donde se llama constantemente a los endpoints siguientes:
  - /users/login_user/
  - /users/get_or_create_user/
  - /users/get_user_details/
  - /tramites/create_tramite/

- **Resultados detallados:**

Type	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
POST	/tramites/create_tramite/	1157	0	26.72	16	167	63	15.2	0
POST	/users/check_or_create_user/	1145	0	13.56	8	78	101	15.04	0
GET	/users/get_user_details/	1116	0	14.53	8	118	120	14.66	0
POST	/users/login_user/	100	0	325.28	277	430	90	1.31	0
	Aggregated	3518	0	27.06	8	430	94.22	46.22	0

Figura 3–18. Estadísticas de rendimiento general

Fuente: Elaboración propia

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
POST	/tramites/create_tramite/	23	24	27	29	36	45	80	170
POST	/users/ check_or_create_user/	11	12	13	15	17	22	63	78
GET	/users/get_user_details/	11	12	13	15	19	29	66	120
POST	/users/login_user/	320	320	330	340	370	420	430	430
	Aggregated	14	18	21	24	32	58	330	430

Figura 3–19. Estadísticas de tiempos de respuesta

Fuente: Elaboración propia

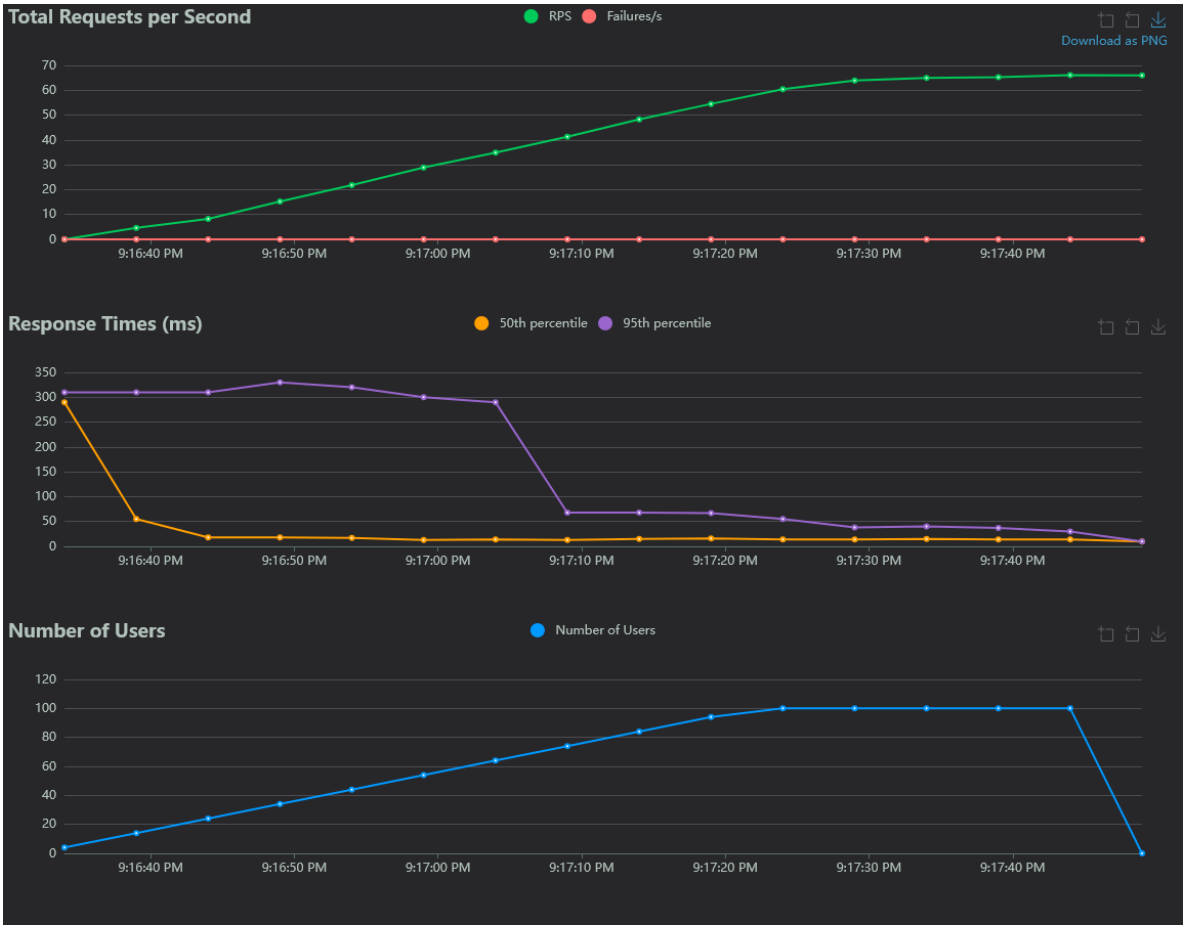

Figura 3–20. Gráficos de estadísticas generales

Fuente: Elaboración propia

3.3.2 Resolución de bugs y optimización

Durante el desarrollo, se realizaron pruebas al backend para la detección de problemas, las cuales fueron detectadas mediante:

1. **Pruebas Unitarias**: Validación de modelos y lógica de negocio.
    - E.g., Se identifico un error que no se borraba el token de un usuario si no cerraba sesión manualmente.
2. **Pruebas de Integración**: Verificación de interacciones entre componentes.
    - E.g., Dentro de la creación de un trámite, se producía un error debido a que no se validaba si los tipos de archivos eran válidos y estaban creados, ocasionando errores al momento de la creación.
3. **Pruebas de rendimiento**: Análisis de carga con Locust y disponibilidad de servidor.
    - E.g., Dentro de cuando se utilizaba el algoritmo de comprobación de archivo, el servidor procesaba esta solicitud y debido al tiempo que influye realizar este análisis, afectaba a todo el resto de las solicitudes.

Dado los problemas y errores que se ocasionaron durante el desarrollo, se utilizaron las siguientes metodologías para comprobar que ocasionaba estos errores y que se hizo para corregirlas:

1. **Reproducción de errores:**
    - Se utilizo y modifico el framework de pruebas de Django con cada cambio que se realizara en los modelos del sistema.
    - Se utilizo herramientas como Postman para confirmar problemas en los Endpoints.

