TEAM COPILOT

José Antonio Jiménez Carmona

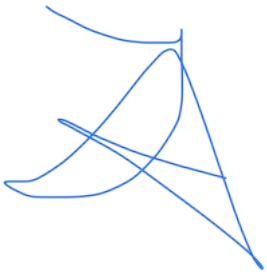
Trabajo Fin de Máster

Universidad Pontificia Comillas
Máster Universitario en Big Data

Director: Juan Antonio Buero Viana

Fdo. author:
José Antonio Jiménez Carmona

Fdo. director (Visto Bueno):
Juan Antonio Buero Viana



Madrid
Junio 2025

Índice general

1. Introducción **5**
1.1. Motivación . . . 6
1.2. Objetivos generales . . . 6

2. Contexto y Estado del Arte **7**
2.1. Contexto . . . 7
2.2. Estado del Arte . . . 7
2.2.1. Onyx . . . 7
2.2.2. Anything LLM . . . 8
2.2.3. Private GPT . . . 8
2.3. Comparativa y Conclusiones . . . 9

3. Planificación y Costes **11**
3.1. Objetivos específicos . . . 11
3.2. Metodología de desarrollo . . . 12
3.3. Planificación . . . 12
3.4. Organización de personal del proyecto . . . 13
3.5. Hardware y software del proyecto . . . 13
3.6. Tecnologías a emplear . . . 14
3.7. Presupuesto del proyecto . . . 15
3.7.1. Coste del personal . . . 15
3.7.2. Coste del hardware y software . . . 16
3.7.3. Resumen del presupuesto . . . 16
3.8. Desviación del proyecto . . . 16

4. Análisis de requisitos **19**
4.1. Participantes . . . 19
4.1.1. Organizaciones . . . 19
4.1.2. Personas . . . 19
4.2. Objetivos . . . 19
4.3. Requisitos de información . . . 25
4.4. Requisitos funcionales . . . 26
4.5. Requisitos no funcionales . . . 31
4.6. Casos de uso . . . 35
4.6.1. Actores . . . 35
4.6.2. Casos de uso . . . 36

5. Diseño **45**
5.1. Modelo de datos . . . 45
5.2. Arquitectura . . . 46
5.2.1. API . . . 46
5.2.2. Base de datos PostgreSQL . . . 46
5.2.3. Interfaz web de usuario . . . 46
5.2.4. Interfaz web de documentación . . . 47

5.2.5. Interfaz web de administración de la base de datos 49
5.3. Procesos 50
5.3.1. Procesamiento de un documento 52
5.3.2. Pregunta y Respuesta 52

6. Detalles de implementación **55**

7. Pruebas del sistema **65**
7.1. Depuración del código y detección de errores en el *back-end* 65
7.2. Depuración del código y detección de errores en el *front-end* 65
7.3. Pruebas unitarias 66
7.4. Pruebas de integración 68

8. Conclusiones y trabajo futuro **71**

Bibliografía **73**

Índice de figuras **75**

Índice de cuadros **77**

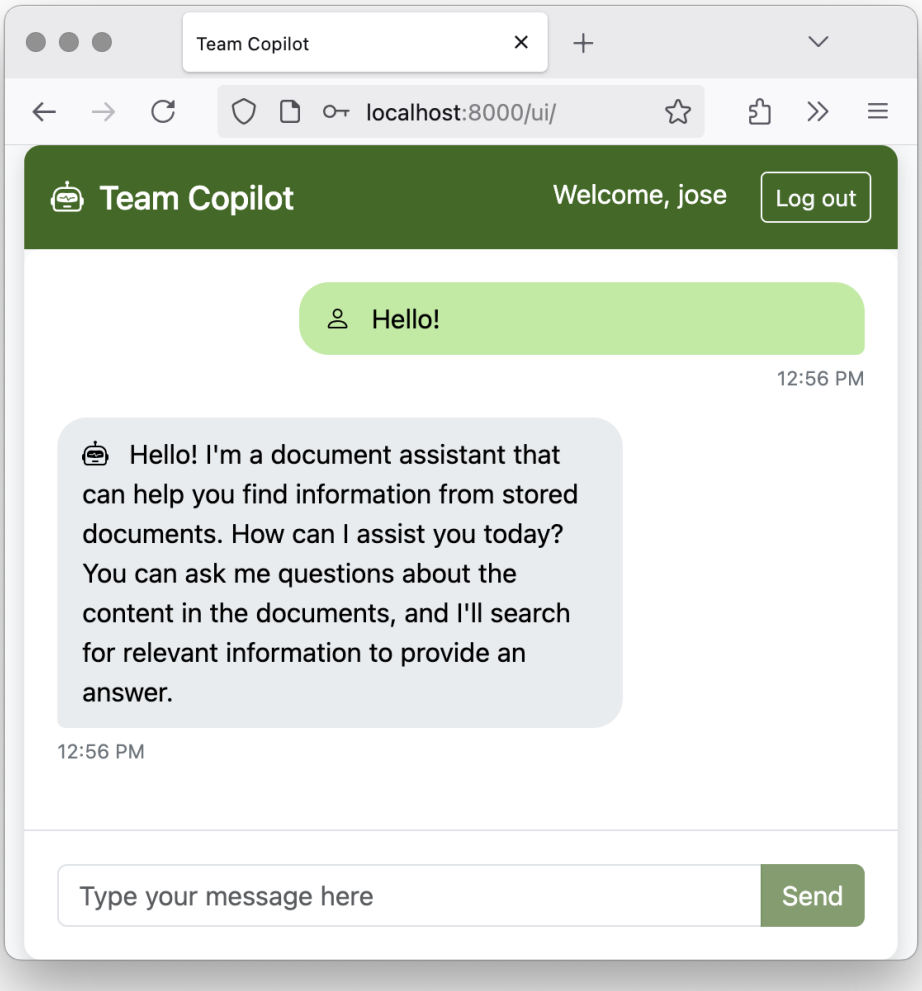
1. Introducción

Este trabajo ha consistido en el desarrollo de una **aplicación web** para gestionar **documentos PDF** y ofrecer un **chatbot** al que se le puedan hacer preguntas sobre el contenido de los documentos.

El objetivo de este trabajo es ofrecer una herramienta sencilla, desarrollada con herramientas libres (de código abierto) que facilite el día a día de un equipo de trabajo cualquiera en una empresa como **BASF Digital Solutions**, empresa para la cual se ha desarrollado el proyecto. En el desarrollo de este proyecto no se han utilizado datos reales de BASF Digital Solutions.

El código fuente del proyecto está disponible en GitHub:
`https://github.com/jajimenez/team-copilot`

Figura 1.1: *Interfaz de usuario de Team Copilot*



1.1. Motivación

Actualmente y, desde la aparición de **ChatGPT** [1], existen servicios de chatbot de inteligencia artificial basados en modelos LLM (*Large Language Model*) entrenados que permiten hacer preguntas sobre conocimiento general. Algunos de estos servicios son el propio ChatGPT, **Claude AI** [2] o **Gemini** [3]. Sin embargo, estos servicios por sí solos no son útiles para responder a preguntas dentro de un dominio del conocimiento específico, como puede ser el conocimiento interno de un equipo de trabajo en una empresa. Estos servicios permiten subir documentos específicos como contexto, pero deben ser los documentos que ya contengan la información requerida, por lo que estos servicios no ayudarían a responder a preguntas sobre documentos que son desconocidos por el usuario.

También, existen algunos proyectos de código abierto que ofrecen un *chatbot* para conocimiento específico o privado de una organización, pero a menudo son demasiado complejos o tienen una doble licencia que no permite su uso libre comercial.

Este proyecto pretende ser una herramienta lo más sencilla posible que aúna funcionalidades de diferentes servicios y proyectos diferentes, como son las APIs que permiten consultar a los modelos LLM y el framework **LangGraph** [4] que facilita desarrollar agentes de inteligencia artificial.

