UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

DEPARTAMENTO DE ELECTROTECNIA E INFORMÁTICA
INGENIERÍA EN INFORMÁTICA

Desarrollo de Backend para sistema de consultas sobre documentación interna mediante lenguaje natural

Juan Esteban Vilches Villalobos

juan.vilches@usm.cl

Carlos Alten
Profesor Guía

Hernán Saavedra
Profesor Correferente

Resumen: La dispersión de información técnica en la empresa Indemin dificulta la resolución de incidencias y eleva los costos operativos por tiempos de inactividad. Este trabajo presenta el desarrollo de un backend para un asistente virtual con arquitectura RAG (Retrieval-Augmented Generation), utilizando Django y LangChain, con el objetivo de centralizar el conocimiento y agilizar el soporte mediante consultas en lenguaje natural a manuales técnicos. La validación de la propuesta se realizó mediante pruebas unitarias, de integración y de carga utilizando Locust, verificando el comportamiento del sistema bajo concurrencia de hasta 100 usuarios. Los resultados confirman la robustez de la API y la eficacia en la recuperación semántica y trazabilidad de las sesiones, proyectando una reducción significativa en los tiempos de respuesta y una mejora en la gestión del conocimiento operativo.

Palabras Clave: Backend, RAG, Asistente Virtual, Django, Trazabilidad.

1 Introducción

Este capítulo abarca el contexto general del proyecto, exponiendo la problemática en la empresa Indemin derivada de la dispersión del conocimiento técnico, los altos costos de tiempo para capacitación y la resolución fragmentada de incidencias. Se presenta la solución propuesta: Un asistente virtual que consulta exclusivamente los manuales de la empresa para entregar respuestas fundamentadas a preguntas de los usuarios hechas en lenguaje natural.

En cuanto a alcances, este trabajo se centra en el desarrollo del sistema del lado del servidor y en la interacción con el modelo de lenguaje. Se formulan objetivos orientados a reducir tiempos de resolución de incidencias, mejorar la precisión de la asistencia y consolidar el conocimiento operativo. Además, se introduce el marco conceptual y la metodología de validación empleada con casos reales y métricas de desempeño. Finalmente, se presenta la organización del informe por capítulos, articulando estos contenidos con los objetivos planteados.

CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒Memoria o trabajo de título ☐ Tesis de Postgrado

Título del trabajo: Desarrollo de Backend para sistema de consultas sobre documentación interna mediante lenguaje natural

Nombre del candidato(a): Juan Esteban Vilches Villalobos

Carrera / Grado: Ingeniería Informática

Campus: Sede Viña del Mar **Departamento:** Electrotecnia e informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Carlos Alten López, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente

DEJO CONSTANCIA que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución.

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL (marcar una opción)

☒ El trabajo **NO contiene** información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por **(marcar una opción)**:

☐ 6 meses ☐ 12 meses ☐ 2 años ☐ 3 años ☐ 5 años ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 10 de marzo de 2026 **Firma:** 

Estudiante o Candidato(a):

Fecha: 06-03-2026 **Firma:** 

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

1.1 Contexto y antecedentes

En cualquier empresa, mantener la continuidad operativa es muy importante para sostener o incluso mejorar su productividad. La organización con la que se trabajó cuenta con una gran cantidad de equipos en diferentes faenas de Chile, equipos cuyo uso seguro y eficiente depende de manuales técnicos y del conocimiento acumulado por personal. La información relevante se encuentra distribuida entre documentos digitales, capacitaciones de larga duración y en soluciones puntuales aplicadas ante incidencias.
En la práctica, el personal de terreno puede encontrarse con dudas o problemas acerca del equipo que opera en reiteradas ocasiones, lo cual causa que estos deban recurrir a terceros para resolver las incidencias. Estos terceros que se mencionan la mayoría de las veces son supervisores o personal de otras áreas, pero este no es personal dedicado exclusivamente a resolver estas situaciones, por el contrario, son trabajadores que tienen otras responsabilidades, pero que, sin embargo, tienen libre acceso a un OneDrive donde se encuentran todos los manuales digitales de los equipos, teniendo que revisarlos para encontrar solución a las problemáticas que van surgiendo en terreno. Es importante mencionar que si es el individuo a quien se le pidió soporte en cuanto a la problemática no encuentra o sabe la solución, este problema sigue escalando hasta que alguien sea capaz de resolverlo.

Frente a ese escenario, la empresa busca una forma sistemática de poner el conocimiento técnico "al alcance de la mano" de quienes operan o mantienen los equipos. La motivación principal es reducir tiempos de paralización de los equipos, mejorar la uniformidad de las indicaciones entregadas y disminuir la carga de capacitaciones recurrentes.

1.2 Definición del problema

Si se analiza el contexto de la empresa, pueden encontrarse diferentes problemas que afectan de forma negativa la continuidad operativa de los equipos en terreno cuando surgen incidencias, a continuación, se detalla cada uno de ellos:

N°	Problema identificado	Descripción del problema
1	Distribución de la información técnica asociada a los equipos	La información técnica que se encuentra documentada no se encuentra bien organizada, las capacitaciones y las soluciones puntuales una vez aplicadas, quedan atrás.
2	Repetición de incidencias	Una misma incidencia puede volver a ocurrir en reiteradas ocasiones a la misma persona o a otra diferente, cualquiera sea el caso, si esta no sabe o no recuerda cómo solucionar el problema, solicita ayuda para el mismo problema paralizando de forma repetitiva las operaciones por algo que ya se ha solucionado antes.
3	Falta de trabajadores con dedicación exclusiva a la resolución de incidencias	Los encargados de resolver las incidencias tienen otras responsabilidades asociadas a sus cargos, lo que demora aún más la resolución de los problemas ya que pueden no tener disponibilidad inmediata. Además, deben dedicar tiempo que podría utilizarse en otras actividades relacionadas con sus funciones, lo que termina afectando su productividad.
4	Falta de acceso a manuales por parte de	Como los operarios u operadores no tienen acceso directo a los manuales de los equipos, lo único que

	los trabajadores en terreno	pueden hacer es escalar la situación a alguien con acceso y esperar a que este tenga disponibilidad de atenderlo, lo que deja paralizado su trabajo y afecta directamente la productividad de la empresa.
5	Búsqueda manual en la documentación para encontrar soluciones	Los encargados de solucionar las incidencias deben buscar manualmente la información relacionada en manuales que pueden llegar a tener cientos de páginas, afectando directamente el tiempo de resolución y, por ende, la paralización del equipo afectado.
6	Altos costos en cuanto a tiempo para capacitar personal	El elevado tiempo que se utiliza para capacitar al personal hace que estas capacitaciones se hagan solo cuando son estrictamente necesarias, por lo que, si uno o varios temas no quedan claros para algunos, difícilmente se podrán repetir.
7	Escalamiento de incidencias	Solicitar ayuda con una incidencia no siempre garantiza que sea solucionada en primera instancia, por lo que puede escalar a otras personas. Esto se relaciona directamente con el tiempo que un equipo puede estar paralizado.

Tabla 1-1 Lista de problemas identificados con sus descripciones

Fuente: Elaboración propia

1.3 Breve descripción sobre la propuesta de solución

Se desarrollará una aplicación web que incorporará un asesor virtual para consultas técnicas sobre los equipos. El usuario podrá seleccionar el equipo, formular su pregunta en lenguaje natural y recibir una respuesta fundamentada en los contenidos pertinentes del manual correspondiente (función principal).

Para el personal con permisos de administrador, la aplicación incluirá:

- **Gestión de usuarios:** creación, edición, asignación de roles y control de accesos.
- **Gestión de manuales:** Agregar y eliminar manuales con clasificación por marca y tipo de equipo, además de editar sus metadatos.
- **Trazabilidad:** Panel para revisar conversaciones por sesión y disponer de registros que faciliten auditoría y mejora continua.

Desde el backend se implementarán funcionalidades clave: autenticación por sesiones (login/logout) y un CRUD administrativo de usuarios con contraseñas cifradas; gestión del catálogo de marcas, equipos y manuales a nivel de metadatos y almacenamiento seguro de archivos; trazabilidad de las conversaciones mediante registro de sesiones, mensajes y citas; y la orquestación del flujo consulta–respuesta con un LLM (Modelo de Lenguaje Grande), exigiendo que el usuario seleccione el equipo para acotar el contexto. También se desarrollarán APIs que garantizarán una integración eficiente y segura con la interfaz web del lado del frontend.

1.4 Objetivos generales y específicos

Objetivo general: Desarrollar un sistema backend seguro para la implementación de un asesor virtual basado en manuales técnicos que integre funcionalidades de gestión y trazabilidad, con el fin de optimizar los tiempos de soporte técnico y centralizar el conocimiento operativo de la empresa.

Objetivos específicos:

1. **Implementar un sistema de autenticación y autorización seguro** basado en roles, que garantice el acceso controlado a las distintas funcionalidades de la plataforma de gestión.

2. **Diseñar y desarrollar un módulo de gestión de usuarios** que facilite las operaciones de creación, edición, habilitación/deshabilitación y asignación de roles a través de la interfaz de administración.

3. **Desarrollar un módulo para la administración de manuales técnicos**, permitiendo las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre los documentos que alimentan al asesor virtual.

4. **Diseñar y desarrollar las APIs necesarias** para una comunicación segura y eficiente entre la interfaz de usuario (frontend) y los servicios del backend.

5. **Implementar el flujo de procesamiento de consultas** en dos fases: primero, realizando una búsqueda y recuperación de fragmentos de texto relevantes de los manuales y, segundo, utilizando un modelo de lenguaje avanzado para la generación de respuestas contextualizadas y coherentes a partir de dicha información.

6. **Establecer un sistema de registro y trazabilidad** que almacene las interacciones del usuario, incluyendo la consulta realizada, los documentos consultados y la respuesta final del asesor, para fines de auditoría y mejora continua.

1.5 Justificación del proyecto

Relevancia técnica
El proyecto aborda una necesidad concreta: disminuir los tiempos de resolución de incidencias y consolidar el conocimiento operativo mediante un backend que expone APIs REST seguras para autenticación por sesiones, gestión de usuarios y catálogos (marcas, equipos, manuales), almacenamiento seguro de archivos, y trazabilidad completa de sesiones, mensajes y citas. Además, orquesta el flujo consulta–respuesta de un asesor virtual acotado por equipo, devolviendo respuestas con citas al manual correspondiente.

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** + **Django REST Framework**, lo que permite exponer APIs REST para funcionalidades clave como la gestión de usuarios, roles y documentación (marcas, equipos y manuales), además de la trazabilidad de sesiones y mensajes. Este diseño modular favorece el desacoplamiento con el frontend, reduce el time-to-feature y facilita la evolución sin reescrituras profundas.

La base de datos utilizada es **PostgreSQL**, un sistema confiable que ofrece consistencia transaccional e indexación avanzada, asegurando consultas eficientes para reportabilidad y auditorías.

El asesor virtual incorpora **LangChain** [1] con embeddings de HuggingFace, un índice FAISS y un cliente oficial de **OpenAI** para generar respuestas, resúmenes y citas trazables sobre los manuales técnicos.

Cómo se benefician los usuarios finales
Para el operador y personal de soporte, el sistema entrega respuestas fundamentadas en los manuales del equipo seleccionado, reduciendo búsquedas manuales y dependencia de terceros. Para administradores, el panel de gestión y la trazabilidad permiten ordenar el repositorio documental, auditar conversaciones y detectar brechas de conocimiento. En conjunto, la experiencia de consulta se vuelve más rápida, consistente y verificable, y la organización gana visibilidad sobre el uso y los cuellos de botella.

Innovación
En lo que respecta al lado del servidor, el proyecto utiliza tecnologías emergentes tales como **LangChain** que permite trabajar con modelos de lenguaje grandes en la aplicación permitiendo construir flujos de trabajo de IA más potentes como chatbots avanzados. Por otro lado, se utiliza **PyTesseract**, una herramienta de reconocimiento óptico de caracteres (OCR) para Python, empleada para extraer texto de documentos escaneados (imágenes o PDF no seleccionables) con el fin de normalizar su contenido antes de la indexación [2]. Esta tecnología no se detalla en el presente documento, sino que se aplica en la implementación de la base de datos vectorial para la búsqueda semántica sobre manuales técnicos [3].

1.6 Metodología aplicada

Se utilizará Scrum para el desarrollo del backend debido a la necesidad de iterar con rapidez, ajustar prioridades según feedback de la empresa y entregar valor incremental (módulos de usuarios, catálogos, trazabilidad y consulta-respuesta) [4]. Scrum facilita visibilidad continua, gestión de riesgos y alineamiento con los responsables de frontend y del buscador documental.

Para alcanzar los objetivos del proyecto, se utilizarán diversas herramientas colaborativas. En primer lugar, se empleará Discord para realizar reuniones y consultas grupales, y Microsoft Teams para la comunicación con el cliente. ClickUp será utilizado para la gestión del proyecto, y finalmente, Git se usará para el control de versiones del código fuente.

1.7 Breve descripción de la organización del informe en capítulos

A continuación, se describe de forma breve y directa la descripción de los capítulos que componen el presente trabajo de título:

1. **Introducción**: Presenta el contexto, motivación y problemática general, la definición del problema, una síntesis de la propuesta de solución, los objetivos del trabajo, la metodología de desarrollo y validación, y la organización del documento.

2. **Marco conceptual**: Fundamentos teóricos que sustentan el desarrollo del backend del sistema. Expone conceptos necesarios para comprender el sistema y fija supuestos y restricciones.

3. **Diseño e implementación**: Se describe detalladamente el proceso de diseño e implementación del backend del sistema, detallando los componentes principales, tecnologías utilizadas y su integración con la interfaz web.

4. **Conclusión**: Resume logros y grado de cumplimiento de objetivos, limitaciones detectadas y líneas de trabajo futuro.

5. **Agradecimientos**: Reconocimiento a personas e instituciones que aportaron al desarrollo del proyecto.

6. **Referencias**: Fuentes bibliográficas y documentales utilizadas.

7. **Anexos**: Material complementario.

2 Marco conceptual

En esta sección se presentarán los fundamentos teóricos que sustentan el desarrollo del sistema.

2.1 Fundamentos del desarrollo de backend

El backend, también conocido como el "lado del servidor", se refiere a la parte de una aplicación que opera en segundo plano, inaccesible directamente para el usuario final. Es la maquinaria interna que procesa solicitudes, gestiona datos y ejecuta la lógica que hace que la aplicación funcione [5].

El concepto surgió junto con el desarrollo de los primeros sistemas informáticos en la década de 1950. Sin embargo, la definición moderna de desarrollo de backend comenzó a tomar forma en la década de 1990 con la llegada de internet y los sitios web dinámicos [6].

El Rol del Backend en la Arquitectura de Software
El backend es la capa que implementa la lógica de negocio, expone servicios (por ejemplo, APIs REST), gestiona datos (persistencia, consultas, transacciones) y aplica políticas de seguridad (autenticación, autorización, auditoría). Su diseño condiciona la integridad de la información, el rendimiento y la evolutividad del sistema (mantenibilidad, escalabilidad, extensibilidad).

Esta capa del servidor es la responsable de recibir las peticiones del cliente (frontend), procesarlas, interactuar con otros sistemas como bases de datos o servicios de terceros, y devolver una respuesta estructurada que el frontend pueda interpretar y presentar al usuario [7]. En esencia, el backend es el pilar invisible que soporta cada interacción digital, transformando las acciones del usuario en resultados funcionales.

A continuación, se resume cómo el backend asume tres funciones esenciales dentro de una aplicación:

- **Procesamiento de datos**: Recibe la información que envía el cliente u otros sistemas, revisa que esté completa y con el formato correcto, la ordena y la transforma cuando hace falta, y prepara una respuesta clara y consistente.

- **Lógica de negocio:** Aplica las reglas del sistema (qué está permitido y qué no), decide los pasos a seguir en cada caso de uso y coordina las acciones necesarias para cumplir las solicitudes del usuario.