2. **Depuración:**
    - Se utilizo de manera extensiva logs dentro de las vistas para rastrear el flujo de ejecución e identificar de esta manera donde se producía el error.

Dentro de los problemas identificados, se identificó problemas de optimización, donde el que tenía mayor impacto es el uso del algoritmo de comprobación de archivos, por lo que se hizo un estudio sobre posibles soluciones que ayudarían con la corrección de estos contratiempos.

Se decidió por utilizar una biblioteca de Python llamada **Celery**, que es comúnmente utilizada para gestionar tareas asíncronas. Esta herramienta nos ayuda a manejar procesos que consuman tiempo y serian convenientes ejecutar en segundo plano, permitiendo liberar recursos al servidor principal y mejorando los tiempos de respuestas.

Con esta implementación se logró que el servidor principal no tenga problemas al momento de procesar un archivo, debido a que lo manejara en segundo plano utilizando esta herramienta. También se realizaron optimizaciones en la base de datos, como normalización de relaciones entre modelos para reducir posibles redundancias.

Junto con las optimizaciones mencionadas y el uso de pipelines de GitLab CI/CD se garantizó que el sistema pueda trabajar sin errores y de manera óptima bajo casos de alta demanda.

3.4 Integración con Otros Componentes

En este proyecto, la integración con otros componentes de software se centra en servicios de AWS y la infraestructura en la nube, garantizando el correcto funcionamiento y almacenamiento seguro de los datos.

Para ámbitos de almacenamiento, se utilizó Amazon S3, donde se guardaron principalmente los archivos subidos por los usuarios y otros documentos relevantes para el funcionamiento del negocio. Esta implementación es manejada mediante solicitudes HTTP, para que se pueda utilizar desde la aplicación móvil y web. Esta implementación nos permite asegurar la seguridad de los archivos, debido a la implementación de la necesidad de claves de acceso y el uso de variables de entorno para manejar la seguridad de las credenciales.

En aspectos de despliegue, se utilizan los servidores disponibles en AWS mediante instancias de AWS EC2, lo que nos permite utilizar un entorno de producción y despliegue escalable y seguro, agregando a esto se utilizan contenedores Docker se ejecutan dentro del servidor, aislando el contenedor del resto de sistema y permitiendo mayor escalabilidad y modularidad del sistema.