1.2. Objetivos generales

Los objetivos generales de este proyecto son los siguientes:

- **Objetivo 1:**
  Gestionar documentos PDF.
- **Objetivo 2:**
  Extraer el texto de los documentos PDF, tanto el texto directo como el texto contenido en imágenes.
- **Objetivo 3:**
  Almacenar el texto extraído de cada documento dividido en diferentes partes para luego encontrar la información relevante a una pregunta dada.
- **Objetivo 4:**
  Transformar el texto de cada parte de cada documento en vectores (*embeddings*), usando un modelo de *embedding* entrenado, para luego poder hacer búsquedas de partes de documentos relacionadas con una pregunta dada, la cual debe ser también transformada en un vector.
- **Objetivo 5:**
  Hacer preguntas sobre el contenido de los documentos a un modelo LLM entrenado, proporcionando al modelo el texto de la pregunta y, como contexto, el texto de las partes de documentos son más similares a (o tienen más relación con) la pregunta.

2. Contexto y Estado del Arte

2.1. Contexto

Actualmente, existe un gran número de servicios de inteligencia artificial para responder a preguntas sobre conocimiento general o para responder a preguntas proporcionando un documento específico como contexto, en diferentes formatos. También existen soluciones que permiten conectar varias fuentes de datos con servicios de inteligencia artificial para poder responder a preguntas sobre las fuentes de datos.

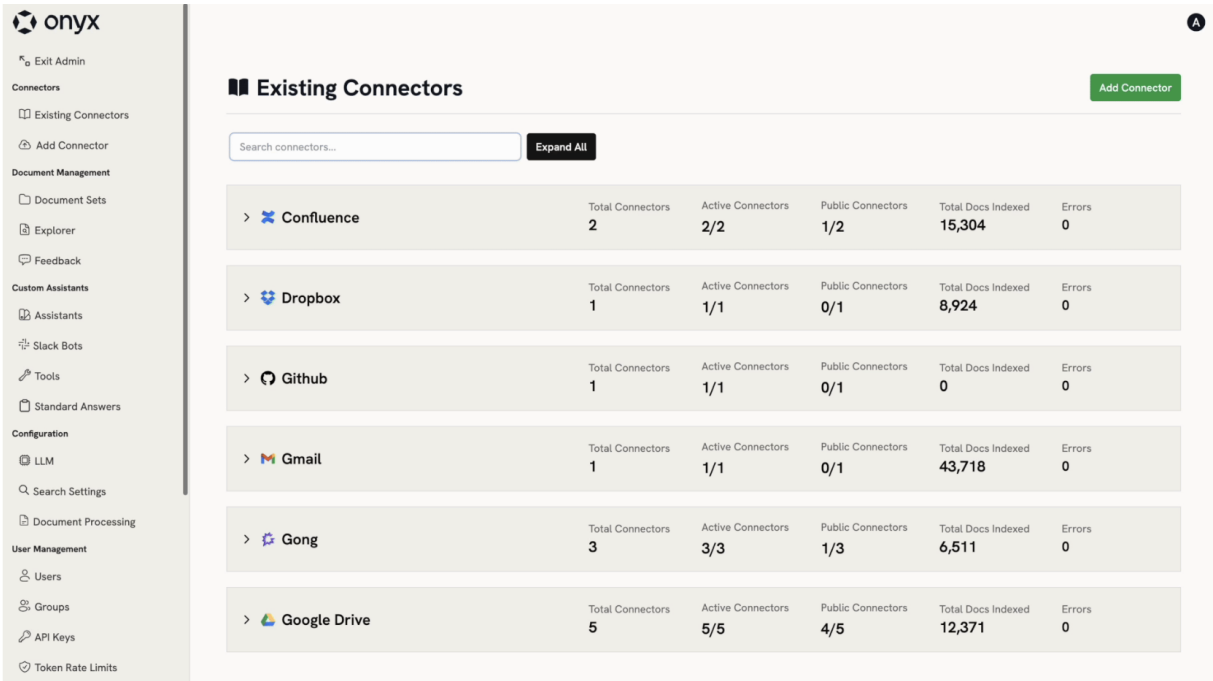
2.2. Estado del Arte

Existen varias soluciones que permiten conectar fuentes de datos con modelos LLM (Large Language Model) de servicios de inteligencia artificial. Veamos a continuación algunas de estas soluciones.

2.2.1. Onyx

Onyx (anteriormente llamado Danswer) [5] es una plataforma de software que conecta los datos de diferentes servicios como Google Drive, Slack, Confluence o Salesforce con diferentes modelos LLM como GPT o Claude. Se caracteriza por ser muy flexible y versátil. Por otro lado, es un proyecto con gran complejidad y una doble licencia.

Figura 2.1: *Captura de pantalla de Onyx*

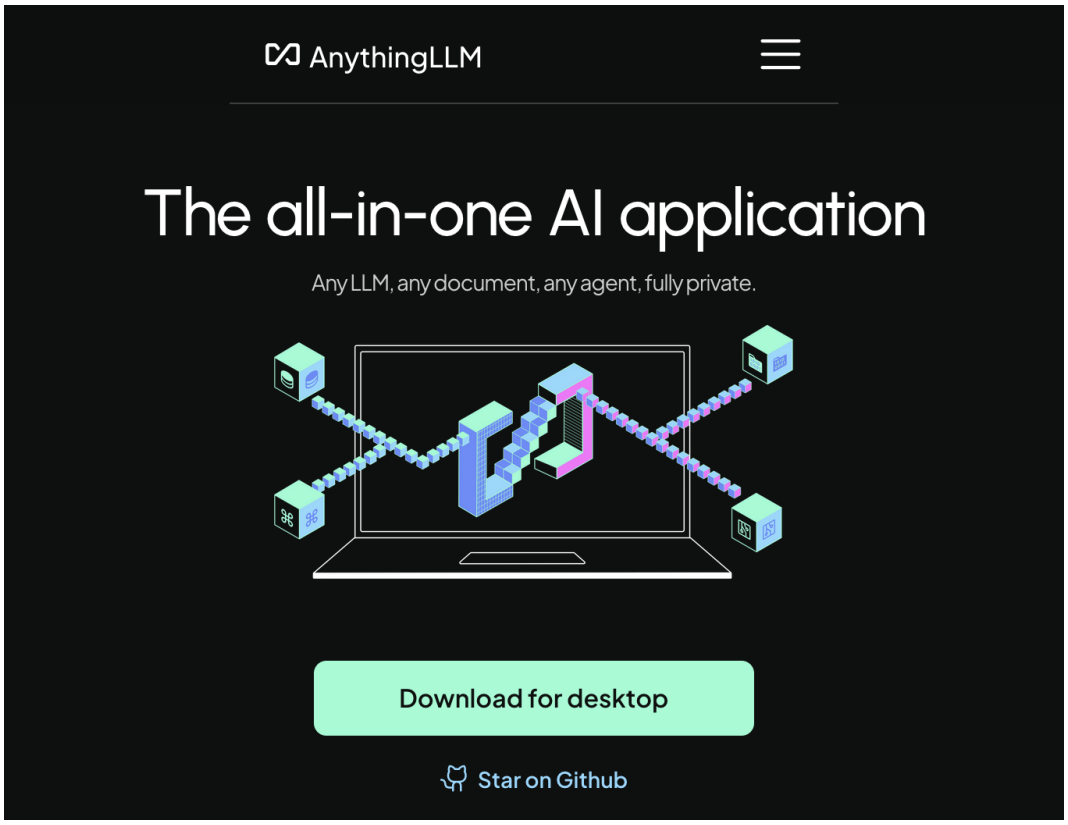


Fuente: GitHub

2.2.2. Anything LLM

Anything LLM [6] es una aplicación que permite convertir documentos o texto en contexto para enviar al modelo LLM de la elección del usuario, para poder hacer preguntas sobre dicho contexto. Se caracteriza por poder conectarse a una gran cantidad de modelos LLM y por ofrecer una aplicación cliente de escritorio. Por el contrario, este proyecto también es complejo y difícilmente modificable para adaptarlo a una organización o proyecto en concreto.