- **Manejo de bases de datos:** Guarda la información de forma segura, la recupera cuando se necesita, evita pérdidas o duplicados y mantiene el historial de cambios para asegurar la coherencia de los datos.

Tipos de sistemas que utilizan backend
El desarrollo de backend es fundamental en muchos sistemas. Algunos casos de uso incluyen:

- **Plataformas de comercio electrónico**: manejo de cuentas de usuario, transacciones e inventarios de productos [6].

- **Redes sociales**: gestión de perfiles de usuario, publicaciones, mensajes y feeds [6].

- **Aplicaciones empresariales**: creación de software que gestiona funciones empresariales, como RR.HH. o gestión de relaciones con los clientes [6].

- **Aplicaciones bancarias y financieras**: garantizar el procesamiento seguro de transacciones y datos financieros [6].

- **Aplicaciones móviles**: proporcionamos el soporte del lado del servidor necesario para las aplicaciones que se ejecutan en teléfonos inteligentes [6].

Ventajas
El desarrollo de backend aporta los siguientes beneficios:

- **Escalabilidad**: Una arquitectura de backend adecuada garantiza que las aplicaciones puedan escalar para manejar más usuarios o datos [6].

- **Seguridad de datos**: Los desarrolladores backend implementan medidas de seguridad sólidas, como protocolos de encriptación y autenticación, para proteger datos confidenciales [6].

- **Optimización del rendimiento**: Mediante una configuración y optimización adecuadas del lado del servidor, los desarrolladores backend garantizan que las aplicaciones se ejecuten de manera eficiente [6].

- **Gestión de datos:** Los desarrolladores backend mantienen y optimizan bases de datos para almacenar, recuperar y administrar grandes cantidades de datos rápidamente [6].

Desventajas
El desarrollo de backend tiene incluye las siguientes dificultades:

- **Complejidad:** Administrar múltiples bases de datos, configuraciones de servidores y garantizar una interacción fluida entre el desarrollo del frontend y del backend puede ser complejo [6].

- **Riesgos de seguridad:** Proteger información confidencial y prevenir violaciones de datos es un desafío importante, especialmente para las aplicaciones que manejan datos privados o financieros [6].

- **Problemas de escalabilidad**: A medida que una aplicación crece, su backend debe optimizarse continuamente para manejar mayores cargas, lo que puede requerir un esfuerzo de ingeniería sustancial [6].

- **Integración con el frontend:** Garantizar una integración y comunicación fluida entre los componentes del frontend y del backend a veces puede ser un desafío, especialmente en aplicaciones grandes y complejas [6].

Tendencias en la industria
Con el paso del tiempo emergen muchas tendencias, actualmente las más populares son:

- **Inteligencia artificial generativa**: Organizaciones de todo el mundo están integrando modelos de lenguaje de gran tamaño (LLM) y modelos fundacionales (FM) para la creación de aplicaciones de IA generativa y así ofrecer nuevas y mejores experiencias a los usuarios [8].

- **Arquitecturas serveless y cloud-native:** Mantener servidores de forma manual empieza a ser parte del pasado, cada vez son más las empresas que pasan su infraestructura física a la nube por múltiples ventajas que esto representa como por ejemplo, automatización (se ahorra la labor manual de gestionar servidores), escalabilidad (se amplían y reducen automáticamente en función del tráfico) y lo más importante, productividad ya que permite a los desarrolladores centrarse en escribir código y optimizar la lógica empresarial [9].

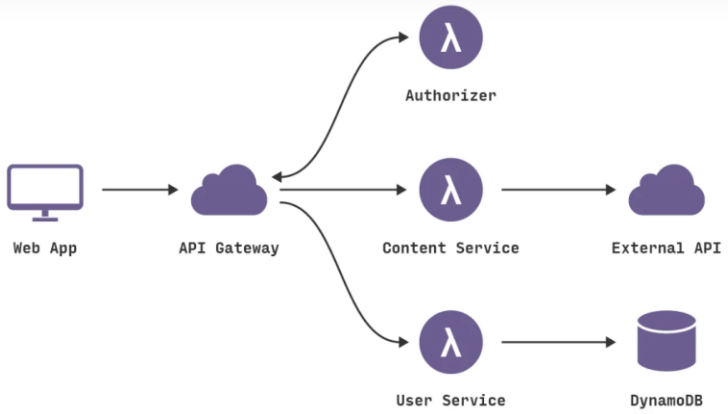

Figura 2-1 Arquitectura serverless con API Gateway y funciones

Fuente: "Functions as a Service: Introducción a arquitecturas serverless". Disponible en: https://nubity.com/functions-as-a-service-introduccion-a-arquitecturas-serverless/

- **Contenedores:** Los contenedores constituyen un mecanismo de empaquetado lógico en el que las aplicaciones pueden extraerse del entorno en que realmente se ejecutan [10]. Esta capacidad de aislamiento simplifica y robustece el despliegue, ya que el contenedor se convierte en un paquete portable y autocontenido que funcionará de manera idéntica y predecible en cualquier entorno.

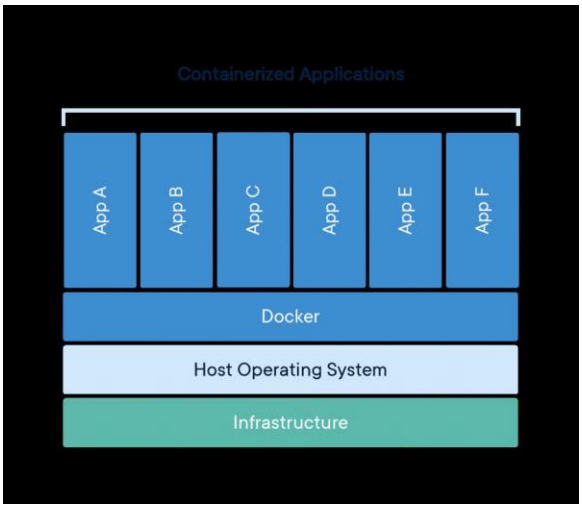

Figura 2-2 Aplicaciones en contenedores

Fuente: "Utilice contenedores para crear, compartir y ejecutar sus aplicaciones". Disponible en https://www.docker.com/resources/what-container/

- **Retrieval-Augmented Generation (RAG)**: Proceso que se encarga de optimizar la salida de un lenguaje de gran tamaño, de modo que antes de generar una respuesta, redirige el LLM para recuperar información relevante de fuentes de conocimiento autorizadas. De este modo, las organizaciones obtienen un mayor control sobre las salidas de texto generadas [11].

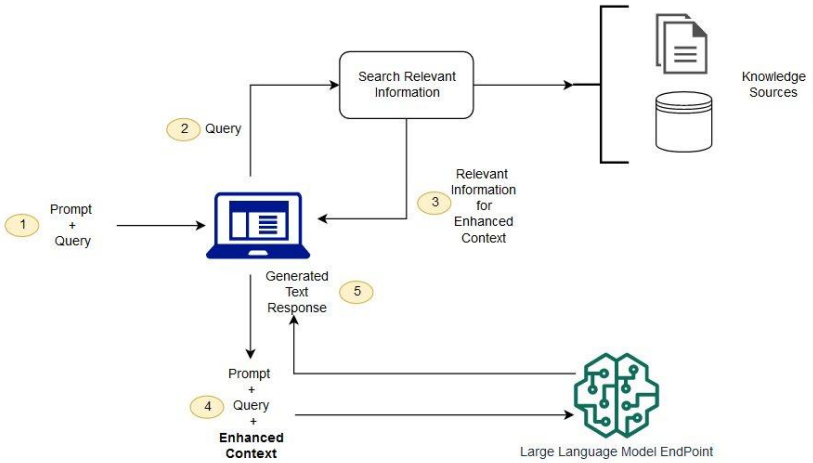

Figura 2-3 Flujo conceptual del uso de RAG con un LLM

Fuente: "¿Qué es RAG?". Disponible en: https://aws.amazon.com/es/what-is/retrieval-augmented-generation/

2.2 Arquitectura de software

La arquitectura de software define cómo se estructuran los componentes del sistema y cómo interactúan. En este proyecto se adopta una arquitectura en capas basada en APIs, con separación clara entre presentación (frontend), lógica de negocio (backend) y persistencia de datos. Esta organización favorece la mantenibilidad, la escalabilidad y la trazabilidad.

Arquitecturas y patrones utilizados

- **Arquitectura en capas**: Arquitectura de software que divide la aplicación en varias capas con roles bien definidos. No fija un número exacto de capas, lo importante es separar responsabilidades y que cada capa solo se comunique con la inmediatamente inferior. Cada una puede implementarse como componente independiente, este diseño busca claridad, mantenibilidad y control de dependencias [12]. En este proyecto se utilizan las siguientes capas:
    - **Capa de Presentación (Frontend)**: Interfaz web que consume APIs para autenticación, gestión de catálogos (marcas, equipos, manuales) e interacción con el asesor virtual.
    - **Capa de Lógica (Backend)**: Servicios que implementan reglas de negocio, validaciones, flujo consulta–respuesta del asesor y trazabilidad de sesiones y mensajes.
    - **Capa de Persistencia**: Base de datos relacional para entidades operativas (usuarios, marcas, equipos, manuales, sesiones, mensajes, citas) y almacenamiento de archivos (manuales). La indexación/búsqueda especializada de manuales se gestiona como un componente externo, se guarda un índice vectorial en disco para que el asistente pueda reutilizar los manuales sin reconstruirlos en cada arranque [3].

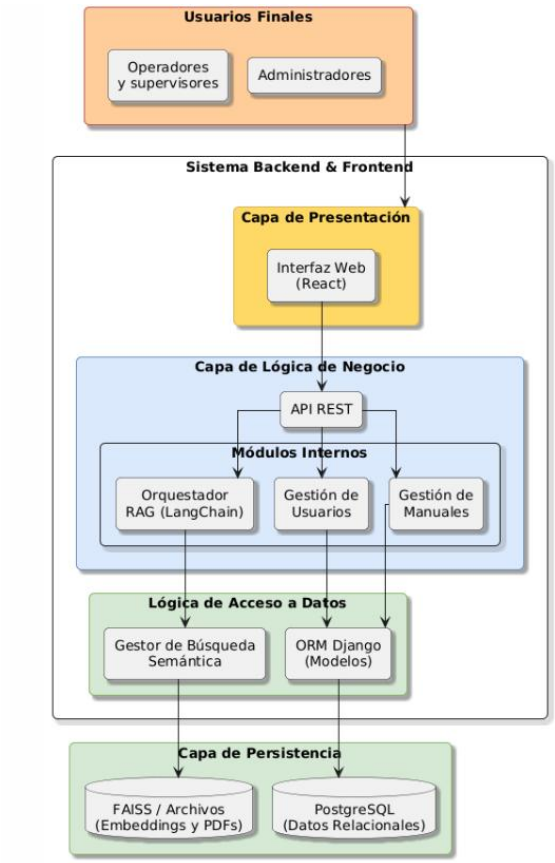

Figura 2-4 Diagrama de ejemplo para una arquitectura en capas del sistema

Fuente: Elaboración propia

- **Patrón MVC y ORM**

    - **Modelo**: Define las reglas de negocio (por ejemplo, qué requisitos debe cumplir un usuario para registrarse) y realiza las operaciones de crear, leer, actualizar o borrar datos.

    - **Vista**: Presenta la información. En aplicaciones modernas, esto suele ser un archivo HTML para el navegador o un objeto JSON que otra aplicación consumirá (caso de este sistema).

    - **Controlador**: Recibe las peticiones del usuario y decide qué hacer con cada solicitud. Pide datos al modelo, recibe la respuesta y luego elige qué datos o estados entrega.

    - **ORM**: Permite trabajar con datos como objetos y persistirlos en tablas sin escribir consultas de bajo nivel, lo que simplifica el mantenimiento y reduce errores.

Es importante destacar que a pesar de que Django por defecto utiliza el patrón de diseño MVT, en este caso el sistema utiliza MVC. Más adelante, en la sección "3.2.2 Uso de buenas prácticas y patrones de diseño" se justifica más en detalle.

2.3 Tecnologías y frameworks utilizados

Para el desarrollo del backend de este proyecto se utilizaron tecnologías que garantizan un sistema eficiente, escalable y seguro. A continuación, se detalla cada una de ellas.

Lenguaje de programación

- **Python**: Seleccionado principalmente por ser un lenguaje de alto nivel con una sintaxis limpia y legible. Cuenta con un ecosistema maduro de librerías web e inteligencia artificial (entre otras), de modo que es fácil incorporar desde frameworks REST hasta herramientas de IA sin reinventar componentes básicos. Su comunidad activa y la documentación extensa reducen el riesgo tecnológico: es sencillo encontrar buenas prácticas, soporte y actualizaciones de seguridad a largo plazo. Además, es multiplataforma y tiene un modelo de paquetes sencillo (pip/virtualenv), lo que simplifica desplegar la misma solución en distintos entornos y automatizar pipelines de integración continua.

Frameworks

- **Django**: Dada la elección de Python como lenguaje de programación, se utiliza Django porque Provee de fábrica una estructura robusta con módulos integrados para administración, autenticación y seguridad, permitiendo activar protecciones nativas contra ataques comunes (como CSRF) y el control de CORS sin configuraciones adicionales. Su ORM permite modelar el dominio completo (incluyendo manuales, control de acceso y trazabilidad) en una única capa coherente que reutiliza validaciones y relaciones del framework, eliminando la gestión manual de consultas SQL o conexiones a la base de datos. Asimismo, el empleo de funciones integradas para la gestión de usuarios y sesiones agiliza la implementación de una infraestructura de seguridad confiable sin necesidad de desarrollar componentes base desde cero.

- **Django REST Framework (DRF)**: Extiende las capacidades de Django para la creación de APIs REST mediante herramientas que centralizan la lógica de los endpoints a través de viewsets y routers. El uso de serializadores y respuestas tipadas garantiza entradas y salidas predecibles, transformando modelos a JSON con validaciones declarativas. Asimismo, DRF estandariza el contrato de la API y reduce el código repetitivo al integrar soporte nativo para autenticación por sesión, permisos y manejo transaccional, elementos fundamentales para el ciclo de consulta-respuesta del asistente.

Base de datos

- **PostgreSQL**: Sistema de base de datos relacional, escalable y robusto [13]. Es compatible con ORM, por lo que puedes trabajar con objetos en el código y que el motor se encargue de las tablas y consultas.

    - **Transacciones ACID**: O todo el cambio se guarda o nada, evitando datos a medias.

    - **Tipos de datos útiles**: JSONB para guardar documentos JSON y consultarlos.

    - **Replicación**: Permite tener copias en otros servidores para alta disponibilidad y lectura sin sobrecargar el principal.

En la práctica, mantiene la integridad (datos correctos), ofrece buen desempeño en consultas y facilita el desarrollo al integrarse bien con frameworks como Django.

Modelo de lenguaje grande (LLM)

- **Gpt-5-mini:** Centraliza la resolución de consultas y el resumen de documentación técnica mediante la API oficial de OpenAI dentro del flujo RAG. El uso de prompts específicos garantiza respuestas en español estrictamente fundamentadas en el contexto recuperado, optimizando la precisión para el personal operativo.

Seguridad

- **Cifrado de contraseñas:** Al crear o editar usuarios, las claves se transforman en hash antes de guardarse, evitando almacenarlas en texto plano. Por ejemplo, al registrar un operador la contraseña enviada desde el panel se hashea automáticamente antes de persistirse, de modo que nunca se guarda texto plano en la base de datos.

- **Sesiones protegidas y cierre controlado**: El inicio de sesión genera la sesión del usuario y entrega una cookie con token CSRF, mientras el cierre limpia la conversación y termina la sesión de forma controlada. En otras palabras, cada petición se hace en nombre de un usuario identificado, las operaciones que modifican datos exigen un comprobante extra (CSRF) para evitar fraudes, y al cerrar sesión, se revocan las credenciales y se deja todo en un estado coherente (conversaciones cerradas y sesión invalidada) para que nadie pueda actuar con la identidad de un usuario registrado.