Cabe mencionar que mientras que el sistema no interactúa con otros sistemas externos, la integración entre los componentes de APIs REST y Celery permiten la interacción con otros sistemas como aplicaciones móviles e interfaces web, permitiendo el acceso a los endpoints y ayudando al usuario final a interactuar con los componentes.

4 Conclusiones

El desarrollo del proyecto permitió un aprendizaje significativo en diversas áreas de la ingeniera de software, destacando especialmente el uso eficiente de técnicas modernas como la metodología ágil **SCRUM**, la implementación de sistemas robustos basadas en **APIs REST** y el diseño de una arquitectura efectiva para la administración de documentos. Además, la integración de algoritmos de **Machine Learning** represento un desafío técnico y personal, al ser la primera vez que se lograba aplicar una funcionalidad compleja en un sistema completo.

Aunque el sistema final logro cumplir con sus objetivos principales, aún existen aspectos que podrían mejorarse, como la optimización del rendimiento, el refinamiento del código para hacerlo más comprensible a terceros y la incorporación de herramientas más avanzadas como GitLab CI/CD para la automatización del despliegue en AWS. Estos aspectos representan áreas donde se puede tomar un aprendizaje para futuros proyectos e iteraciones.

En términos del aprendizaje clave que más importancia le tome personalmente, fue la comunicación en equipo, donde en los primeros sprints, la falta de coordinación afecto el cumplimiento de ciertos objetivos, lo que subrayo la necesidad de establecer comunicaciones claras y efectivas con el resto del equipo.

El impacto del proyecto en los usuarios fue evidente al presentar el producto final, donde se valoró el modularidad del sistema y su enfoque en la gestión eficiente de los documentos. Además, se destacó su potencial para ser adaptado a otros mercados que actualmente carecen de plataformas de gestión efectivas, ampliando el alcance y la relevancia del proyecto.

Finalmente, el proyecto no solo contribuyo a la sociedad al optimizar procesos manuales y mejorar la gestión documental, sino que también aporto a la disciplina informática al integrar tecnologías modernas y demostrar su aplicación práctica en un entorno real.

5 Agradecimientos.

Quisiera agradecer a mi familia por el enorme apoyo que me han dado, a mis amistades por haberme ayudado en las dificultades universitarias, en especial a Martin Valenzuela, por haber estado por mi cuando lo necesitaba, que en paz descanses.

6 Referencias

[1] PyPi, «PyPi - PyTesseract,» [En línea]. Available: https://pypi.org/project/pytesseract/.

[2] IBM, «IBM - What is optical character recognition,» [En línea]. Available: https://www.ibm.com/think/topics/optical-character-recognition.

[3] K. Shwaber y J. Sutherland, «The Scrum Guide,» [En línea]. Available: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf.

[4] A. Ken, «Gluo Mx,» gluo_, [En línea]. Available: https://www.gluo.mx/blog/backend-que-es-y-para-que-sirve.

[5] Amazon, «Amazon AWS,» [En línea]. Available: https://aws.amazon.com/microservices/.

[6] Cloudflare, «Cloudflare,» [En línea]. Available: https://www.cloudflare.com/learning/serverless/what-is-serverless/.

[7] Docker, «docker,» [En línea]. Available: https://www.docker.com/resources/what-container/.

[8] Geeksforgeeks, «Geeksforgeeks,» [En línea]. Available: https://www.geeksforgeeks.org/layered-architecture-in-computer-networks/.

[9] Geeksforgeeks, «Geeksforgeeks,» [En línea]. Available: https://www.geeksforgeeks.org/mvc-framework-introduction/.

[10] G. A. Martínez, «Codmind,» [En línea]. Available: https://blog.codmind.com/aprendiendo-el-estilo-arquitectonico-rest-a-profundidad/.

[11] Geeksforgeeks, «geeksforgeeks,» [En línea]. Available: https://www.geeksforgeeks.org/django-project-mvt-structure/.

[12] PostgreSQL, «PostgreSQL,» [En línea]. Available: https://www.postgresql.org/about/.

[13] Amazon, «Amazon AWS,» [En línea]. Available: https://aws.amazon.com/es/s3/.

[14] Locust, «Locust,» [En línea]. Available: https://locust.io/.

[15] Django Software Foundation, «Django Project,» [En línea]. Available: https://www.djangoproject.com/.

7 Anexos

Repositorio de documentación del proyecto (Casos de usos, Respuestas JSON, Endpoints)