Figura 2.2: *Sitio web de Anything LLM*

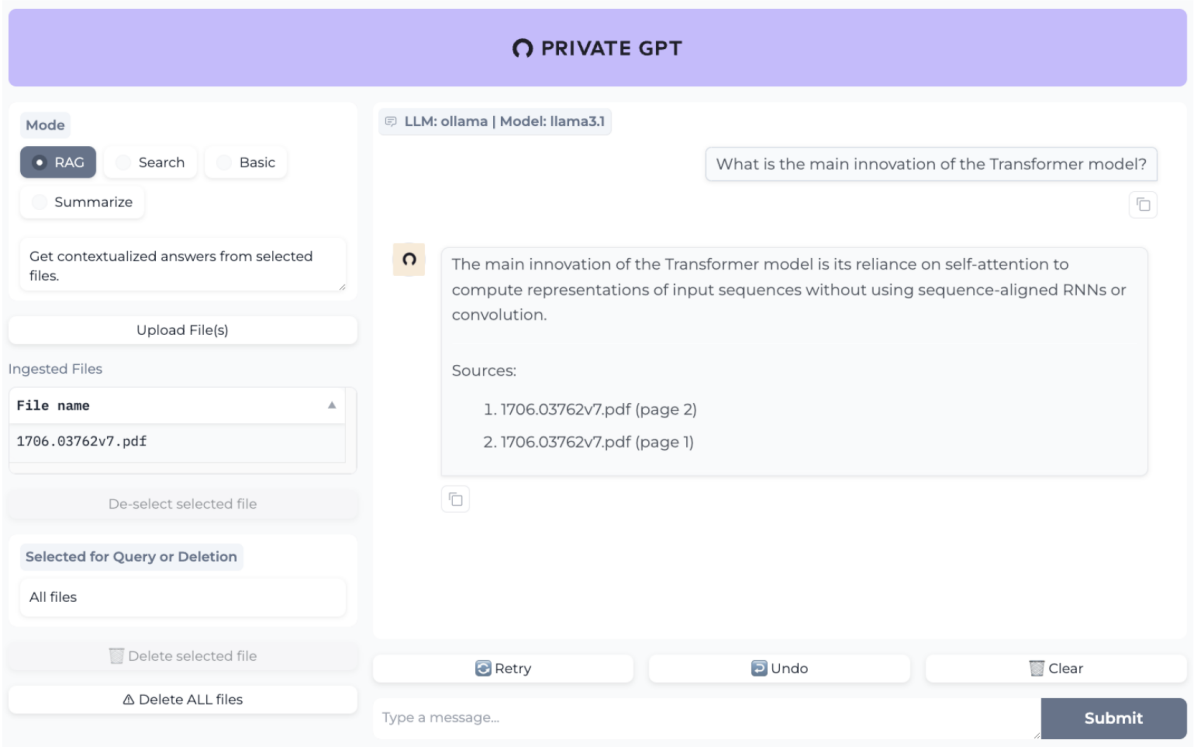


Fuente: Anything LLM

2.2.3. Private GPT

Private GPT [7] es un proyecto que ofrece un chatbot para hacer preguntas sobre documentos que se caracteriza por utilizar modelos LLM de código abierto en local, lo que garantiza una total confidencialidad de los datos. Como contrapartida, usar modelos en local requiere de un hardware con una gran capacidad de procesamiento para hacer inferencias.

Figura 2.3: *Captura de pantalla de Private GPT*



Fuente: GitHub

2.3. Comparativa y Conclusiones

A continuación, comparamos las soluciones anteriores con Team Copilot:

Cuadro 2.1: *Comparativa de soluciones de chatbots de documentos*

	Onyx	**Anything LLM**	**Private GPT**	**Team Copilot**
Gestión de documentos	**Sí**	**Sí**	**Sí**	**Sí**
Baja complejidad (fácilmente adaptable)	**No**	**No**	**Sí**	**Sí**
Puede ejecutarse con pocos recursos	**Sí**	**Sí**	**No**	**Sí**
100 % software libre	**No**	**Sí**	**Sí**	**Sí**

Podemos ver que Team Copilot ofrece un buen conjunto de características que hacen que sea una buena herramienta para implantar en una organización cuando se quiere ayudar a los miembros de la organización en su día a día ofreciéndoles una forma fácil de acceder a su propio conocimiento interno.

3. Planificación y Costes

3.1. Objetivos específicos

Los objetivos específicos del proyecto son los siguientes:

- Desarrollar un modelo de datos para gestionar la información de documentos.
- Desarrollar una aplicación web completa, con un *back-end* en forma de API y un *front-end* (interfaz de usuario) sencillo.
- Desarrollar una interfaz de usuario de permita identificarse en el sistema proporcionando un nombre de usuario y su contraseña correspondiente.
- Desarrollar una interfaz de usuario de permita hacer preguntas sobre el contenido de los documentos gestionados.
- Desarrollar una interfaz de usuario que responda a las preguntas hechas.
- Desarrollar una interfaz de usuario que sólo responda a preguntas sobre el contenido de los documentos gestionados y no sobre conocimiento general, respondiendo claramente que no sabe la respuesta cuando no la encuentre en los documentos.
- Responder a las preguntas en *streaming* (por partes).
- Usar un modelo de *embedding* pre-entrenado remoto para generar representaciones en forma de vector de las preguntas y del contenido de los documentos.
- Usar un modelo LLM *Large Language Model* pre-entrenado remoto para responder a las preguntas.
- Usar un modelo de *embedding* de Voyage AI [8].
- Usar un modelo LLM de Anthropic [9].
- Hacer la interfaz de usuario adaptable a dispositivos móviles.
- Crear archivos de configuración de Docker para facilitar el despliegue del proyecto.
- Utilizar sólo herramientas software libres y que funcionen en entornos libres (por ejemplo, Linux).
- Desarrollar la aplicación web utilizando el lenguaje de programación Python [10] y el *framework* FastAPI [11].
- Gestionar las conversaciones por medio de un agente desarrollado con el *framework* LangGraph [4].
- Desarrollar la interfaz de usuario utilizando el lenguaje de programación JavaScript y el *framework* Bootstrap [12].

3.2. Metodología de desarrollo

El método para desarrollar este proyecto ha seguido el modelo de Cascada ya que desde el principio se tuvieron bien definidos los requisitos. Las fases del proyecto han sido:

1. **Análisis**
   Esta primera fase ha consistido en estudiar el problema que se trata, estudiar su solución y los requisitos de la misma y organizar el trabajo a realizar.

2. **Diseño**
   En la fase de Diseño, se han definido el modelo de datos, la arquitectura y los requisitos específicos de software de la solución.

3. **Desarrollo**
   En esta fase se ha implementado la solución, escribiendo el código fuente de la misma y realizando cualquier otro trabajo necesario.

4. **Pruebas**
   La última fase ha consistido en comprobar que la solución cumple con los requisitos y funciona adecuadamente.

3.3. Planificación

Como se ha mencionado anteriormente, este proyecto ha consistido en las fases de Análisis, Diseño, Desarrollo y Pruebas. En la fase de Desarrollo se han construido los siguientes componentes:

1. ***Back-end* de la aplicación**
   Esto es la aplicación web en sí, escrita en forma de API con FastAPI, incluyendo el modelo de datos y la lógica de los diferentes *endpoints*.

2. **Interfaz de usuario (*front-end*)**
   Esta interfaz muestra un formulario para identificarse y una sección para poder hacer preguntas y mostrar las respuestas.

Las fechas y la dedicación estimada en horas de cada fase han sido las siguientes:

Cuadro 3.1: *Planificación del proyecto*

Fase	**Comienzo**	**Final**	**Dedicación estimada (h)**
Análisis	1 febrero 2025	28 febrero 2025	20
Diseño	1 marzo 2025	31 marzo 2025	35
Desarrollo	1 abril 2025	30 abril 2025	70
Pruebas	1 mayo 2024	31 mayo 2025	25

El total de la dedicación estimada es **150 horas**.

Los diagramas de Gannt y de recursos del proyecto son los siguientes.