- **Barreras de plataforma**: Se mantienen activos los middlewares de seguridad, autenticación, CSRF y anti–clickjacking, además de limitar CORS y orígenes de confianza al dominio autorizado del frontend. Esto quiere decir que, si alguien intenta consumir la API desde otro dominio, o montar la UI en un sitio externo, o enviar un POST sin CSRF, el backend lo deniega por defecto.

- **Control de acceso y validaciones**: Los catálogos y manuales solo admiten administradores autenticados, se exige autenticación y contexto antes de preguntar al asistente, y las cargas de archivos verifican que sean PDFs válidos dentro de transacciones controladas.

Pruebas y validación

- **Pruebas unitarias**:
  - **Pytest**: Corre pruebas rápidas para funciones y reglas del sistema.

- **Pruebas de integración**:
  - **APIClient de Django REST Framework**: Simula llamadas reales a endpoints para ver si el backend responde bien.

- **Pruebas de carga:**
  - **Locust**: Herramienta para pruebas de carga, simula muchos usuarios entrando al sistema para ver cómo responde.

Despliegue

- **Docker**: para encapsular la aplicación y sus dependencias en contenedores, de modo que el entorno de ejecución sea idéntico en desarrollo y producción. Esto

reduce "funciona en mi máquina", facilita la reproducción de fallos y simplifica el despliegue.

Control de versiones y repositorio remoto

- **Git y GitHub**: Se utiliza Git como sistema de control de versiones distribuido para gestionar el historial de cambios, permitir el trabajo en ramas y asegurar la integridad del código fuente durante el desarrollo. Como complemento, se emplea GitHub como plataforma de alojamiento remoto, lo que facilita la centralización del repositorio, la gestión de copias de seguridad en la nube y el uso de herramientas de colaboración.

Tecnologías emergentes

- **LangChain**: Facilita la creación de aplicaciones basadas en grandes modelos de lenguaje (LLM) [1]. La arquitectura usa LangChain para buscar primero los fragmentos de manual más relevantes (con ayuda de embeddings, que son representaciones numéricas del texto) y, con esa información, generar la respuesta. Cada respuesta incluye citas y metadatos del manual de origen, de modo que se puede verificar de dónde salió la información y auditar el proceso.

A modo de resumen, una tabla comparativa con las tecnologías elegidas. Cabe destacar que solo se incluyen las principales categorías de la elección de tecnologías:

Componente	**Opción elegida**	**Alternativas consideradas**	**Ventajas clave de la elección**	**Justificación de elección**
Lenguaje de programación	Python	TypeScript/Node.js; Java (Spring); Go	Sintaxis clara; ecosistema web + IA muy maduro; comunidad grande; rapidez de desarrollo.	Alinea backend y capacidades de IA (RAG/LLM) en un mismo ecosistema, reduciendo tiempo de desarrollo y mantenimiento.
Frameworks (web/API)	Django + Django REST Framework (DRF)	FastAPI; Flask; Spring Boot; Express/NestJS	Admin, auth, ORM y seguridad "de fábrica"; DRF aporta serializadores, permisos y rutas consistentes.	Facilita una entrega rápida de nuevas funciones con buenas prácticas de seguridad y un estándar de API consistente, evitando tener que reconstruir la base técnica desde cero.
Base de datos	PostgreSQL	MySQL/MariaDB; SQLite (dev); MongoDB	ACID robusto; JSONB; extensiones; gran integración con ORM de Django.	Equilibrio entre integridad y flexibilidad (JSONB) con excelente soporte en Django.

Modelo de lenguaje grande	GPT-5-mini	GPT-4o-mini; Claude; Gemini; Llama 3 (local)	Buen multilenguaje; API estable; costo/calidad adecuados; fácil integración.	Calidad suficiente para respuestas y menores costos que otros modelos de lenguaje grandes.

Tabla 2-1 Tabla comparativa y justificativa de tecnologías usadas

Fuente: Elaboración propia

3 Diseño e implementación

Este capítulo describe detalladamente el proceso de diseño y desarrollo del backend del sistema.

3.1 Definición de requerimientos

Antes de proceder con el diseño de la arquitectura y la selección de componentes, es fundamental establecer los requisitos que guiarán el desarrollo del backend. Estos se han derivado del análisis de la problemática de la empresa Indemin y de los objetivos planteados anteriormente, clasificándose en requerimientos funcionales (lo que el sistema debe hacer) y no funcionales (cómo debe comportarse el sistema).

Requerimientos Funcionales (RF)
Estos requerimientos describen las interacciones específicas que el backend debe soportar para satisfacer las necesidades de los usuarios operativos y administradores.

1. **RF-01 Autenticación y Autorización**: El sistema debe permitir a los usuarios iniciar y cerrar sesión de manera segura. Además, debe diferenciar permisos basándose en roles (administrador y usuario básico), restringiendo el acceso a los módulos de gestión exclusivamente a los administradores.

2. **RF-02 Gestión de Usuarios (CRUD)**: El sistema debe proveer una interfaz administrativa (API) que permita crear, leer, actualizar y deshabilitar cuentas de usuario. Esto incluye la asignación de roles y la gestión de credenciales.

3. **RF-03 Gestión de Catálogos y Manuales**: El sistema debe permitir la administración de marcas y equipos, así como la carga, actualización y eliminación de manuales técnicos en formato PDF. El sistema debe ser capaz de procesar estos archivos para su posterior indexación.

4. **RF-04 Procesamiento de Consultas (RAG)**: El sistema debe recibir preguntas en lenguaje natural asociadas a un equipo específico, recuperar los fragmentos de información más relevantes desde la base de datos vectorial y generar una respuesta coherente utilizando un Modelo de Lenguaje Grande (LLM).

5. **RF-05 Trazabilidad y Auditoría**: El sistema debe registrar automáticamente las sesiones de chat, las preguntas realizadas por los usuarios, los fragmentos citados y las respuestas generadas por el asistente para su posterior revisión y mejora continua.

Requerimientos No Funcionales (RNF)
Estos requerimientos definen atributos de calidad, restricciones técnicas y estándares que aseguran la estabilidad y viabilidad del sistema.

1. **RNF-01 Seguridad de Datos:** Las contraseñas de los usuarios deben almacenarse encriptadas (hashing) en la base de datos. La comunicación entre el cliente y el servidor debe protegerse contra ataques comunes, implementando mecanismos como tokens CSRF para la gestión de sesiones.

2. **RNF-02 Interoperabilidad:** El backend debe exponer sus funcionalidades a través de una API REST estándar, utilizando JSON como formato de intercambio de datos, permitiendo así su consumo desacoplado por parte del frontend (React) u otros clientes futuros.

3. **RNF-03 Persistencia y Coherencia:** El sistema debe garantizar la integridad de los datos relacionales (usuarios, metadatos) mediante transacciones ACID proporcionadas por el motor de base de datos PostgreSQL.

4. **RNF-04 Escalabilidad Modular:** La arquitectura debe separar claramente la lógica de negocio, la gestión de vectores y la interfaz de usuario, permitiendo que cada componente pueda ser mantenido o escalado de forma independiente.

3.2 Diseño de Componentes

Una vez definidos los requerimientos que delimitan el alcance del proyecto, es necesario establecer la estructura técnica que permitirá materializar la solución. A continuación, se detalla la arquitectura de software seleccionada junto con la organización de sus componentes. Posteriormente, se abordan de manera específica los patrones de diseño y las buenas prácticas adoptadas para asegurar la calidad del código, finalizando con la estrategia de integración que garantiza el funcionamiento cohesivo del sistema.

3.2.1 Arquitectura y estructura de los componentes.

Para garantizar que el desarrollo del proyecto sea eficiente y ordenado, se han seleccionado una arquitectura y componentes clave que asegurarán la correcta implementación del sistema. Es por este motivo que se detallará la arquitectura elegida, por qué se eligió y cómo es que funciona, además de los componentes involucrados en ella.

Elección de arquitectura:
Como elección se optó por la arquitectura en capas, la justificación es que, al dividir la aplicación en un número no definido de capas, cada una con su rol bien definido, permite separar las responsabilidades del sistema facilitando el **mantenimiento evolutivo** del software. Esto asegura que realizar cambios o mejoras en una capa específica no impacte negativamente en el resto de la aplicación, garantizando la estabilidad del proyecto a largo plazo. La solución propuesta consta de tres capas:

- **Capa de presentación (frontend):** Implementada en React, solo incluye una interfaz web la cual es completamente responsiva para el uso en dispositivos con pantallas más pequeñas.

- **Capa de lógica del negocio (Backend):** Construida en Django para controlar el flujo y las reglas del negocio; LangChain se encarga de la semántica (RAG) y devuelve contenido enriquecido.

- **Capa de persistencia:** Base de datos relacional PostgreSQL para la gestión de los datos claves del negocio (marcas, equipos, manuales, etcétera). Por otro lado, para la gestión de los vectores (embeddings) de los manuales, se emplean índices vectoriales generados con FAISS, lo que permite realizar búsquedas por similitud semántica de alta eficiencia.

Es importante aclarar sobre el uso de FAISS que respecto a persistencia, se separa el archivo binario del índice de sus metadatos y las actualizaciones se realizan de forma atómica mediante operaciones del sistema de archivos en Python (primero se escribe a un archivo temporal y solo cuando todo está correcto se reemplaza el definitivo), con FAISS integrado vía LangChain como almacén vectorial y embeddings generados con modelos de HuggingFace, dentro de un flujo orquestado por Django/DRF; esto evita archivos "a medias" ante fallos, permite cargar el estado rápidamente sin recalcular embeddings y, dado que el índice es un archivo binario grande, minimiza el riesgo de corrupción al reemplazarlo en una sola operación. Además, se valida la coherencia entre el índice y sus metadatos y, si se detectan discrepancias, se reconstruye automáticamente desde los documentos originales. En concurrencia, las operaciones que modifican el índice se ejecutan con acceso exclusivo para impedir escrituras simultáneas, y las reindexaciones (actualizaciones del índice para integrar cambios) se agrupan en una ventana breve de tiempo mediante debounce, que consolida múltiples subidas cercanas en una única actualización; este esquema no bloquea las subidas concurrentes: las cargas se aceptan y se integran en una actualización consolidada del índice, manteniendo la aplicación operativa mientras se evita escribir simultáneamente el archivo binario; si se producen cambios mientras se ejecuta la reindexación, se programa una pasada adicional al finalizar, reduciendo reescrituras innecesarias del archivo binario y evitando solapamientos y trabajo redundante.

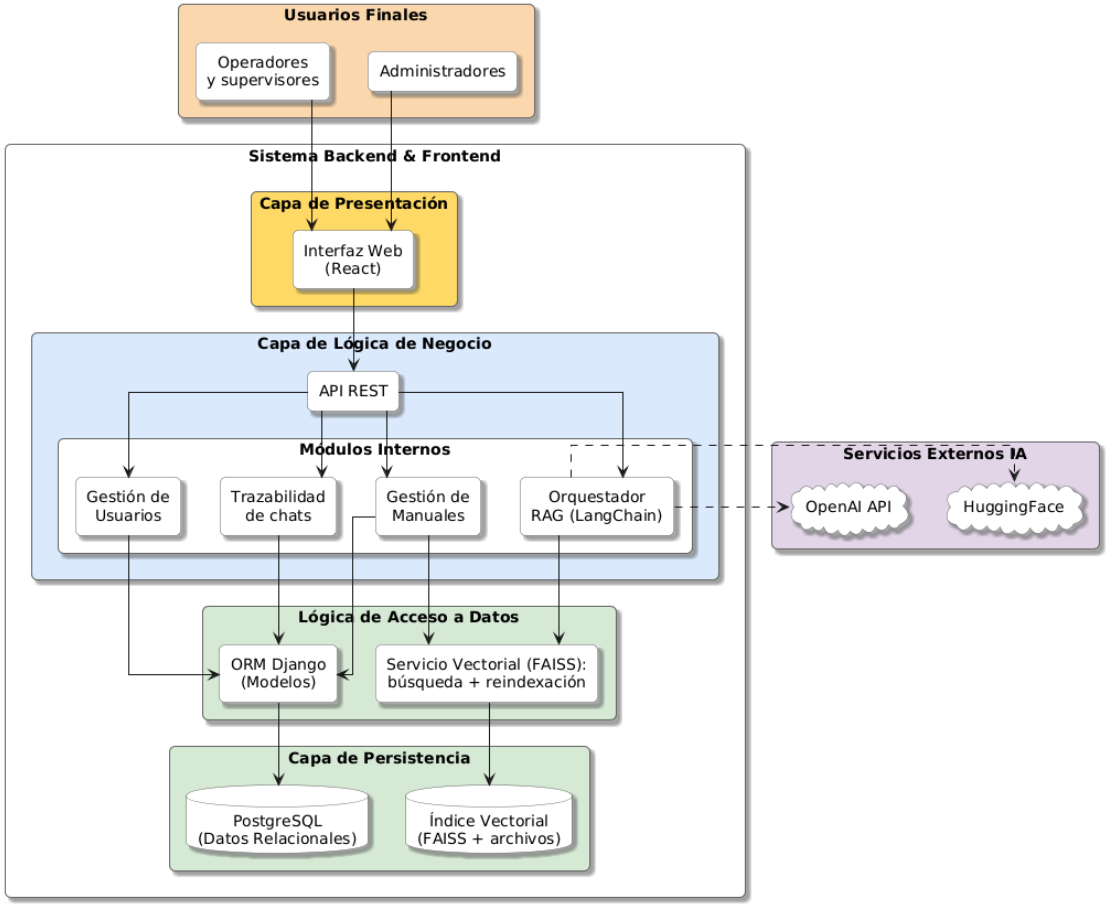

Figura 3-1 Diagrama de arquitectura de 3 capas

Fuente: Elaboración propia

La Figura 3-1 ilustra la arquitectura de software del sistema, diseñada bajo el patrón clásico de tres capas para garantizar la separación de responsabilidades. En el nivel superior, la **capa de presentación** corresponde a la interfaz web desarrollada en React, utilizada por operadores/supervisores y administradores para autenticarse e interactuar con las funcionalidades del asistente y del panel administrativo. Las solicitudes del frontend se canalizan hacia la **capa de lógica de negocio** mediante una API REST, donde se concentran los módulos internos responsables de la gestión de usuarios, la administración de manuales, la trazabilidad de chats y el orquestador RAG (LangChain), el cual coordina la recuperación de información y la generación de respuestas. Para ello, la aplicación accede a los datos a través del ORM de Django y de un servicio vectorial basado en FAISS, encargado tanto de la búsqueda semántica como de la reindexación del conocimiento; adicionalmente, los manuales administrados por el sistema se almacenan como archivos PDF en el servidor (gestión de media), manteniendo su referencia asociada a los registros relacionales. Finalmente, la **capa de persistencia** consolida el almacenamiento en PostgreSQL para la información estructurada y en archivos locales para el índice vectorial (FAISS + metadatos), permitiendo que las consultas del asistente se resuelvan combinando datos relacionales y contenido indexado desde los manuales.

Tomando en cuenta esta lógica de arquitectura apreciable en la figura anterior, para tener una mejor idea acerca de la topología que se utiliza en este proyecto, a continuación, un diagrama de despliegue que describe la arquitectura en capas que fue implementada.