Figura 3.1: *Diagrama de Gannt del proyecto*

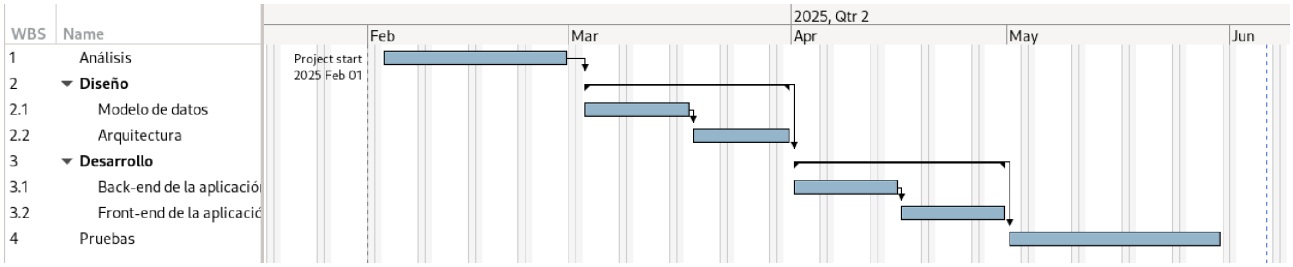
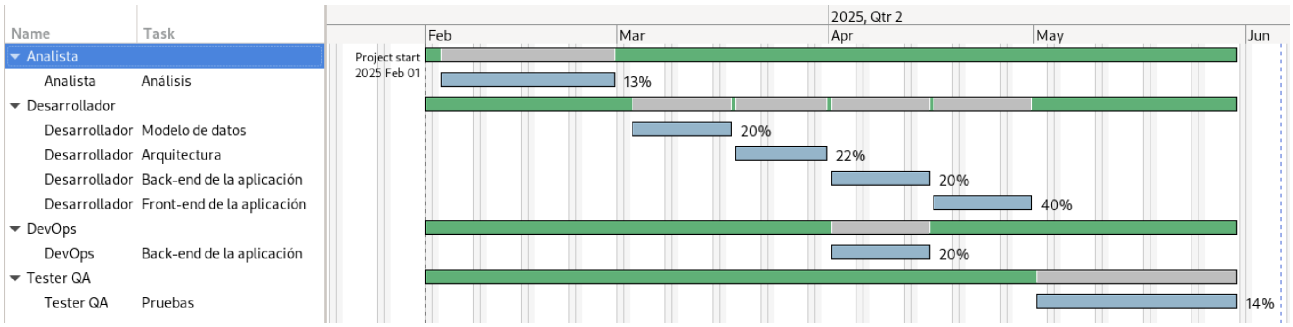


Figura 3.2: *Diagrama de recursos del proyecto*



3.4. Organización de personal del proyecto

A continuación, se definen los miembros del proyecto.

Cuadro 3.2: *Organización de personal del proyecto*

Nombre	**Rol**	**Funciones**
Juan Antonio Buero Viana	Cliente	Supervisión del proyecto
José Antonio Jiménez Carmona	Analista, desarrollador, DevOps, *tester*	Desarrollo del proyecto

3.5. Hardware y software del proyecto

Para el desarrollo y el despliegue del proyecto, se requieren los siguientes elementos de hardware y software.

Cuadro 3.3: *Hardware y software del proyecto*

Elemento	**Tipo**	**Función**
Ordenador portátil	Hardware	Desarrollo
Sistema operativo	Software	Desarrollo
Entorno de desarrollo	Software	Desarrollo
Cuenta en una plataforma de nube	Hardware	Despliegue
Instancia de máquinas virtual remota	Hardware	Despliegue
Base de datos relacional	Hardware	Despliegue

3.6. Tecnologías a emplear

Las tecnologías a emplear en el proyecto son las siguientes:

- **Visual Studio Code**
  Entorno de desarrollo integrado (IDE) como herramienta principal donde escribir el código del proyecto y ejecutar el proyecto y sus pruebas.
- **Docker**
  Herramienta de contenedores donde ejecutar y orquestar los diferentes componentes del sistema.
- **PostgreSQL**
  Sistema de gestión de bases de datos donde se almacenará toda la información del sistema.
- **PgVector**
  Extensión de PosgreSQL para permitir a PostgreSQL manajear datos en formato vector.
- **Documentos PDF**
  Documentos PDF con información pública sobre una asignatura de la Universidad de Sevilla. Estos documentos serán usados para probar el sistema.
- **API de Anthtopic** [9]
  API que permite hacer consultas a un modelo LLM.
- **API de Voyage AI** [8]
  API que permite convertir textos a vectores.
- **Python**
  Lenguaje de programación en el que se escribirá el proyecto.
- **FastAPI**
  Framework de Python que permite desarrollar fácilmente APIs.
- **LangGraph**
  Framework de Python que permite desarrollar fácilmente agentes de inteligenia artificial que se conectan a APIs de modelos LLM.
- **Bootstrap**
  Framework de JavaScript CSS que permite desarrollar interfaces web HTML de aspecto profesional y adaptables a distintos dispositivos.

3.7. Presupuesto del proyecto

A continuación, se define el presupuesto del proyecto teniendo en cuenta su planificación, su organización del personal y el hardware y software necesarios.

3.7.1. Coste del personal

Para calcular el coste del personal, usamos como referencia el sitio web Glassdor [13], que establece los siguientes salarios medios al año:

- Analista: 30.000 € [1]
- Desarrollador Python: 31.269 € [2]
- DevOps: 35,639 € [3]
- Tester QA: 22.937 [4]

Asumiendo que un año tenga 1920 horas de trabajo (12 meses por 20 días laborales por 8 horas), los salarios medios por hora serían los siguientes:

- Analista: 15,63 €
- Desarrollador Python: 16,29 €
- Ingeniero de DevOps: 18,56 €
- Tester Python: 11,95 €

Veamos ahora nuestro coste de personal estimado, teniendo en cuenta que de la fase de Diseño (35 horas) se encarga un desarrollador y de la fase de Desarrollo (70 horas), un desarrollador y un ingeniero de DevOps.

Cuadro 3.4: *Presupuesto, coste del personal*

Rol	**Cantidad**	**Horas estimadas**	**Coste por hora**	**Coste total**
Analista	1	20	15,63 €	312,6 €
Desarrollador	1	35 (Diseño) + 52,5 (Desarrollo) = 87,5	16,29 €	1.425,375 €
DevOps	1	17,5	18,56 €	324,8 €
Tester	1	25	11,95 €	298,75 €

El coste total estimado del personal es **2.361,525 €**.

[1] `https://www.glassdoor.es/Sueldos/analista-sueldo-SRCH_KO0,8.htm`

[2] `https://www.glassdoor.es/Salaries/spain-python-developer-salary-SRCH_IL.0,5_IN219_KO6,22.htm?countryRedirect=true`

[3] `https://www.glassdoor.es/Sueldos/devops-sueldo-SRCH_KO0,6.htm`

[4] `https://www.glassdoor.es/Sueldos/qa-tester-sueldo-SRCH_KO0,9.htm`

3.7.2. Coste del hardware y software

Teniendo en cuenta los elementos de infraestructura y software requeridos definidos anteriormente, a continuación enumeramos el coste por cada elemento. Para calcular el total del coste de cada elemento, consideramos la duración estimada del proyecto, que es de 4 meses.

Hemos tomado como referencia el coste de una instancia de máquina virtual en AWS en 2024 dado a través de la Calculadora de Precios de AWS [14], que es 22,48 dólares al mes o, al cambio actual, 19,62 euros al mes. El coste total asociado de la instancia para los 4 meses de duración del proyecto es **78,48 €**.

El ordenador portátil con el que se ha desarrollado el proyecto es un MacBook Pro, cuyo coste total es 1.457 € [15]. Siendo su tiempo de amortización 4 años (48 meses) y la duración del proyecto 4 meses, su coste total asociado es **121,42 €**.

No hay costes asociados al software porque se ha utilizado el sistema operativo incluido con el portátil, el entorno de desarrollo gratuito Visual Studio Code y el sistema gestor de bases de datos gratuito PostgreSQL.

Cuadro 3.5: *Presupuesto, coste de infraestructura y software*

Elemento	**Coste**	**Comentarios**
Ordenador portátil	121,42 €	
Instancia de máquina virtual remota	78,48 €	La instancia incluye un servidor PostgreSQL.
Total	**199,9 €**	

El coste total del hardware y el software es **199,9 €**.

A este coste habría que sumarle el coste de usar las APIs que Voyage AI y Anthropic, el cual varía mucho dependiendo de qué modelo en concreto se usa.

3.7.3. Resumen del presupuesto

Sumando el coste del personal y el coste del software y hardware, el presupuesto total para desarrollar el proyecto (excluyendo el coste de usar las APIs que Voyage AI y Anthropic) es **2.561,43 €**.

3.8. Desviación del proyecto

Veamos ahora los números reales de horas dedicadas al proyecto y la diferencia con los números estimados durante la planificación del proyecto.

La forma de registrar las horas de trabajo fue mediante la creación de una hoja de cálculo donde cada día de trabajo se apuntaba la hora de inicio del trabajo y la hora final del trabajo, junto a la fase del proyecto de la que se trataba.

Las fechas se mantuvieron sin cambios. Sin embargo, el número de horas de trabajo se desvió, debido principalmente a dudas técnicas sobre cómo afrontar el problema durante el desarrollo y un mayor tiempo del esperado dedicado a realizar pruebas del sistema.

Cuadro 3.6: *Planificación del proyecto*

Fase	**Dedicación estimada (h)**	**Dedicación real (h)**	**Desviación**
Análisis	20	23	15 %
Diseño	35	40	14,29 %
Desarrollo	70	98	40 %
Pruebas	25	35	40 %
Total	**150**	**196**	**30,67** %

4. Análisis de requisitos

En este capítulo veremos los participantes, objetivos, requisitos y casos de uso del proyecto.

4.1. Participantes

Las organizaciones y personas participantes del proyecto son las siguientes.

4.1.1. Organizaciones

Cuadro 4.1: *Análisis de requisitos, organización Departamento de Lenguajes y Sistemas Informáticos*

Nombre	BASF Digital Solutions S.L.
Dirección	Novus Building, Camino de la Fuente de la Mora 1, Planta 1, 28050 Madrid
Observaciones	-

4.1.2. Personas

Cuadro 4.2: *Análisis de requisitos, participante Juan Antonio Álvarez García*

Nombre	Juan Antonio Buero Viana
Organización	BASF Digital Solutions S.L.
Rol	Cliente
Observaciones	-

Cuadro 4.3: *Análisis de requisitos, participante José Antonio Jiménez Carmona*

Nombre	José Antonio Jiménez Carmona
Organización	-
Rol	Consultor-desarrollador
Observaciones	-

4.2. Objetivos

Definimos formalmente los objetivos del proyecto.

Cuadro 4.4: *Objetivo OBJ-0001*

ID	OBJ-0001
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Descripción	El sistema debe gestionar usuarios.
Subojetivos	-
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.5: *Objetivo OBJ-0002*

ID	OBJ-0002
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Descripción	El sistema debe gestionar documentos.
Subojetivos	-
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.6: *Objetivo OBJ-0003*

ID	OBJ-0003
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Descripción	El sistema debe consistir en una aplicación web completa, con un *back-end* en forma de API y un *front-end* (interfaz de usuario) sencillo.
Subojetivos	-
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.7: *Objetivo OBJ-0004*

ID	OBJ-0004
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Descripción	El sistema debe tener una interfaz de usuario de permita identificarse en el sistema proporcionando un nombre de usuario y su contraseña correspondiente.
Subojetivos	-
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.8: *Objetivo OBJ-0005*

ID	OBJ-0005
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Descripción	El sistema debe permitir al usuario hacer preguntas sobre el contenido de los documentos gestionados.
Subojetivos	-
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.9: *Objetivo OBJ-0006*

ID	OBJ-0006
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Descripción	El sistema debe responder a las preguntas hechas por el usuario.
Subojetivos	-
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.10: *Objetivo OBJ-0007*

ID	OBJ-0007
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Descripción	El sistema debe responder sólo a preguntas sobre el contenido de los documentos gestionados y no sobre conocimiento general, respondiendo claramente que no sabe la respuesta cuando no la encuentre en los documentos.
Subojetivos	-
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.11: *Objetivo OBJ-0008*

ID	OBJ-0008
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Descripción	El sistema debe responder a las preguntas en *streaming* (por partes).
Subojetivos	-
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.12: *Objetivo OBJ-0009*

ID	OBJ-0009
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Descripción	El sistema debe usar un modelo de *embedding* pre-entrenado remoto para generar representaciones en forma de vector de las preguntas y del contenido de los documentos.
Subojetivos	-
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.13: *Objetivo OBJ-0010*

ID	OBJ-0010
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Descripción	El sistema debe usar un modelo LLM *Large Language Model* pre-entrenado remoto para responder a las preguntas.
Subojetivos	-
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.14: *Objetivo OBJ-0011*

ID	OBJ-0011
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Descripción	El sistema debe verse adecuadamente en dispositivos móviles.
Subojetivos	-
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.15: *Objetivo OBJ-0012*

ID	OBJ-0012
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Descripción	El sistema debe ser fácilmente desplegable con contenedores.
Subojetivos	-
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.16: *Objetivo OBJ-0013*

ID	OBJ-0013
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Descripción	El sistema debe usar herramientas libres (de código abierto) y debe poder ejecutarse en entornos libres.
Subojetivos	-
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.17: *Objetivo OBJ-0014*

ID	OBJ-0014
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Descripción	El sistema debe permitir gestionar documentos sólo a usuarios que sean Personal Técnico.
Subojetivos	-
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.18: *Objetivo OBJ-0015*

ID	OBJ-0015
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Descripción	El sistema debe permitir gestionar usuarios sólo a usuarios que sean Administradores.
Subojetivos	-
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

4.3. Requisitos de información

Definimos formalmente los requisitos de información del proyecto.

Cuadro 4.19: *Requisito de información IRQ-0001*

ID	IRQ-0001
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Dependencias	-
Descripción	El sistema debe almacenar la información correspondiente a usuarios (aunque, inicialmente, sólo se necesita un usuario administrador), concretamente: ▪ ID ▪ Nombre de usuario ▪ Dirección de e-mail ▪ Nombre completo ▪ Está Habilitado ▪ Es Personal Técnico ▪ Es Administrador ▪ *Hash* de la contraseña
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.20: *Requisito de información IRQ-0002*

ID	IRQ-0002
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Dependencias	-
Descripción	El sistema debe almacenar la información correspondiente a documentos, concretamente: ▪ ID ▪ Nombre ▪ Estado
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.21: *Requisito de información IRQ-0003*

ID	IRQ-0003
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Dependencias	-
Descripción	El sistema debe almacenar la información correspondiente a partes/trozos de documentos, concretamente: ▪ ID ▪ ID del documento al que pertenece ▪ Índice de la parte (respecto al documento al que pertenece) ▪ Texto ▪ Vector (*embedding*) correspondiente al texto
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

4.4. Requisitos funcionales

Definimos formalmente los requisitos funcionales del proyecto.

Cuadro 4.22: *Requisito funcional FRQ-0001*

ID	FRQ-0001
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Dependencias	▪ OBJ-0001 ▪ OBJ-0004 ▪ IRQ-0001
Descripción	El sistema debe mostrar un botón que al pulsarlo abra un formulario para que el usuario se autentique con su nombre de usuario y contraseña.
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.23: *Requisito funcional FRQ-0002*

ID	FRQ-0002
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Dependencias	▪ OBJ-0001 ▪ IRQ-0001
Descripción	Una vez que el usuario está autenticado, el sistema debe mostrar un botón que al pulsarlo cierre la sesión.
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.24: *Requisito funcional FRQ-0003*

ID	FRQ-0003
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Dependencias	▪ OBJ-0005 ▪ OBJ-0006
Descripción	Una vez que el usuario está autenticado, el sistema debe mostrar un cuadro de texto donde el usuario puede escribir una pregunta y un botón para enviarla.
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.25: *Requisito funcional FRQ-0004*

ID	FRQ-0004
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Dependencias	▪ OBJ-0005 ▪ OBJ-0006
Descripción	Al pulsar el botón *Enviar*, la pregunta del usuario debe enviarse al sistema.
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.26: *Requisito funcional FRQ-0005*

ID	FRQ-0005
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Dependencias	▪ OBJ-0005 ▪ OBJ-0006 ▪ OBJ-0008
Descripción	Al recibir una pregunta, el sistema debe mostrar la respuesta por partes (en *streaming*) conforme se van generando las distintas partes.
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.27: *Requisito funcional FRQ-0006*

ID	FRQ-0006
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Dependencias	-
Descripción	El sistema debe mostrar la hora a la que se envió cada pregunta del usuario.
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.28: *Requisito funcional FRQ-0007*

ID	FRQ-0007
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Dependencias	-
Descripción	El sistema debe mostrar el texto "Pensando..."mientras el sistema esté procesando una pregunta del usuario.
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.29: *Requisito funcional FRQ-0008*

ID	FRQ-0008
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Dependencias	-
Descripción	El sistema debe mostrar la hora a la que se ha recibido cada respuesta del sistema, una vez que se haya recibido la respuesta completa.
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.30: *Requisito funcional FRQ-0009*

ID	FRQ-0009
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Dependencias	-
Descripción	Al procesar una pregunta, si ha habido algún error en el sistema, el sistema debe mostrar el mensaje de error en su respuesta, con un color de fondo rojo.
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

4.5. Requisitos no funcionales

Definimos formalmente los requisitos no funcionales del proyecto.