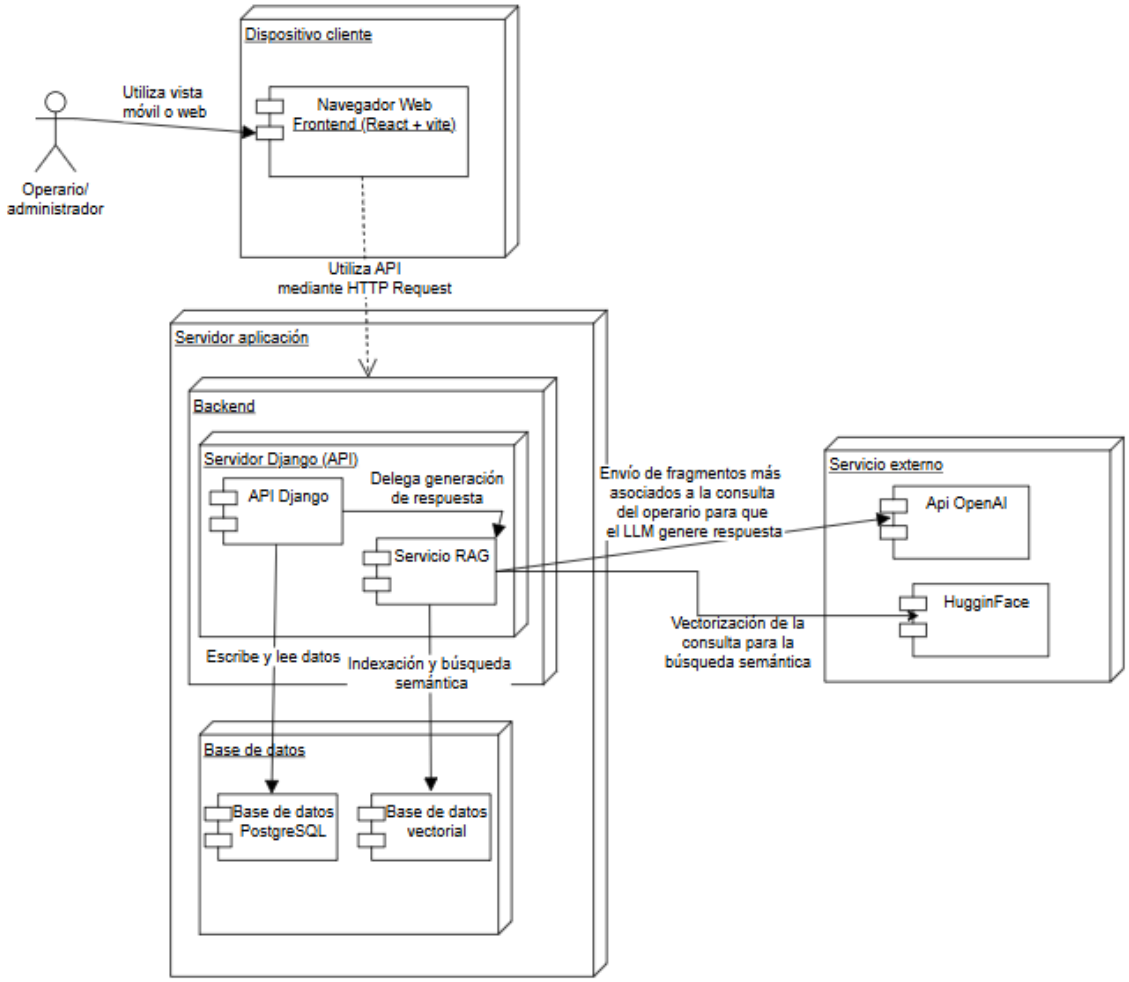

Figura 3-2 Diagrama de despliegue de la arquitectura utilizada

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.

Para facilitar la comprensión de las interacciones dentro del sistema, a continuación, primero se presenta una figura (Figura 3-3) correspondiente a los modelos de datos relacionados que el sistema utiliza, especificando el tipo de dato que contiene cada entidad y las claves foráneas que las relacionan entre sí. Posterior a esto se muestra la figura 3-4 que corresponde al diagrama de clases detallado del backend, el cual ilustra cómo se estructura la lógica de negocio mediante la conexión entre los modelos de datos, los mecanismos de serialización y los controladores de la API.

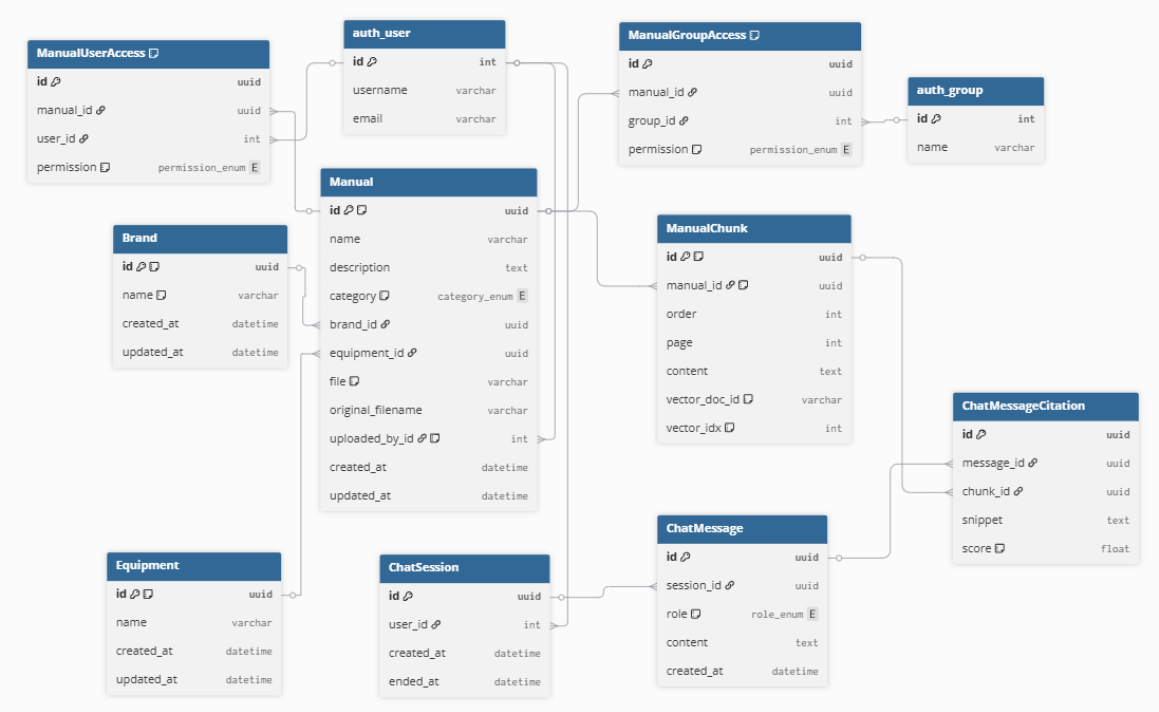

Figura 3-3 Modelo relacional de la base de datos

Fuente: Elaboración propia

El diagrama muestra la base de datos que sostiene el asistente virtual industrial. En el centro está la entidad Manual, que guarda los metadatos del documento (nombre, descripción, categoría, archivo y usuario que lo subió) y se relaciona con Brand y Equipment para normalizar marca y equipo. Cada manual se divide en ManualChunk, trozos ordenados por página y contenido que incluyen referencias al índice vectorial (vector_doc_id y vector_idx) para habilitar búsquedas semánticas en el motor RAG. El control de acceso se gestiona mediante ManualUserAccess y ManualGroupAccess, que asignan permisos a usuarios (auth_user) y grupos (auth_group) sobre cada manual. La interacción con el chatbot se registra en ChatSession (por usuario y tiempos) y en ChatMessage, que almacena cada mensaje con su rol (usuario o asistente) y contenido. Para asegurar trazabilidad y explicabilidad, ChatMessageCitation enlaza mensajes con los chunks citados, guardando el fragmento y una puntuación de relevancia. En conjunto, la estructura permite control de acceso, auditoría y recuperación eficiente de conocimiento: un manual tiene muchos chunks, una sesión tiene muchos mensajes, los mensajes pueden citar múltiples chunks y los permisos se asignan por usuario o grupo a cada manual.

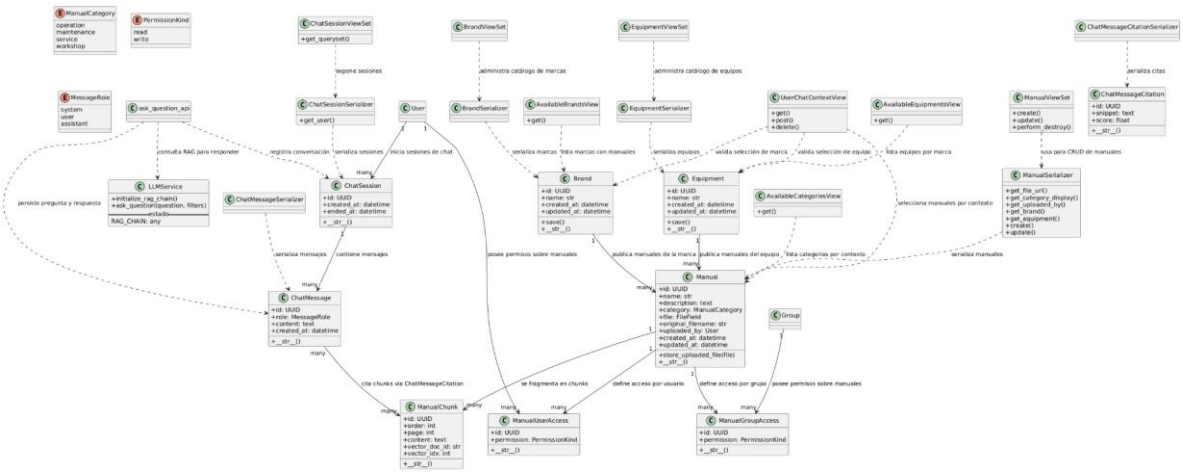

Figura 3-4 Diagrama de clases

Fuente: Elaboración propia

La figura muestra el diagrama de clases del backend y cómo se conectan sus componentes. En el dominio, Brand, Equipment y Manual modelan los manuales técnicos, que se fragmentan en ManualChunk; los permisos se gestionan mediante ManualUserAccess y ManualGroupAccess. La trazabilidad de conversación se registra en ChatSession y ChatMessage, con citas a fragmentos a través de ChatMessageCitation. Los serializers (ManualSerializer, BrandSerializer, EquipmentSerializer, ChatMessageSerializer, ChatSessionSerializer, ChatMessageCitationSerializer) exponen estas entidades a la API, y las vistas/controladores (ManualViewSet, BrandViewSet, EquipmentViewSet, ChatSessionViewSet, AvailableBrandsView, AvailableEquipmentsView, AvailableCategoriesView, UserChatContextView) orquestan el flujo: listar catálogos, validar el contexto del usuario y ejecutar operaciones administrativas. La vista ask_question_api integra el servicio LLMService para consultar el índice RAG, responder preguntas y persistir la conversación, como indican las dependencias y etiquetas de relación ("consulta RAG para responder", "registra conversación", "persiste pregunta y respuesta"). En conjunto, el diagrama clarifica qué clase hace qué, cómo circulan los datos por la API y cómo se vinculan respuesta y evidencias.

En términos de arquitectura, el sistema se estructura mediante módulos funcionales independientes. Esta disposición garantiza la separación de responsabilidades, un aspecto determinante para asegurar tanto la eficiencia operativa como la escalabilidad de la solución propuesta. A continuación, se describen los módulos que integran el sistema, detallando sus componentes lógicos y el rol que desempeñan dentro de la plataforma.

Módulo asistenteVirtualIndustrial (proyecto principal de Django):
Este módulo corresponde al proyecto raíz de Django y actúa como punto central de configuración y coordinación del backend. Desde aquí se define la conexión a la base de datos, las aplicaciones instaladas, la configuración de seguridad, los middlewares, la gestión de archivos estáticos y de medios, entre otros aspectos transversales. Además, expone el enrutamiento principal de URLs, derivando las peticiones entrantes hacia los módulos funcionales accounts y chatBot. Más que implementar lógica de negocio, este módulo funciona como "contenedor" y punto de entrada del sistema, asegurando que todos los demás módulos se integren y funcionen de forma coherente.

- **Componente lógico de configuración global:**
  Agrupa todos los ajustes comunes del proyecto (parámetros de base de datos, integraciones externas, configuración de internacionalización, etc.), centralizando las decisiones técnicas que afectan a todo el backend.

- **Componente lógico de enrutamiento principal:**
  Define las rutas base y delega enrutamiento hacia las aplicaciones especializadas, conectando la capa HTTP con los módulos de negocio sin mezclar responsabilidades.

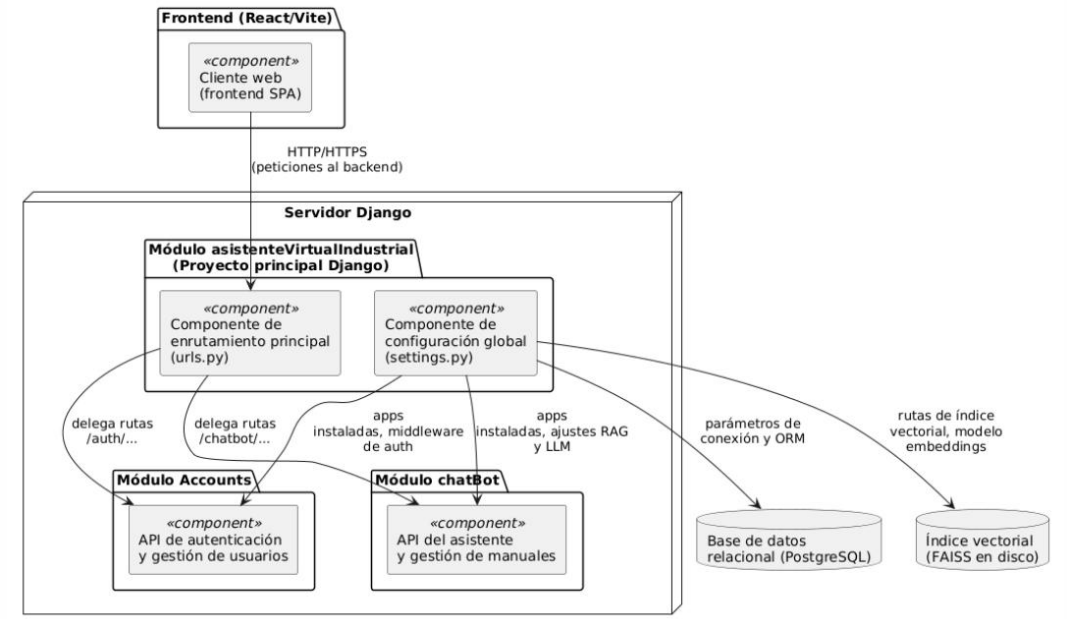

Figura 3-5 Diagrama de componentes del proyecto principal

Fuente: Elaboración propia

Módulo Accounts y sus componentes lógicos:
El módulo accounts concentra la funcionalidad relacionada con la identidad de los usuarios, su autenticación y el control de acceso a los recursos del sistema. Este módulo se apoya en el sistema de usuarios de Django y lo extiende según las necesidades del proyecto, proporcionando una base sólida de seguridad y administración de cuentas.

- **Componente lógico de autenticación y control de acceso:**
  Se encarga del proceso de inicio y cierre de sesión, validación de credenciales y, en caso de utilizarse, manejo de tokens o sesiones seguras. A partir de los roles y grupos asignados a cada usuario, controla qué partes del sistema puede utilizar y qué operaciones puede ejecutar. Este componente permite proteger acciones sensibles (como la administración de manuales o la visualización de trazas de conversación) y garantizar que solo personas autorizadas puedan acceder a información crítica del asistente virtual industrial.

- **Componente lógico de gestión de usuarios:**
  Administra el ciclo de vida de las cuentas dentro de la plataforma. Incluye la creación y actualización de usuarios, el cambio de contraseñas, la modificación de datos básicos (por ejemplo, nombre y correo) y la asignación de grupos o

roles. Desde este componente se facilita que personal administrativo o de TI mantenga un registro ordenado de los usuarios, controle altas y bajas, y se asegure de que los permisos se ajusten a las funciones reales de cada perfil dentro de la organización.

- **Componente lógico de perfiles y administración:**
  Complementa la autenticación incorporando información adicional del usuario (como área, cargo u otros atributos relevantes) y configurando cómo se administran estos datos a través del panel de administración de Django. De esta forma, los responsables del sistema pueden gestionar usuarios, revisar sus permisos y actualizar información de forma visual, sin necesidad de acceder directamente a la base de datos. Este componente contribuye a que la gestión de cuentas sea más amigable y segura.

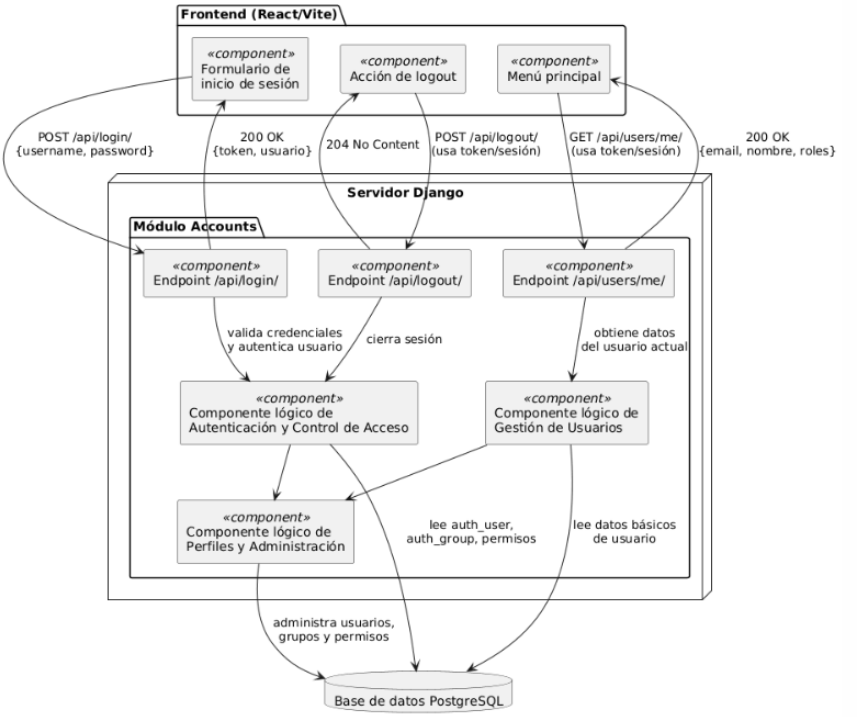

Figura 3-6 Diagrama de componentes del módulo Accounts

Fuente: Elaboración propia

En conjunto, los componentes lógicos de accounts permiten identificar de forma fiable a cada usuario, controlar sus permisos y mantener la información de cuentas en un estado consistente y administrable.

Módulo chatBot y sus componentes lógicos:
El módulo chatBot implementa la lógica principal del asistente virtual industrial. Aquí se gestiona la base de conocimiento construida a partir de manuales técnicos, se controla el acceso a dicha información y se registran las interacciones entre los usuarios y el asistente. Además, este módulo es el punto de unión entre la base de datos relacional y la base de datos vectorial utilizada para búsquedas semánticas.

1. **Componente lógico de gestión de manuales y catálogo técnico:**
   Modela las entidades relacionadas con la documentación técnica, como marcas

(Brand), equipos (Equipment) y manuales (Manual). Permite registrar para cada manual su nombre, descripción, categoría, archivo asociado y usuario que lo subió, además de vincularlo a la marca y al equipo correspondiente. Este componente organiza la información técnica de forma estructurada, facilitando su posterior búsqueda y reutilización por parte del asistente.

2. **Componente lógico de control de acceso a manuales:** Define las reglas que determinan qué usuarios y grupos pueden acceder a cada manual, mediante estructuras como ManualUserAccess y ManualGroupAccess. Gracias a este componente, es posible implementar políticas de seguridad específicas para la documentación, restringiendo ciertos manuales solo a perfiles autorizados (por ejemplo, personal de mantenimiento o supervisores). Al mismo tiempo, asegura que el asistente virtual solo utilice manuales a los que el usuario actual tiene permiso de acceso, manteniendo coherencia entre la experiencia conversacional y las restricciones de la organización.

3. **Componente lógico de fragmentación e indexación semántica (RAG y base vectorial):**
   Se encarga de dividir cada manual en fragmentos más pequeños (ManualChunk), donde se almacena el contenido, la página y el orden de cada sección. A partir de estos fragmentos se generan representaciones vectoriales (embeddings) que se guardan en la base de datos vectorial (FAISS). Los identificadores vector_doc_id y vector_idx almacenados en la base relacional sirven para enlazar cada fragmento con su vector correspondiente. Este componente es esencial para habilitar el enfoque de Retrieval-Augmented Generation (RAG), permitiendo que el asistente encuentre, dentro de grandes manuales técnicos, los párrafos más relevantes para responder a las preguntas de los usuarios.

4. **Componente lógico de conversación y trazabilidad de respuestas:** Modela las sesiones de chat (ChatSession) y los mensajes (ChatMessage) intercambiados entre usuario y asistente. Cada sesión registra el usuario participante y el intervalo temporal de la conversación, mientras que cada mensaje almacena quién lo envió (rol de usuario o asistente), el contenido y la fecha de creación. El modelo ChatMessageCitation vincula las respuestas del asistente con los fragmentos de manual que las respaldan, guardando el trozo de texto citado y una puntuación de relevancia. Este componente permite reconstruir el contexto de cada interacción, analizar el comportamiento del asistente y ofrecer explicaciones sobre qué información de la base de conocimiento se utilizó para generar una respuesta concreta.

5. **Componente lógico de orquestación con el modelo de lenguaje (LLM):**
   Se encarga de preparar y enviar las consultas al modelo de lenguaje externo, implementadas principalmente en servicios como llm_service.py. A partir de la pregunta del usuario y de los fragmentos relevantes recuperados por el componente de indexación semántica, construye el prompt enriquecido con contexto, ajusta parámetros del modelo (como temperatura o longitud máxima de la respuesta) y realiza las llamadas a la API del LLM. Una vez obtenida la respuesta, la devuelve al resto del módulo para que sea registrada como mensaje de chat y, cuando corresponde, vinculada a las citas de manuales utilizadas. De esta forma, el LLM queda encapsulado detrás de este componente y el resto del sistema no depende directamente de los detalles de integración con el proveedor de IA.

6. **Componente lógico de API y servicios del asistente:**
   Expone la funcionalidad del módulo al resto del sistema, y en particular al frontend. A través de vistas y servicios, recibe las consultas de los usuarios, aplica filtros según los manuales a los que tienen acceso, coordina la recuperación de fragmentos relevantes desde la base vectorial, invoca al componente de orquestación con el LLM para generar la respuesta y finalmente registra la conversación y sus citas para mantener trazabilidad. Este componente integra los demás (manuales, permisos, fragmentación, LLM e interacciones) en un único flujo de trabajo, haciendo posible la experiencia de asistente conversacional especializado en documentación industrial.

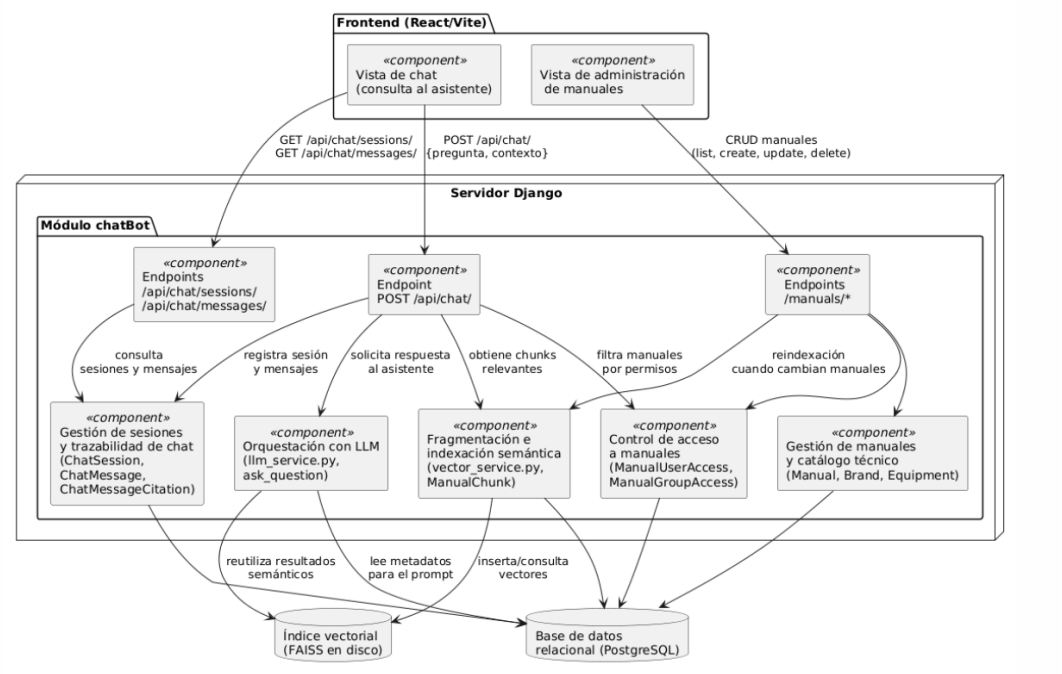

Figura 3-7 Diagrama de componentes del módulo chatBot

Fuente: Elaboración propia

En conjunto, los componentes lógicos del módulo chatBot permiten gestionar de extremo a extremo el conocimiento técnico de la organización, controlar quién puede acceder a qué información y registrar de forma detallada cómo el asistente utiliza esos datos para responder a las consultas de los usuarios.

3.2.2 Uso de buenas prácticas y patrones de diseño.

Para el diseño del sistema, se utilizaron tres patrones clave:

1. **Patrón MVC (Model-View-Controller).**
2. **REST (Estilo Arquitectónico).**
3. **RAG (Patrón de Arquitectura de IA).**

Si bien Django internamente utiliza el patrón MVT (Model-view-Template), debido a que el backend del sistema funciona exclusivamente como API, el frontend se trabaja de forma externa y el flujo dentro del proyecto es que el usuario interactúa con la Vista

(frontend React), que llama al Controlador (vistas/endpoint del backend) y recibe datos para mostrarlos en la interfaz, este proyecto adopta el patrón MVC. A continuación, se explican más a detalle ambos patrones y la justificación de la elección.

Patrón MVC (Model-View-Controller):
El patrón MVC separa la aplicación en tres componentes interconectados:

1. **Modelo**: Gestiona la lógica de negocio de la aplicación y los datos que se encuentran en la base de datos.

2. **Vista**: Se encarga de mostrar al usuario los datos del modelo. Generalmente mediante respuestas JSON o vistas HTML.

3. **Controlador**: Es el intermediario entre modelo y vista. Se encarga de procesar cada solicitud HTTP y enruta a vistas y modelos correspondientes.

Patrón MVT (Model-View-Template):
Se utiliza principalmente en Django, es bastante similar a MVC, pero con algunas diferencias:

1. **Modelo**: Similar al patrón MVC, gestiona la lógica de negocio de la aplicación y los datos que se encuentran en la base de datos.

2. **Vista**: Recibe entradas del usuario, las procesa y en respuesta renderiza un **template** (plantilla). Similar al controlador en MVC.

3. **Template (plantilla)**: Renderiza elementos HTML o de la interfaz de usuario. Se combina frontend y backend en un mismo framework.

Para el desarrollo de este proyecto, como se mencionó anteriormente, el patrón utilizado es MVC debido a las siguientes razones:

- **Desarrollo escalable y fácil de mantener:**
  - Gracias al patrón MVC, el backend pueden trabajarse, mantenerse y escalarse de forma independiente al frontend, facilitando así futuras actualizaciones.

- **Backend y frontend independientes (separados):**
  - Dado que el frontend está desarrollado en React, no se utiliza el sistema de plantillas que dispone Django, sino que se limita al backend a proveer datos a través de APIs REST. De este modo el sistema se alinea con los principios REST al permitir la separación entre frontend (cliente) y backend (servidor).

- **Foco en APIs REST:**
  - En lugar de que las vistas rendericen plantillas HTML, en este caso se utilizan para procesar la lógica de negocio y para el retorno de respuestas en formato JSON.

En síntesis, aunque la arquitectura nativa de Django se rige por el patrón MVT —integrando lógica y presentación—, esta propuesta reorienta su funcionamiento hacia un esquema desacoplado. Se optó por separar las responsabilidades para que el backend opere exclusivamente como una API REST, lo que maximiza la modularidad y permite

que la interfaz de usuario sea gestionada de forma independiente por un frontend externo.

REST (Estilo Arquitectónico):
La transferencia de estado representacional (REST) es una arquitectura de software que impone condiciones sobre cómo debe funcionar una API [14]. En este proyecto se utiliza para la comunicación entre backend y frontend dadas las características que aporta para esta interacción:

1. **Separación cliente–servidor**: el backend Django/DRF solo expone lógica y datos en JSON, mientras que el frontend React se encarga completamente de la interfaz. Esto permite desarrollar, desplegar y escalar cada parte por separado.

2. **Sin estado en el servidor (stateless, en lo posible):** cada petición HTTP incluye toda la información necesaria (token de autenticación, parámetros, cuerpo JSON), lo que hace que el sistema sea más rápido, fácil de escalar y sencillo de reparar si ocurren errores.

3. **Formato de datos estándar (JSON):** las respuestas son objetos JSON que el cliente puede mapear directamente a estados y componentes de React sin transformaciones complejas.

4. **Interfaz uniforme**: el uso de rutas claras y respuestas JSON consistentes simplifica el consumo desde el frontend y reduce el acoplamiento.

La adopción de este enfoque REST ha hecho posible construir un backend liviano y especializado en exponer servicios de datos, que puede ser consumido por el frontend de forma autónoma y adaptable. Gracias a esta separación, la capa de servidor puede evolucionar y escalar sin afectar directamente a la interfaz de usuario, manteniendo una comunicación clara y estable entre ambos lados de la aplicación.

Buenas prácticas de desarrollo:
Respecto a las buenas prácticas respecto al desarrollo del sistema, se detallan las siguientes:

- **Estructura modular:** El sistema está dividido en módulos claros (accounts, chatBot y el proyecto principal asistenteVirtualIndustrial). Cada uno agrupa modelos, vistas, URLs y servicios de un dominio específico (autenticación, gestión de manuales, chat y RAG), lo que facilita el mantenimiento y la evolución independiente de cada parte.

- **Separación estricta frontend–backend:** El backend Django/DRF se expone únicamente como API REST, mientras que la interfaz está desarrollada en React/Vite. Esta separación reduce el acoplamiento, permite desplegar cada capa por separado y hace posible reutilizar el backend desde otros clientes.

- **Uso de Django REST Framework y rutas consistentes:** Se utilizan ViewSet y DefaultRouter para generar rutas REST de forma automática y coherente, junto con respuestas JSON estandarizadas. Esto proporciona una interfaz uniforme para el frontend y simplifica el consumo de la API.

- **Encapsulamiento de lógica compleja en servicios:** La lógica de integración con la base vectorial (FAISS) y con el modelo de lenguaje (LLM) se concentra en

servicios dedicados, evitando que las vistas accedan directamente a detalles de infraestructura. Esto mejora la organización del código, facilita el testing y permite cambiar proveedores o modelos con menor impacto.

- **Manejo explícito de respuestas HTTP y errores:** Las vistas de la API devuelven códigos HTTP claros (por ejemplo, 200 para éxito, 400/401 para errores de validación o autenticación), junto con mensajes JSON comprensibles. Esto simplifica el tratamiento de errores en el frontend y mejora la experiencia de usuario.

3.2.3 Integración con el sistema general.

La integración del backend con el resto del sistema se realiza siguiendo un enfoque de API REST, donde el servidor Django actúa como proveedor de servicios y el frontend desarrollado en React/Vite consume dichos servicios mediante peticiones HTTP y respuestas en formato JSON. Esta separación clara de responsabilidades permite que ambos componentes evolucionen de forma independiente y que otros clientes potenciales (por ejemplo, futuras aplicaciones móviles) puedan reutilizar la misma API sin cambios en la lógica interna del backend.