Cuadro 4.31: *Requisito no funcional NFR-0001*

ID	NFR-0001
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Dependencias	▪ OBJ-0011
Descripción	La interfaz web debe tener sólo una página web o sección, donde esté la parte para que el usuario se autentique y la parte para que el usuario escriba y envíe una pregunta.
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.32: *Requisito no funcional NFR-0002*

ID	NFR-0002
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Dependencias	
Dependencias	▪ OBJ-0005 ▪ OBJ-0006
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.33: *Requisito no funcional NFR-0003*

ID	NFR-0003
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Dependencias	
Descripción	Las contraseñas de los usuarios deben estar almacenadas de forma segura.
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.34: *Requisito no funcional NFR-0004*

ID	NFR-0004
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Dependencias	▪ OBJ-0002 ▪ IRQ-0002
Descripción	Cuando se sube un documento PDF al sistema, el sistema debe programar su procesamiento para que ocurra lo antes posible, estableciendo inicialmente el estado del documento como "Pendiente".
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.35: *Requisito no funcional NFR-0005*

ID	NFR-0005
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Dependencias	▪ OBJ-0002 ▪ IRQ-0002
Descripción	Cuando comienza el procesamiento de un documento, el sistema establece el estado del documento como "Procesando".
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.36: *Requisito no funcional NFR-0006*

ID	NFR-0006
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Dependencias	▪ OBJ-0002 ▪ IRQ-0002
Descripción	El texto de los documentos PDF debe extraerse del texto en sí de los documentos y de las imágenes de los documentos mediante OCR.
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.37: *Requisito no funcional NFR-0007*

ID	NFR-0007
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Dependencias	▪ OBJ-0002 ▪ IRQ-0002
Descripción	Cuando termina el procesamiento de un documento, el sistema establece el estado del documento como Çompletado".
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.38: *Requisito no funcional NFR-0008*

ID	NFR-0008
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Dependencias	▪ OBJ-0002 ▪ IRQ-0002
Descripción	Cuando termina el procesamiento de un documento, el sistema debe borrar el archivo original.
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.39: *Requisito no funcional NFR-0009*

ID	NFR-0009
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Dependencias	▪ OBJ-0002 ▪ IRQ-0002
Descripción	En caso de que haya ocurrido un error durante el procesamiento de un documento, el sistema establece el estado del documento como "Fallado", borrando el archivo original.
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

4.6. Casos de uso

Definimos formalmente los actores y los casos de uso del proyecto.

4.6.1. Actores

Los actores del sistema son:

- **ACT-0001**: Usuario cliente

- **ACT-0002**: Usuario personal técnico
- **ACT-0003**: Usuario administrador

4.6.2. Casos de uso

Los casos de uso del sistema son:

Cuadro 4.40: *Caso de uso UC-0001*

ID	UC-0001
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Dependencias	▪ OBJ-0001 ▪ OBJ-0004 ▪ IRQ-0001
Descripción	Autenticarse en el sistema.
Pre-condición	-
Secuencia	1. El usuario cliente abre la interfaz de usuario de la aplicación. 2. El usuario cliente hace clic en el botón *Log in*. 3. El sistema muestra un formulario. 4. El usuario introduce su nombre de usuario, su contraseña y hace clic en el botón *Accept*.
Post-condición	-
Excepciones	-
Frecuencia esperada	-
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.41: *Caso de uso UC-0002*

ID	UC-0002
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Dependencias	▪ OBJ-0001 ▪ OBJ-0004 ▪ IRQ-0001
Descripción	Cerrar sesión en el sistema.
Pre-condición	El usuario debe haber abierto la interfaz web de usuario y haberse autenticado.
Secuencia	1. El usuario hace clic en el botón *Log out*.
Post-condición	-
Excepciones	-
Frecuencia esperada	-
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.42: *Caso de uso UC-0003*

ID	UC-0003
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Dependencias	▪ OBJ-0005 ▪ OBJ-0006
Descripción	Enviar una pregunta al sistema.
Pre-condición	El usuario debe haber abierto la interfaz web de usuario y haberse autenticado.
Secuencia	1. El usuario escribe la pregunta. 2. El usuario hace clic en el botón *Send*. 3. El sistema devuelve la respuesta al usuario por partes (en *streaming*).
Post-condición	-
Excepciones	-
Frecuencia esperada	-
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.43: *Caso de uso UC-0004*

ID	UC-0004
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Dependencias	▪ OBJ-0001 ▪ IRQ-0001
Descripción	Crear un usuario.
Pre-condición	El usuario debe ser Administrador.
Secuencia	1. El usuario hace una petición HTTP a la API del sistema, proporcionando los datos del usuario.
Post-condición	-
Excepciones	-
Frecuencia esperada	-
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.44: *Caso de uso UC-0005*

ID	UC-0005
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Dependencias	▪ OBJ-0001 ▪ IRQ-0001
Descripción	Modificar un usuario.
Pre-condición	El usuario debe ser Administrador.
Secuencia	1. El usuario hace una petición HTTP a la API del sistema, proporcionando el ID del usuario y sus nuevos datos.
Post-condición	-
Excepciones	-
Frecuencia esperada	-
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.45: *Caso de uso UC-0006*

ID	UC-0006
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Dependencias	▪ OBJ-0001 ▪ IRQ-0001
Descripción	Borrar un usuario.
Pre-condición	El usuario debe ser Administrador.
Secuencia	1. El usuario hace una petición HTTP a la API del sistema, proporcionando el ID del usuario.
Post-condición	-
Excepciones	-
Frecuencia esperada	-
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.46: *Caso de uso UC-0007*

ID	UC-0007
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Dependencias	▪ OBJ-0002 ▪ IRQ-0002
Descripción	Crear un documento.
Pre-condición	El usuario debe ser Personal Técnico.
Secuencia	1. El usuario hace una petición HTTP a la API del sistema, proporcionando un nombre del documento y el archivo del documento PDF.
Post-condición	-
Excepciones	-
Frecuencia esperada	-
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.47: *Caso de uso UC-0008*

ID	UC-0008
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Dependencias	▪ OBJ-0002 ▪ IRQ-0002
Descripción	Modificar un documento.
Pre-condición	El usuario debe ser Personal Técnico.
Secuencia	1. El usuario hace una petición HTTP a la API del sistema, proporcionando el nuevo nombre del documento y el nuevo archivo del documento PDF.
Post-condición	-
Excepciones	-
Frecuencia esperada	-
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

Cuadro 4.48: *Caso de uso UC-0009*

ID	UC-0009
Versión	1
Fuente	Juan Antonio Buero Viana
Autor	José Antonio Jiménez Carmona
Dependencias	▪ OBJ-0002 ▪ IRQ-0002
Descripción	Borrar un documento.
Pre-condición	El usuario debe ser Personal Técnico.
Secuencia	1. El usuario hace una petición HTTP a la API del sistema, proporcionando el ID del documento a borrar.
Post-condición	-
Excepciones	-
Frecuencia esperada	-
Importancia	Crítica
Urgencia	Inmediata
Estado	Validado
Estabilidad	Alta
Comentarios	-

5. Diseño

En este capítulo describimos el diseño del sistema.