En la práctica, el frontend se comunica con el backend enviando HTTP requests a endpoints bien definidos, y recibiendo siempre respuestas JSON estructuradas:

Métodos de integración

1. **APIs REST**
   - Cada funcionalidad principal del sistema (autenticación, gestión de usuarios, gestión de manuales, conversación con el asistente) se expone a través de endpoints REST claramente definidos.
   - Las respuestas se estructuran en JSON, entregando datos en un formato estándar que puede ser consumido fácilmente por el frontend u otros clientes.
   - Se utilizan métodos HTTP semánticos (GET, POST, PUT/PATCH, DELETE) para indicar el tipo de operación realizada sobre cada recurso.
2. **Mensajes esperados**
   - El frontend envía solicitudes HTTP al backend incluyendo, según corresponda, parámetros en la URL, cuerpo JSON y cabeceras de autenticación.
   - El backend procesa estas solicitudes y responde con un objeto JSON que incluye:
     - Código de estado HTTP: indica el resultado de la operación (por ejemplo, 200 OK, 201 Created, 400 Bad Request, 401 Unauthorized, 404 Not Found).
     - Mensaje de error o éxito: describe el resultado de la acción, facilitando el manejo de mensajes en el frontend.

- Datos: información solicitada o procesada (por ejemplo, datos de usuario autenticado, listado de manuales, respuesta del asistente y trazabilidad asociada).

Documentación de las llamadas esperadas
A continuación, se describen las llamadas principales entre frontend y backend, organizadas por módulos clave:

1. Llamados relacionados con usuario (módulo accounts).

Descripción llamada	**Endpoint**	**Parámetros (cuerpo JSON)**	**Respuesta esperada**
Iniciar sesión	POST /auth/api/login/	{"username":"string", "password": "string"}	**200**{"message":"Login correcto","user_id":int,"username":"string","is_staff":bool}. **401**{"error":"Credenciales inválidas"}, **405.**
Cerrar sesión	POST /auth/api/logout/	Sin parámetros	**200** {"message":"Sesión cerrada"}; **405** si no es POST.
Listar usuarios	GET /auth/api/users/	Sin parámetros	**200** [{"id":int,"username":"string","email":"string","first_name":"string","last_name":"string","is_staff":bool,"is_active":bool}].
Crear usuario	POST /auth/api/users/	{"username":"string","email":"string?","first_name":"string?","last_name":"string?","is_staff":bool?,"is_active":bool?,"password":"string?"}	**201** {"id":int,"username":"string","email":"string",...,"is_active":bool} (sin exponer password).
Obtener usuario por id	GET /auth/api/users/{id}/	Sin parámetros	**200** {"id":int,"username":"string","email":"string",...}.
Actualizar usuario	PUT|PATCH /auth/api/users/{id}/	{"email":"string?","first_name":"string?","last_name":"string?","is_staff":bool?,"is_active":bool?,"password":"string?"}	**200** {"id":int,"username":"string","email":"string", ...} (si viene password, se hashea)**.**

Eliminar usuario	DELETE /auth/api/users/{id}/	Sin parámetros	**204** (sin contenido).

Tabla 3-1 Llamadas relacionadas al módulo Accounts

Fuente: Elaboración propia

2. Llamados relacionados con el asistente y manuales.

Descripción llamada	**Endpoint**	**Parámetros (cuerpo JSON)**	**Respuesta esperada**
Enviar pregunta al asistente	POST /chat/api/ask/	{"question":"string"}	**200** {"answer":"string", "citations":[{"manual_id":"uuid","manual_name":"string","page":int?,"category":"string","snippet":"string","score":float?}], "safety_instructions":["string"], "context_manuals":[{"id":"uuid","name":"string","file_url":"url","original_filename":"string"}]}; **400** (JSON inválido, sin pregunta o sin contexto), 401, 503 (RAG no inicializado).
Obtener marcas disponibles	GET /chat/api/options/brands/	Sin parámetros	**200** [{"id":"uuid","name":"string"}].
Obtener equipos por marca	GET /chat/api/options/equipments/?brand_id={uuid}	Sin parámetros	**200** [{"id":"uuid","name":"string"}]; error **400.**
Obtener categorías por marca y equipo	GET /chat/api/options/categories/?brand_id={uuid}&equipment_id={uuid}	Sin parámetros	**200** [{"value":"operation|maintenance|service|workshop","label":"Operación|Mantenimiento|Servicio técnico|Taller"}]; error **400** faltan parámetros.
Leer contexto de chat del usuario	GET /chat/api/user-context/	Sin parámetros	**200** {"brand_id":"uuid", "brand_name":"string","equipment_id":"uuid","equipment

			_name":"string","category":"string","category_display":"string","manual_ids":["uuid"],"manual_count":int} o 204 si no hay contexto.
Establecer contexto de chat	POST /chat/api/user-context/	{"brand_id":"uuid","equipment_id":"uuid","category":"operation|maintenance|service|workshop"}	**201** {contexto_mencionado_arriba}; errores **400** (ids inválidos o sin manuales).
Eliminar contexto de chat	DELETE /chat/api/user-context/	Sin cuerpo	**204** (sin contenido).

Tabla 3-2 Llamadas relacionadas a la interacción con el agente del módulo ChatBot

Fuente: Elaboración propia

3. Llamadas relacionadas a los administradores.

Descripción llamada	**Endpoint**	**Parámetros (cuerpo JSON)**	**Respuesta esperada**
Listar sesiones de chat (trazabilidad, admin)	GET /chat/api/chat-sessions/?user={id?}	Sin parámetros	[**200** [{"id":"uuid","user":{"id":int,"username":"string",...},"created_at":"iso","ended_at":"iso|null","messages":[{"id":"uuid","role":"user|assistant|system","content":"string","created_at":"iso","citations":[{"id":"uuid","chunk_id":"uuid","chunk_manual_id":"uuid","chunk_manual_name":"string","snippet":"string","score":float?}]}]}]]. **400** (JSON inválido, sin pregunta o sin contexto), **401**, **503** (RAG no inicializado).
CRUD marcas	GET|POST /chat/api/brands/ GET|PUT|PATCH|DELETE /chat/api/brands/{id}/	Crear/actualizar: {"name":"string"}	**200|201** {"id":"uuid","name":"string","created_at":"iso","updated_at":"iso"}; **204** en DELETE.
CRUD equipos	GET|POST /chat/api/equipments/	Crear/actualizar: {"name":"string"}	.**200|201** {"id":"uuid","name":"string","created_at":"iso","updated_at":"iso"};

	GET|PUT|PATCH|DELETE /chat/api/equipments/{id}/		**204** en DELETE.
CRUD manuales	GET|POST /chat/api/manuals/ GET|PUT|PATCH|DELETE /chat/api/manuals/{id}/	Crear: multipart con file (PDF) y campos JSON: {"name":"string","description":"string?","category":"operation|maintenance|service|workshop","brand_name":"string","equipment_name":"string"} Actualizar: opcional multipart (file) y/o JSON: {"name":"string?","description":"string?","category":"...","brand_name":"string?","equipment_name":"string?"}	**201|200** [{"id":"uuid","name":"string","description":"string","category":"string","category_display":"string","brand": {"id":"uuid","name":"string"},"equipment": {"id":"uuid","name":"string"},"file_url":"url|null","original_filename":"string|null","uploaded_by": {"id":int,"username":"string"}|null,"created_at":"iso","updated_at":"iso","warning":"string?"};] **204** en DELETE.

Tabla 3-3 llamadas relacionadas a la administración en el módulo chatBot

Fuente: Elaboración propia

3.3 Detalles de la codificación y desarrollo

El desarrollo del asistente técnico inteligente se ha estructurado bajo una metodología ágil Scrum, dividiendo el trabajo en iteraciones o sprints. A continuación, se detalla el Product Backlog priorizado por sprint.

Sprint 1: Primer prototipo funcional (MVP)

Objetivo principal:

- Producir primer prototipo funcional del sistema.

Actividades técnicas:

- Configuración inicial Django como framework para el backend.
- Integrar GitHub con el proyecto.
- Creación de APIs REST para la comunicación con el frontend.
- Creación de modelos para la base de datos.

- Desarrollo de login para acceso al sistema.

- Desarrollo de RAG para el flujo de consulta-respuesta con modelo de lenguaje pequeño (SML).

Sprint 2: Implementación de módulos de gestión de usuarios, manuales y trazabilidad

Objetivo principal:
- Implementar módulos de administración y trazabilidad del sistema.

Actividades técnicas:
- Desarrollo de módulo para la gestión de usuarios.

- Desarrollo de módulo para la gestión de manuales.

- Desarrollo de módulo para trazabilidad de chats.

Sprint 3: Integración de LLM y contenerizar el sistema

Objetivo principal:
- Utilizar un LLM de OpenAI para mejorar rendimiento general del asistente virtual y garantizar la reproducibilidad del entorno de desarrollo y producción a través de la contenerización del sistema.

Actividades técnicas:
- Integración de LLM GPT-5 mini mediante API de OpenAI.

- Encapsular el sistema y sus dependencias en un contenedor para que se ejecute de manera consistente en cualquier infraestructura.

Historias de usuario
Para definir los requisitos desde la visión del cliente, utilizamos historias de usuario: explicaciones breves que detallan qué necesita el usuario y por qué es valioso. A continuación, se detallan las historias correspondientes a las funcionalidades esenciales desarrolladas en el proyecto.

- Como operario o administrador, quiero un sistema de inicio de sesión con nombre de usuario y contraseña para ingresar al sistema.

- Como operario, quiero poder realizar consultas en un chat y recibir respuestas generadas a partir de la información almacenada de manuales para resolver problemas sobre los equipos de una forma más rápida

- Como usuario en general, quiero que la interfaz sea atractiva e intuitiva para facilitar la navegación dentro de la aplicación.

- Como administrador, quiero tener acceso a un panel para gestionar usuarios y sus credenciales.

- Como administrador, quiero poder gestionar los manuales dentro del sistema a través de una interfaz para mantener la información del chatbot actualizada.

- Como administrador, quiero poder visualizar un historial de chat por usuario para tener trazabilidad sobre el funcionamiento del chatbot.

- Como usuario, quiero obtener una respuesta segura y clara, para Utilizar la información correctamente y evitar accidentes.

Tomando como base las historias de usuario, los casos de uso formalizan las interacciones técnicas entre los actores y el sistema. Su objetivo es especificar el comportamiento del software necesario para satisfacer cada requerimiento, desglosando tanto el flujo nominal de eventos como las excepciones. A continuación, un diagrama de casos de uso y además, se documentan los casos de uso más significativos implementados en la solución.

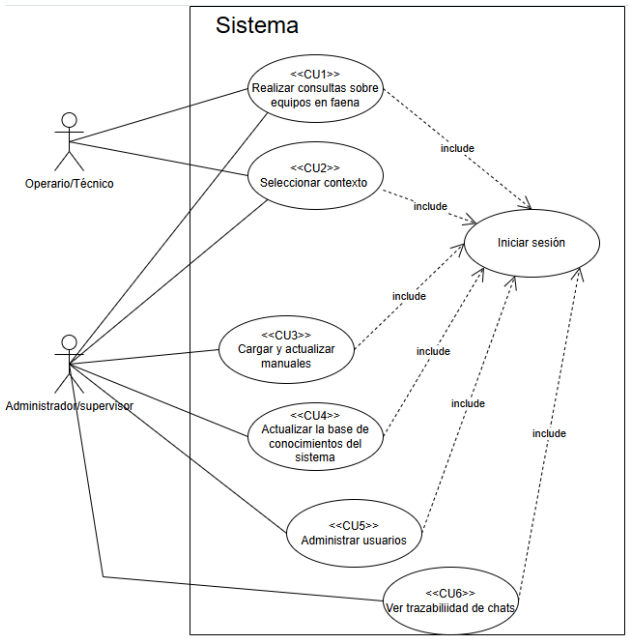

Figura 3-8 Diagrama de casos

Fuente: Elaboración propia

Caso de uso 1: Hacer consultas al chatbot sobre equipos de faena.

Actor principal	**Propósito**	**Flujo principal**	**excepciones**
Operario-Administrador	Permitir al operario o administrador hacer consultas sobre los equipos que utiliza en faenas con la finalidad de resolver un problema mediante la información de los manuales.	1. Usuario inicia sesión. 2. El sistema valida las credenciales ingresadas por el usuario. 3. El usuario ingresa a una ventana donde se le pide elegir marca de equipo, equipo y categoría de manual para generar contexto	2da Si las credenciales no son correctas, se retorna un error.

		4. El sistema filtra los manuales que tengan las características ingresadas en el paso anterior. 5. El usuario hace su consulta mediante lenguaje natural. 6. El sistema envía respuesta al usuario referenciando los fragmentos que uso para responder y dando acceso a los manuales usados para responder.	

Tabla 3-4 Caso de uso 1

Fuente: Elaboración propia

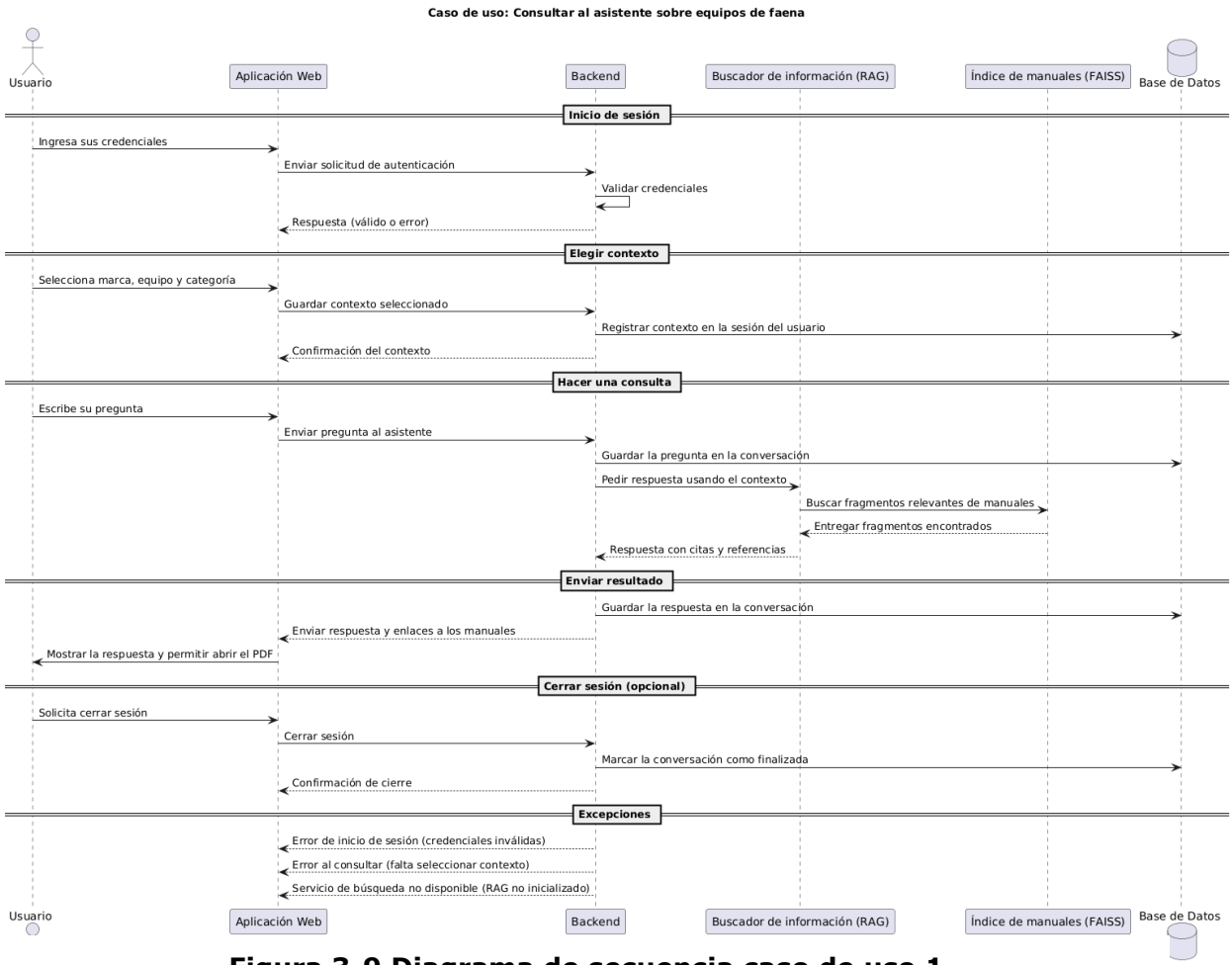

Figura 3-9 Diagrama de secuencia caso de uso 1

Fuente: Elaboración propia

Caso de uso 3: Carga y actualización de manuales en PDF

Actor principal	Propósito	Flujo principal	excepciones
Administrador	Cargar documentos al sistema para expandir su base de conocimientos.	1. Usuario inicia sesión. 2. El sistema valida las credenciales ingresadas. 3. El usuario accede al panel de administración de manuales. 4. El usuario crea un manual llenando el formulario requerido. 5. El sistema guarda el nuevo manual en la base de datos relacional y vectorial.	2da Si las credenciales no son correctas, se retorna un error. 3ra El acceso a este panel es exclusivo para administradores. 4ta En el caso de campos inválidos, se entrega un error y no se realiza la operación.