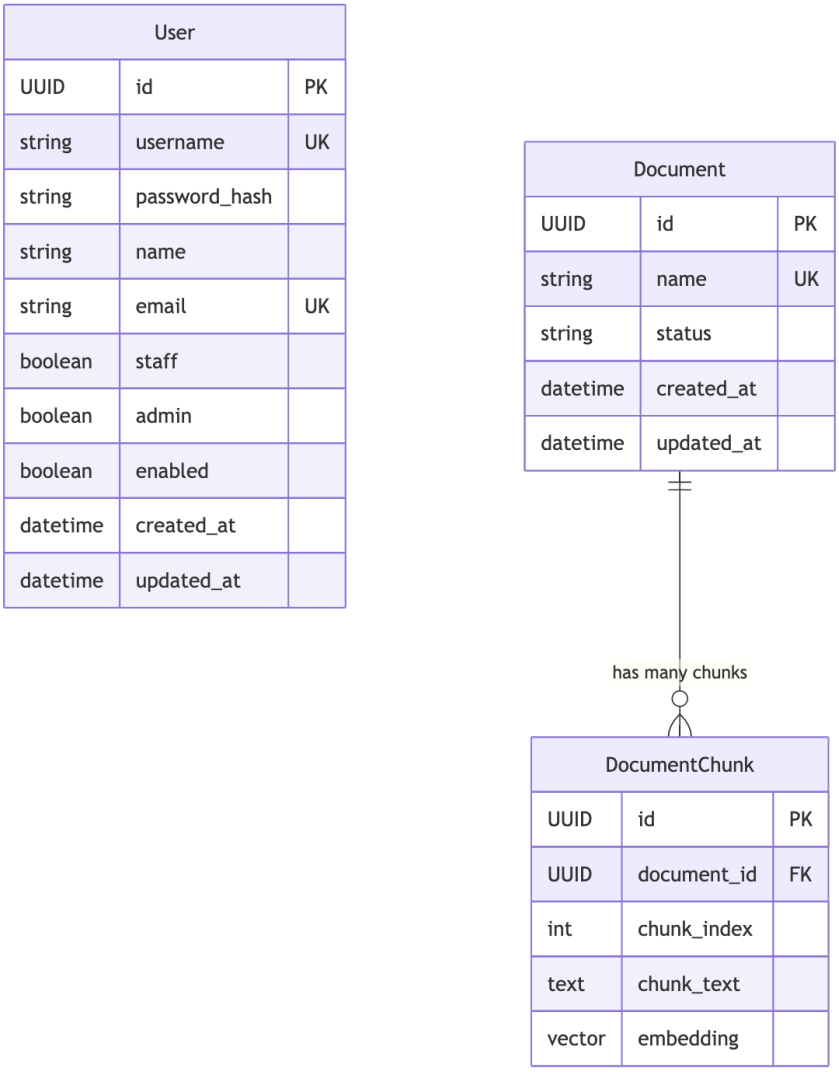
5.1. Modelo de datos

El modelo de datos del sistema lo componen las siguientes entidades:

- ***User*:** Usuarios del sistema.
- ***Document*:** Documentos gestionados.
- ***DocumentChunk*:** Partes/trozos de los documentos.

A continuación, vemos el diagrama UML del modelo de datos:

Figura 5.1: *Diagrama UML del modelo de datos*



5.2. Arquitectura

Los componentes que componen la arquitectura del sistema son:

- **API**
- **Base de datos PostgreSQL**
- **Interfaz web de usuario**
- **Interfaz web de documentación**
- **Interfaz web de administración de la base de datos**

Todos estos componentes se pueden ejecutar en contenedores de Docker.

5.2.1. API

La API es el componente principal, y está escrita usando el *framework* FastAPI.

La aplicación tiene un agente basado en LangGraph que gestiona los chats y utiliza un modelo de embeddings remoto de Voyage AI [8] y un modelo LLM remoto de Anthropic [9]. Por lo tanto, se requieren una clave API de Voyage AI y una clave API de Anthropic para ejecutar el agente.

La aplicación utiliza las librerías de Python **PyMuPDF** y **PyTesseract** para extraer el texto de los documentos PDF. PyMuPDF se utiliza para extraer texto plano e imágenes, y PyTesseract se utiliza para extraer texto de las imágenes previamente extraídas a través de OCR (Reconocimiento Óptico de Caracteres).

5.2.2. Base de datos PostgreSQL

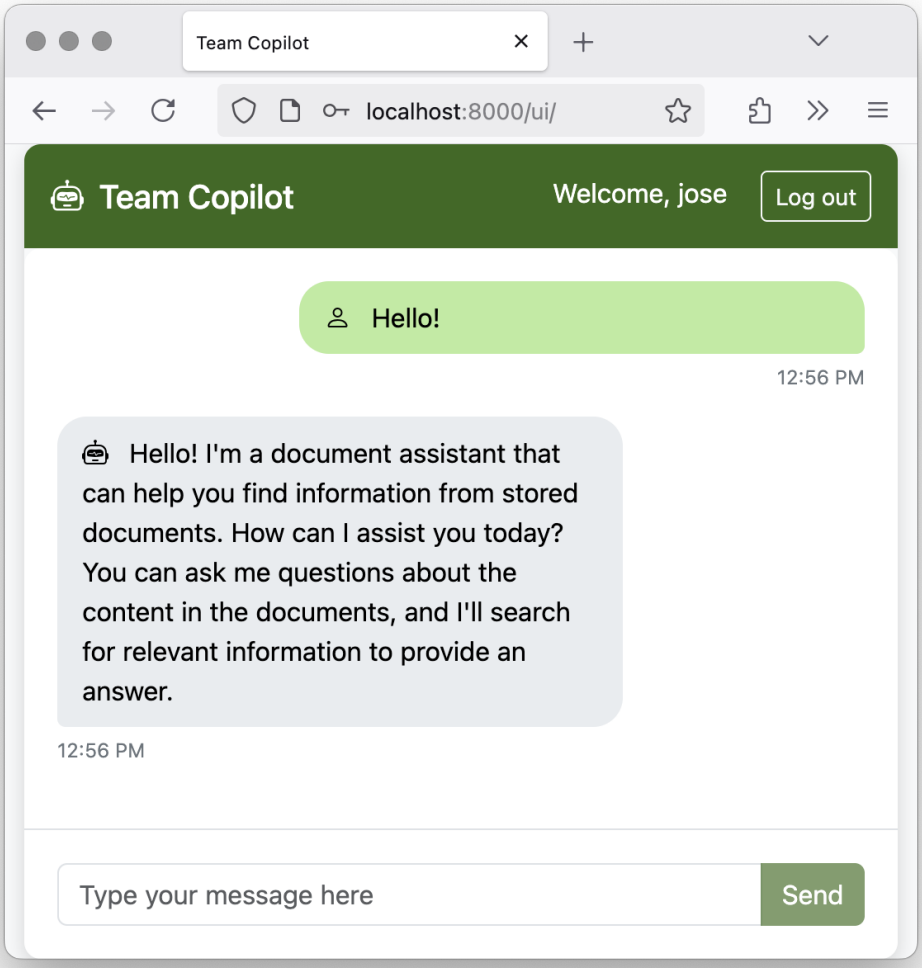
La base de datos almacena toda la información del sistema, incluidos los usuarios y los documentos.

El texto extraído de cada documento se almacena en una base de datos PostgreSQL configurada como una base de datos vectorial con la extensión de PostgreSQL **PgVector**.

5.2.3. Interfaz web de usuario

La interfaz web de usuario es una página HTML única que sirve la propia aplicación de FastAPI para ofrecer una forma de interactuar con el sistema. Esta interfaz utiliza el *framework* **Bootstrap** [12] para sus estilos CSS.

Figura 5.2: *Interfaz web de usuario*



5.2.4. Interfaz web de documentación

Esta interfaz web la ofrece FastAPI para disponer de una forma sencilla de probar los diferentes *endpoints* de la API.