Tabla 3-5 Caso de uso 3

Fuente Elaboración propia

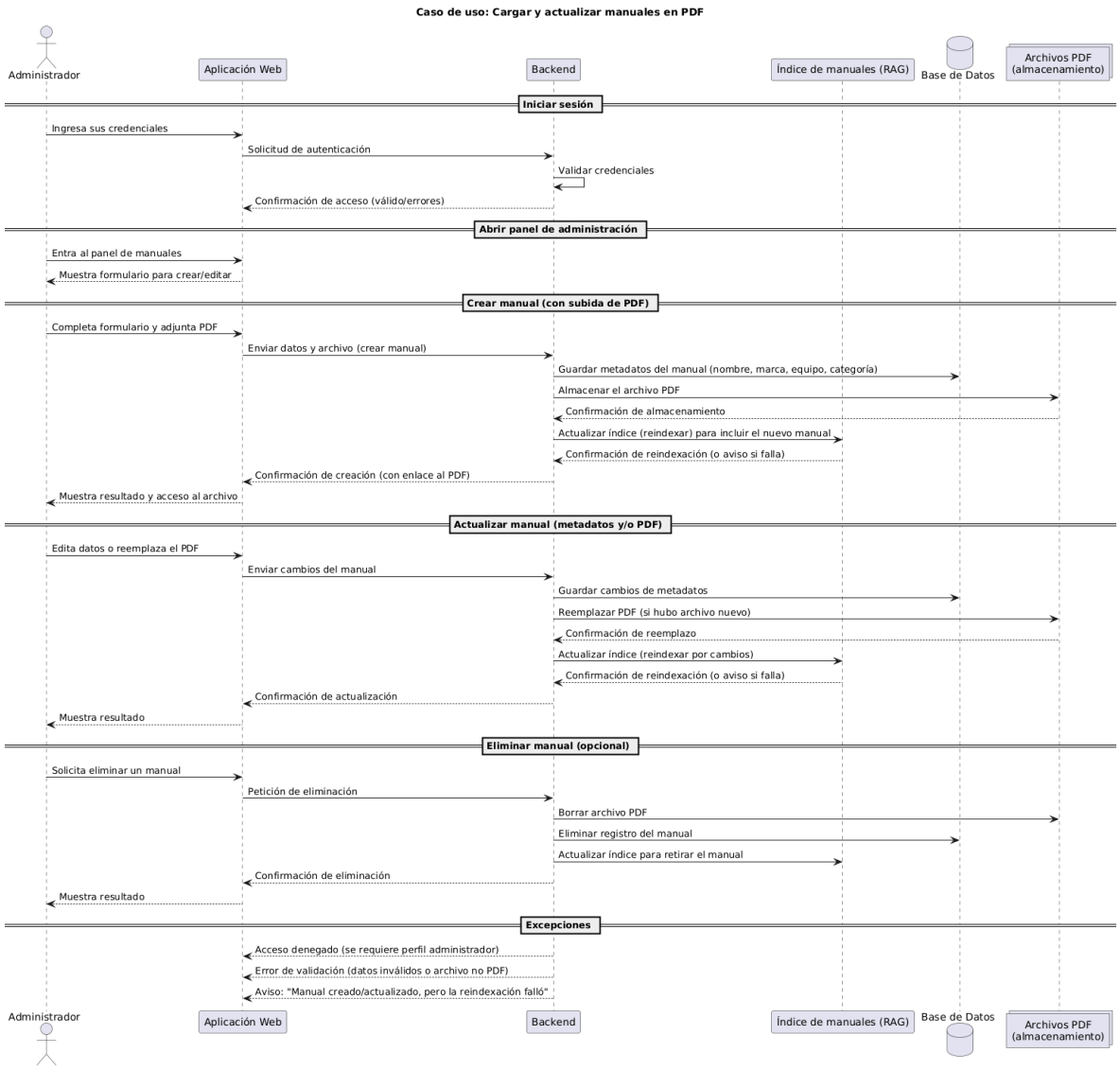

Figura 3-10 Diagrama de secuencia caso de uso 3

Fuente: Elaboración propia

Caso de uso 6: Auditoría y trazabilidad de conversaciones

Actor principal	Propósito	Flujo principal	excepciones
Administrador	Permitir a usuarios administradores revisar sesiones de chat con	1. Usuario inicia sesión. 2. El sistema valida las credenciales ingresadas.	2da Si las credenciales no son correctas, se retorna un error. 3ra El acceso a este panel es

	mensajes y citas para trazabilidad.	3. El usuario accede al panel de trazabilidad de chats. 4. El usuario puede visualizar todos los mensajes del chat de cualquier sesión, ordenados por usuario y fecha.	exclusivo para administradores.

Tabla 3-6 Diagrama de caso de uso 6

Fuente. Elaboración propia

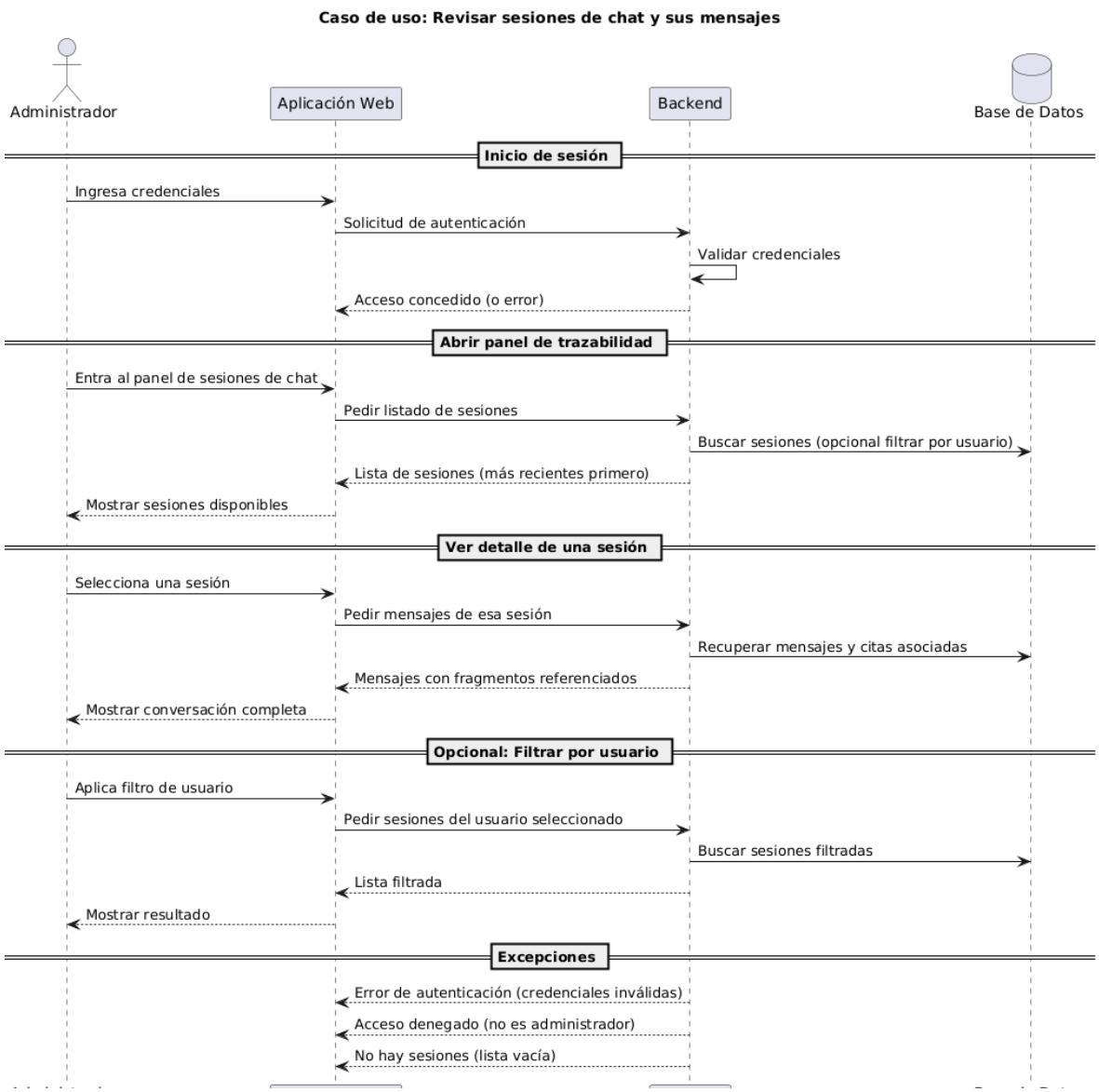

Figura 3-11 Diagrama de secuencia caso de uso 6

Fuente: Elaboración propia

Control de versiones
Para el control de versiones de este proyecto se utilizó **Git**, siguiendo la siguiente estructura de ramas:

- **Main**: Rama principal, versión más nueva y estable.
- **Backend**: Rama para el desarrollo de la lógica de negocio e integración del LLM al sistema.
- **Desarrollo**: Rama para la integración de cambios o nuevas funcionalidades antes de subirlos a Main.
- **Vector-db**: Rama para Procesamiento de manuales: extracción de texto y OCR, limpieza e indexación de fragmentos en FAISS.
- **Frontend**: Rama para integración de cambios relacionados al frontend del sistema.
- **Docker**: Rama para despliegue utilizando contenedores.

3.4 Pruebas y Validación

Con el objetivo de asegurar la **integridad y operatividad** del funcionamiento del backend, se ha ejecutado un riguroso plan de validación. Este proceso abarca la verificación de módulos aislados, la interacción entre sistemas y el análisis de eficiencia bajo carga. Las metodologías empleadas se detallan a continuación.

3.4.1 Estrategias de testing aplicadas a los componentes.

1. **Pruebas unitarias:**
   - **Objetivo:** Comprobar que cada parte del sistema (autenticación, contexto, catálogo, carga de manuales y registro de conversaciones) funciona bien por separado.
   - **Componentes probados:**
     - **Autenticación**:
       - Login correcto crea una sesión de chat activa.
       - Login con credenciales inválidas devuelve un error claro.
       - Logout cierra la sesión y marca el fin de la conversación.
     - **Contexto de consulta**:
       - Crear contexto con marca, equipo y categoría guarda la selección del usuario.
       - Pedir el contexto devuelve la selección actual; si no existe, se avisa.
       - Hacer una pregunta sin contexto devuelve un error pidiendo completar la selección.
     - **Catálogo disponible**:
       - Listar marcas con manuales cargados.
       - Listar equipos filtrados por una marca.

        - Listar categorías filtradas por marca y equipo.

    - **Carga y actualización de manuales (admin)**:
        - Crear manual con PDF válido guarda el archivo y los datos.
        - Actualizar manual modifica metadatos y, si cambia el PDF, avisa de reindexación.
        - Eliminar manual borra el archivo y lo retira del índice.

    - **Trazabilidad (admin)**:
        - Ver listado de sesiones de chat con sus mensajes y citas, ordenado del más reciente al más antiguo.
        - Asegurar que solo administradores pueden acceder a esta vista.

2. **Herramientas utilizadas**:
    - Módulo de pruebas integrado de Django (unittest) con cliente de API para simular llamadas.
    - Simulación (mock) de la respuesta del asistente y del proceso de reindexación para evitar dependencias externas y acelerar las pruebas.

3. **Resultados**:
    - Se ejecutaron 9 pruebas y todas pasaron ("OK").

2. Pruebas de integración:

- **Objetivo:** Validar que todo el sistema funcione en conjunto: inicio de sesión, selección de contexto, consultas al asistente, administración de manuales y trazabilidad de conversaciones.

- **Componentes probados:**
    - Backend ↔ Base de datos: creación de sesiones de chat, mensajes, manuales y catálogos.
    - Autenticación y permisos: inicio/cierre de sesión, acceso de administradores a rutas protegidas.
    - Endpoints REST y reglas de negocio: validar que cada ruta responda con el código y datos correctos.
    - Almacenamiento de archivos: subida/reemplazo/eliminación de PDFs de manuales.
    - Registro de trazas: guardar mensajes del usuario y del asistente en cada sesión.

- **Herramientas utilizadas:**
    - Django + unittest con APITestCase para pruebas automáticas de endpoints.
    - Simulación (mock) del asistente (RAG) y de la reindexación para evitar servicios externos y acelerar.
    - (Opcional) Postman para comprobaciones manuales puntuales de las rutas principales.

- **Metodología:**
    - Cada escenario define: precondiciones, acción (llamada HTTP), respuesta esperada (código y JSON) y efectos en la base de datos.

- Se usan datos mínimos (una marca, un equipo y un manual) y una carpeta temporal para PDFs, sin tocar archivos reales.
- Se documentan casos felices y casos con error (por ejemplo, sin contexto o sin permisos).

- **Resultados obtenidos:**
  - Se ejecutaron 8 pruebas y todas pasaron (OK).

3. **Pruebas funcionales:**
   - **Objetivo**: Validar que los flujos completos de la plataforma cumplen los requerimientos de negocio, seguridad y trazabilidad definidos.

   - **Componentes probados:**
     - Autenticación por sesión y cierre de sesión.
     - Consulta asistida (chat) con contexto del usuario y citas de manuales.
     - Gestión de manuales PDF: carga, actualización, eliminación y reindexación.
     - Catálogos de marcas y equipos (opciones disponibles).
     - Gestión del contexto de usuario (agregar, listar, eliminar).
     - Auditoría y trazabilidad de sesiones de chat.
     - Control de permisos para operaciones administrativas.

   - **Métodos utilizados**:
     - Pruebas manuales end-to-end desde la interfaz y pruebas directas de API.
     - Validación de restricciones de acceso según perfil (administrador/usuario).
     - Verificación de respuestas y estructura JSON devuelta por la API.

   - **Ejemplos**:
     - Un usuario inicia sesión, define su contexto y realiza una consulta; recibe respuesta con citas y manuales relacionados.
     - Un usuario intenta consultar sin contexto definido; el sistema responde con error indicando el requerimiento de contexto.
     - Un administrador carga un manual PDF y verifica que quede disponible en las consultas (reindexación aplicada).
     - Un usuario sin permisos intenta crear/editar/eliminar manuales; el sistema bloquea la acción por falta de autorización.
     - Un usuario consulta las opciones de marcas y equipos para configurar su contexto.
     - Un administrador revisa el historial de sesiones y mensajes para fines de auditoría.

4. **Pruebas de rendimiento:**
   - **Objetivo**: Evaluar la capacidad del backend para manejar múltiples peticiones simultáneas bajo distintas condiciones de carga, manteniendo tiempos de respuesta aceptables y baja tasa de errores.

   - **Componentes probados:**
     - Autenticación por sesión (login).
     - Lectura de catálogos: marcas, equipos y categorías.
     - Fijación de contexto de chat (requiere CSRF).

- Consultas al asistente (RAG) con contexto.

- **Herramientas utilizadas**:
  - **Locust para simular alto tráfico y medir**:
    - Tiempos de respuesta promedio y percentiles (p50, p95, p99).
    - Tasa de errores por endpoint.
    - Throughput (solicitudes por segundo).

- **Metodología utilizada:**
  - Generación de reportes detallados por endpoint para identificar cuellos de botella.
  - Configuración de escenarios con distintos niveles de usuarios simultáneos (ramp-up y sostenido).
  - Validación de sesión/CSRF y estructura JSON en respuestas durante la carga.

- **Escenario ejecutado**:
  - Con el objetivo de establecer una línea base de rendimiento (baseline) y validar la estabilidad de la arquitectura lógica del sistema, las pruebas de carga se ejecutaron en un **entorno local de desarrollo controlado**.

  - Para garantizar la reproducibilidad de los resultados obtenidos y la transparencia del escenario de pruebas, a continuación, se detallan las especificaciones técnicas exactas del equipo anfitrión (host) donde se orquestaron los contenedores del sistema:

    - **Procesador (CPU)**: Intel® Core™ i7-8750H (6 núcleos físicos, 12 procesadores lógicos) @ 2.20 GHz (Frecuencia base).
    - **Memoria RAM**: 16 GB DDR4 @ 2667 MHz (Formato SODIMM).
    - **Almacenamiento Principal**: SSD NVMe Kingston SA2000M8500G de 500GB.

  - Incremento de carga en pasos de 2 usuarios hasta alcanzar 100 usuarios simultáneos, invocando de forma continua:
    - /auth/api/login/
    - /chat/api/options/brands/
    - /chat/api/options/equipments/?brand_id={id}
    - /chat/api/options/categories/?brand_id={id}&equipment_id={id}
    - /chat/api/user-context/ (POST con X-CSRFToken)
    - /chat/api/ask/ (POST con pregunta)

- **Resultados:**

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
POST	auth_login	100	0	4700	11000	12000	5450.3	3059	12399	82	0	0
POST	chat_ask	314	33	15000	37000	37000	15970.09	62	38194	2056.95	9.6	0.5
GET	options_brands	221	1	280	1900	2900	545.23	71	3362	1257.61	3.3	0
GET	options_categories	216	0	260	1200	1800	388.22	67	2384	49	3.1	0
GET	options_equipments	216	0	270	1400	2900	448.44	67	3595	62	3.2	0
POST	user_context	99	2	210	740	1200	289.66	91	1153	5610.95	0	0
	Aggregated	1166	36	510	28000	37000	5051.06	62	38194	1296.29	19.2	0.5

Figura 3-12 Estadísticas generales de rendimiento

Fuente: Elaboración propia

Type	Name	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)
POST	auth_login	4700	11000	12000	5450.3	3059	12399
POST	chat_ask	15000	37000	37000	15970.09	62	38194
GET	options_brands	280	1900	2900	545.23	71	3362
GET	options_categories	260	1200	1800	388.22	67	2384
GET	options_equipments	270	1400	2900	448.44	67	3595
POST	user_context	210	740	1200	289.66	91	1153
	Aggregated	510	28000	37000	5051.06	62	38194

Figura 3-13 Estadísticas de tiempo

Fuente: Elaboración propia

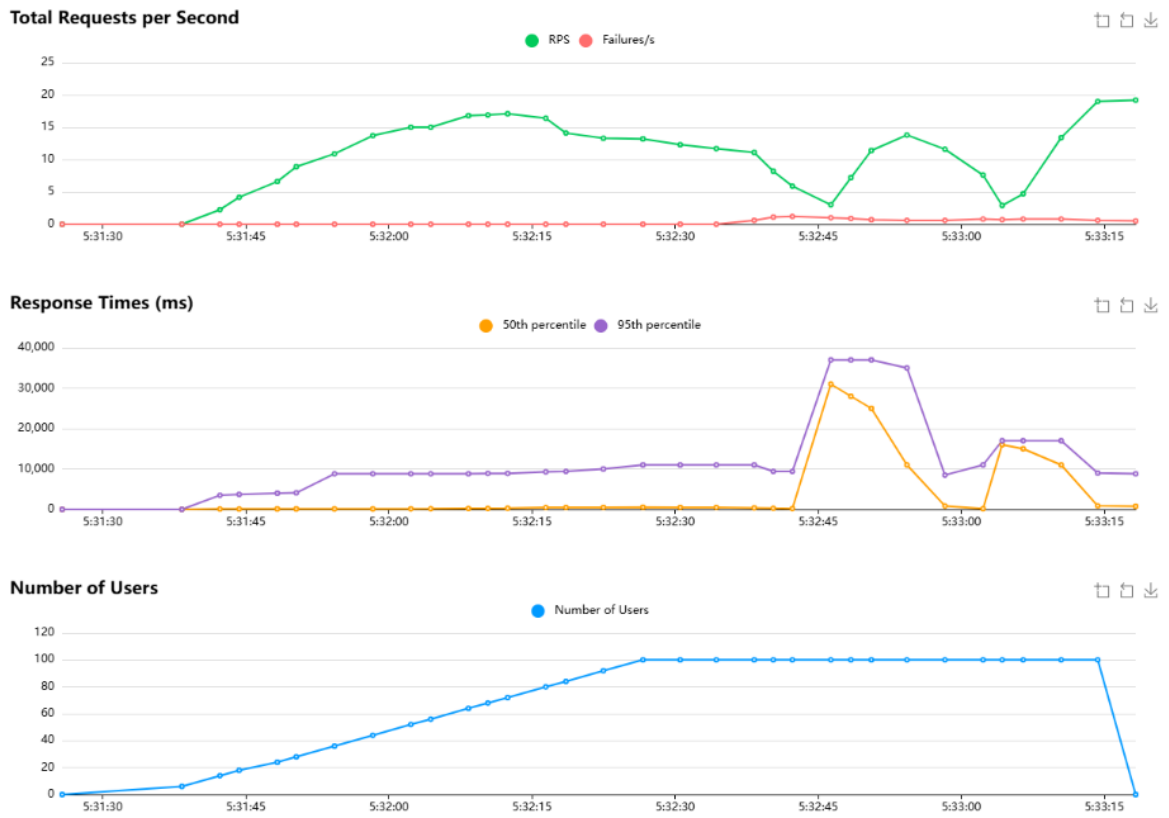

Figura 3-14 Gráficas de estadísticas generales

Fuente: Elaboración propia

3.4.2 Resolución de bugs y optimización

De las pruebas realizadas al backend para detectar problemas durante el desarrollo, la mayoría de estos problemas se detectaron mediante:

1. **Pruebas unitarias**: Validan reglas de negocio y modelos.
   - Ejemplo: detección de errores en autenticación (códigos 401/403/400 correctos), manejo de contexto de chat y validación de categorías sin manuales.

2. **Pruebas de integración**: Verifican la interacción entre componentes.
   - Ejemplo: flujo completo login → opciones (marcas/equipos/categorías) → fijación de contexto (CSRF) → consulta al asistente → auditoría de sesión y mensajes.

3. **Pruebas de rendimiento:** Carga con Locust y análisis de disponibilidad del servidor.
   - Ejemplo: medición de p50/p95/p99 por endpoint; identificación de cuello de botella en POST /chat/api/ask/ (RAG/LLM) y latencias más altas en POST /auth/api/login/ bajo ramp-up.

Metodología aplicada

1. **Reproducción de errores**: Uso de manage.py test (APITestCase) y colecciones Postman para confirmar fallos en endpoints; escenarios negativos (sin sesión, sin contexto, payload inválido).

2. **Depuración**: Instrumentación de logs en vistas críticas (consulta RAG, reindexación), revisión de trazas de sesión y mensajes; análisis de tiempos por fase (DB, FAISS, LLM).

3.5 Integración con otros componentes

En este proyecto, la integración con otros componentes de software se centra en la incorporación de un motor de inteligencia artificial externo y en el procesamiento semántico local de los documentos técnicos, garantizando que la generación de respuestas mantenga coherencia y trazabilidad. La capa de conversación utiliza un modelo de lenguaje de OpenAI (GPT-5 mini) para enriquecer las consultas de los usuarios, mientras que un índice vectorial residente en el servidor permite recuperar fragmentos relevantes de los manuales previamente procesados. El flujo se estructura de modo que la búsqueda contextual (vectorización y filtrado por catálogo) ocurra antes de invocar al modelo, reduciendo coste y latencia sin exponer detalles internos al cliente.

Para el tratamiento de archivos se emplea extracción y limpieza de contenido de documentos PDF mediante herramientas especializadas de parsing y OCR, cuyos resultados se dividen en segmentos normalizados y se transforman en vectores; este material alimenta el índice interno que actúa como puente entre la información almacenada y la capa de razonamiento del modelo. Las claves y credenciales del proveedor de IA se resguardan mediante variables de entorno y nunca se devuelven al cliente, manteniendo separación clara entre lógica de negocio y configuración sensible.

El despliegue se orienta a entornos escalables con servidores preparados para concurrencia asíncrona y la posibilidad futura de incorporar contenedores para aislar procesos de indexación frente al servicio interactivo. Aunque el sistema actualmente no consume pasarelas de pago ni otros servicios transaccionales externos, su diseño desacoplado —basado en contratos JSON estables y una orquestación que distingue consulta, contexto y trazabilidad— permite integrar sin fricción aplicaciones web, clientes móviles u otros módulos analíticos (por ejemplo, una cola de tareas para reindexaciones masivas o un servicio predictivo). La modularidad lograda asegura que nuevas capacidades (streaming de respuestas, cache distribuida, integración con sistemas de monitoreo) puedan añadirse sin comprometer la operación central ni la seguridad de los datos.

4 Conclusiones

Durante el desarrollo del proyecto se constató que la efectividad del sistema dependía menos de sustituir modelos y más de orquestar adecuadamente sus componentes. La separación entre recuperación semántica, generación con IA y trazabilidad permitió construir un flujo robusto, cuya integridad fue validada mediante la ejecución exitosa de 9 pruebas unitarias y 8 pruebas de integración, alcanzando un 100% de operatividad en los módulos evaluados. La autenticación basada en sesión resultó suficiente para la interacción web, y el uso de contexto (marca, equipo y categoría) demostró que la calidad de los metadatos incide directamente en la pertinencia de las respuestas.

En retrospectiva, habría sido conveniente desacoplar antes las operaciones más costosas (búsqueda de fragmentos relevantes en el índice vectorial, construcción del contexto RAG e invocación al LLM) del ciclo sincrónico petición–respuesta, migrándolas a ejecución asíncrona o a una cola de tareas. Esto reduciría la retención de recursos del servidor durante estas etapas y mejoraría el comportamiento bajo concurrencia.

El principal desafío técnico se observó en la latencia del asistente bajo concurrencia, donde los percentiles altos crecían cuando no se controlaba el tamaño del contexto ni el número de pasajes, sin embargo, bajo condiciones de 100 usuarios simultáneos se logró un tiempo p95 de 37 segundos. La normalización de la extracción de texto (PDF y OCR) representó otro reto, ya que se debía garantizar consistencia en el índice mediante limpieza, segmentación y etiquetado uniforme. Estos hallazgos subrayaron la necesidad de medir por etapas (búsqueda semántica y generación del modelo) para identificar cuellos de botella reales.

Entre los logros del proyecto destaca la implementación de un asistente capaz de responder con citas verificables desde manuales, un pipeline documental que transforma PDFs en un índice vectorial reutilizable y una suite de pruebas que abarca validaciones unitarias, integración y rendimiento. La trazabilidad de sesiones y mensajes aporta insumos para auditoría y mejora continua, fortaleciendo la confiabilidad del sistema.

Respecto del impacto, el sistema contribuye a reducir los tiempos de búsqueda y a aumentar la productividad al ofrecer respuestas contextualizadas con respaldo documental. Con un tiempo de respuesta promedio de aproximadamente 16 segundos, el asistente permite al operario obtener soluciones inmediatas sin tener que esperar la disponibilidad de terceros que no tienen dedicación exclusiva a la resolución de incidencias y poseen otras responsabilidades. Para la disciplina informática, el proyecto constituye un caso práctico de RAG con buenas prácticas de ingeniería: contratos claros (qué datos se envían y qué datos se reciben, con estructura y tipos claros, para que ambos lados funcionen incluso si internamente cambian), observabilidad por etapas y desacoplamiento de responsabilidades que habilitan la evolución sin reescrituras costosas.

De cara al futuro, se propone mejorar la experiencia percibida mediante streaming de respuestas, colas asíncronas para picos de trabajo y despliegues con más workers en entornos ASGI. Se recomienda optimizar el RAG (ajustes de chunking, filtros y caché de resultados frecuentes), exponer métricas por etapa hacia sistemas de monitoreo y mantener la abstracción necesaria para alternar el proveedor de IA o el almacén vectorial sin fricción. En síntesis, se concluye que diseñar para medir desde el inicio, junto con contratos estables y módulos intercambiables, facilita priorizar mejoras que efectivamente elevan la calidad y la escalabilidad del sistema.

5 Agradecimientos

Quisiera expresar mi profundo agradecimiento a mi familia y amistades por el inmenso apoyo brindado durante todos estos años. A pesar de las dificultades, su presencia incondicional fue fundamental para mí. Sin ustedes, jamás habría alcanzado este hito, el cual será el cimiento sobre el que construiré mis próximas metas.

6 Referencias

[1] AWS, «aws.amazon.com,» [En línea]. Available: https://aws.amazon.com/es/what-is/langchain/.

[2] S. Hoffstaetter, «pypi.org,» [En línea]. Available: https://pypi.org/project/pytesseract/.

[3] C. Bilbao, «Diseño e implementación de una base de datos vectorial para la búsqueda semántica en manuales técnicos,» 2025.

[4] K. Schwaber y J. Sutherland, «scrumguides.org,» 2020. [En línea]. Available: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Spanish-European.pdf.

[5] E. e. d. IONOS, «ionos.com,» 06 07 2024. [En línea]. Available: https://www.ionos.com/es-us/digitalguide/paginas-web/creacion-de-paginas-web/que-es-el-backend/#content-en-qu-consiste-un-backend.

[6] Zazz, [En línea]. Available: https://www.zazz.io/glossary/backend-development#:~:text=Historia/Origen%2D%20%C2%BFQu%C3%A9%20significa,ampliamente%20utilizados%20en%20la%20actualidad..

[7] AWS, «aws.amazon.com,» [En línea]. Available: https://aws.amazon.com/es/compare/the-difference-between-frontend-and-backend/.

[8] Amazon Web Services, «aws.amazon.com,» [En línea]. Available: https://aws.amazon.com/es/ai/generative-ai/?trk=7487c20c-a7d1-4b37-b755-26b138534f21&sc_channel=ps&ef_id=Cj0KCQjwrojHBhDdARIsAJdEJ_cHcQnrFrRYWs0FwYSpkXbwauBjq53y0oPw34X-wXa9al4t7MY9EuoaAuMxEALw_wcB:G:s&s_kwcid=AL!4422!3!770465782004!e!!g!!generative%20ai!.

[9] Google Cloud, [En línea]. Available: https://cloud.google.com/discover/what-is-serverless-architecture?hl=es.

[10] Google Cloud, «cloud.google.com,» [En línea]. Available: https://cloud.google.com/learn/what-are-containers?hl=es.

[11] Amazon Web Services, «aws.amazon.com,» [En línea]. Available: https://aws.amazon.com/es/what-is/retrieval-augmented-generation/.

[12] O. J. B. Iturralde, «reactiveprogramming.io,» 2021. [En línea]. Available: https://reactiveprogramming.io/blog/es/estilos-arquitectonicos/capas.

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

[14] Amazon Web Services, «aws.amazon.com,» 2026. [En línea]. Available: https://aws.amazon.com/es/what-is/restful-api/.

7 Anexo

Repositorio de documentación del proyecto (Casos de uso).