Figura 5.3: *Interfaz web de documentación (parte 1)*

Figura 5.4: *Interfaz web de documentación (parte 2)*

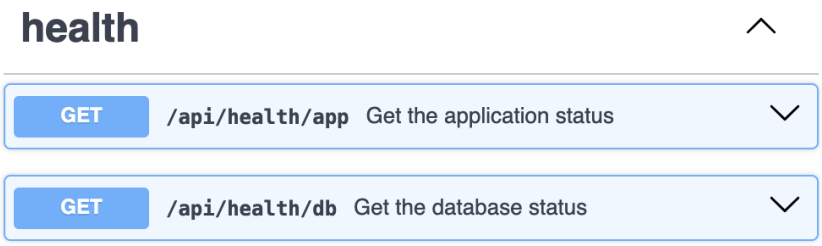


Figura 5.5: *Interfaz web de documentación (parte 3)*

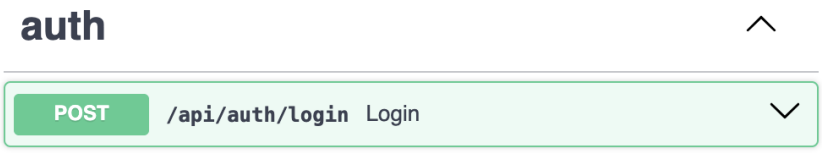


Figura 5.6: *Interfaz web de documentación (parte 4)*

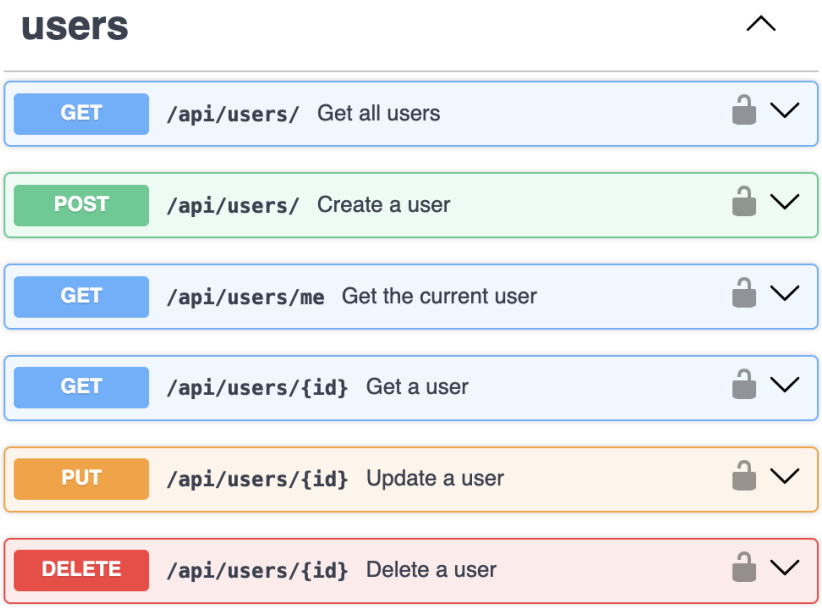


Figura 5.7: *Interfaz web de documentación (parte 5)*

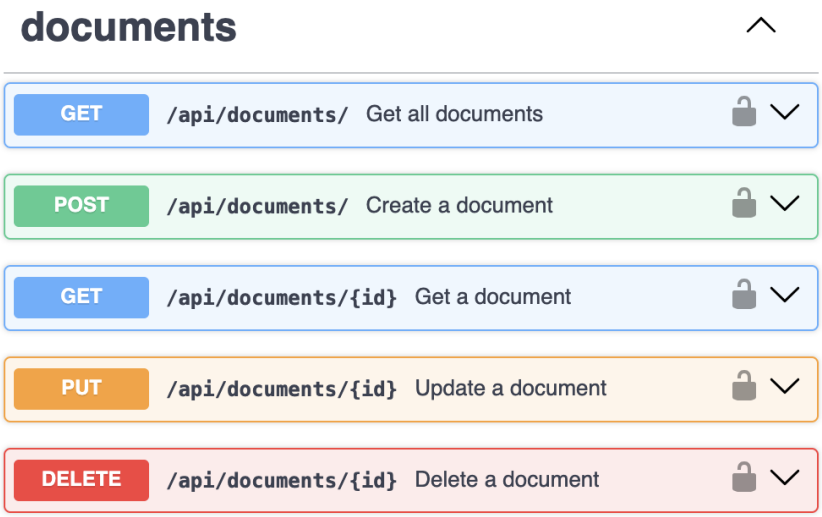


Figura 5.8: *Interfaz web de documentación (parte 6)*

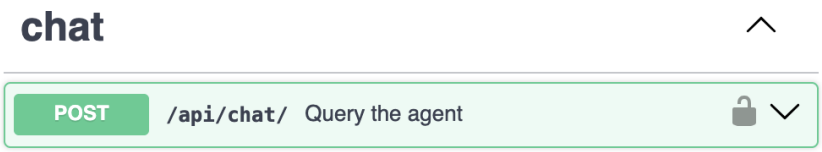
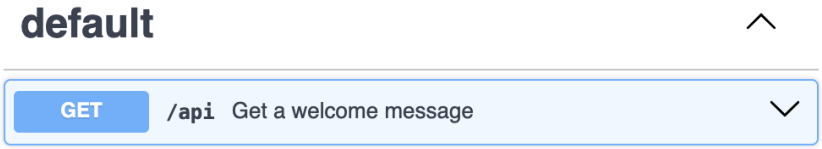


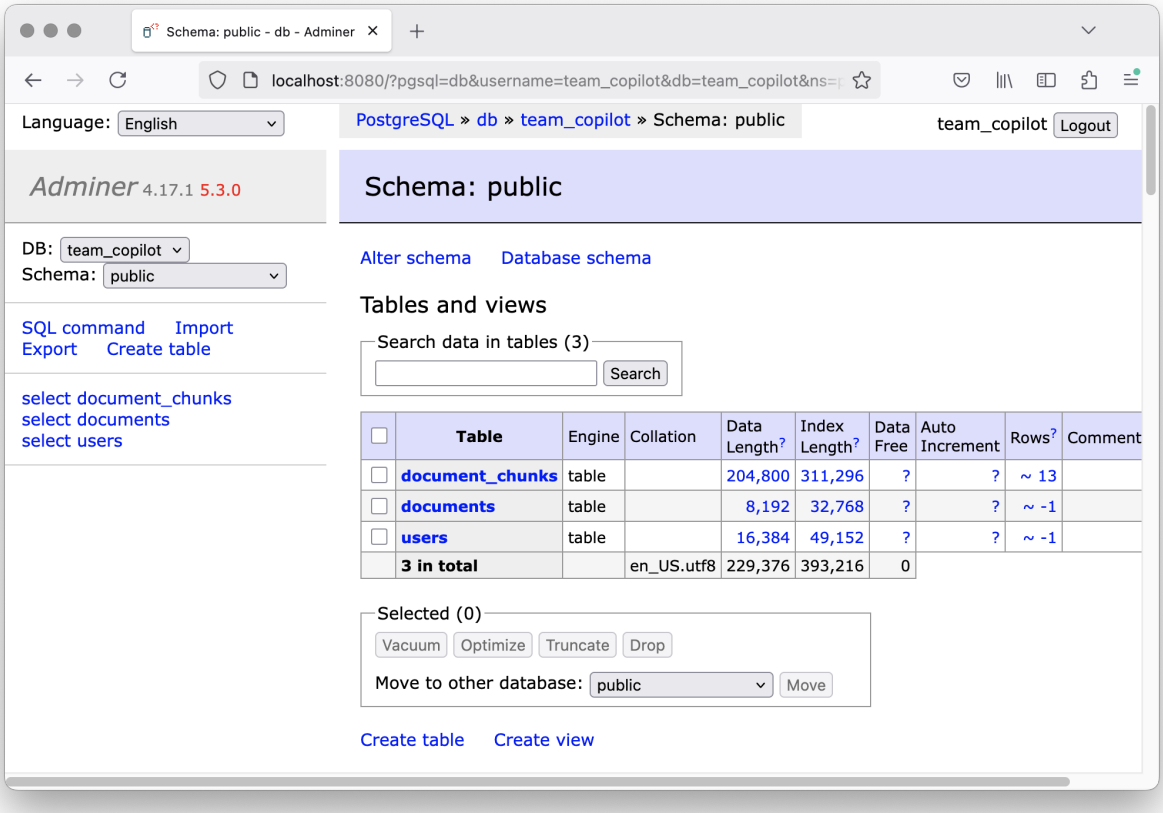
Figura 5.9: *Interfaz web de documentación (parte 7)*



5.2.5. Interfaz web de administración de la base de datos

Esta interfaz no ha sido desarrollada para este proyecto, pero la proporciona la aplicación externa ***Adminer***, la cual está conectada a la base de datos y permite administrarla.

Figura 5.10: *Interfaz web de administración de la base de datos*



5.3. Procesos

El sistema consta principalmente de dos procesos: el **procesamiento de un documento** y la **pregunta-respuesta**.

Figura 5.11: *Diagrama de flujo (parte 1)*

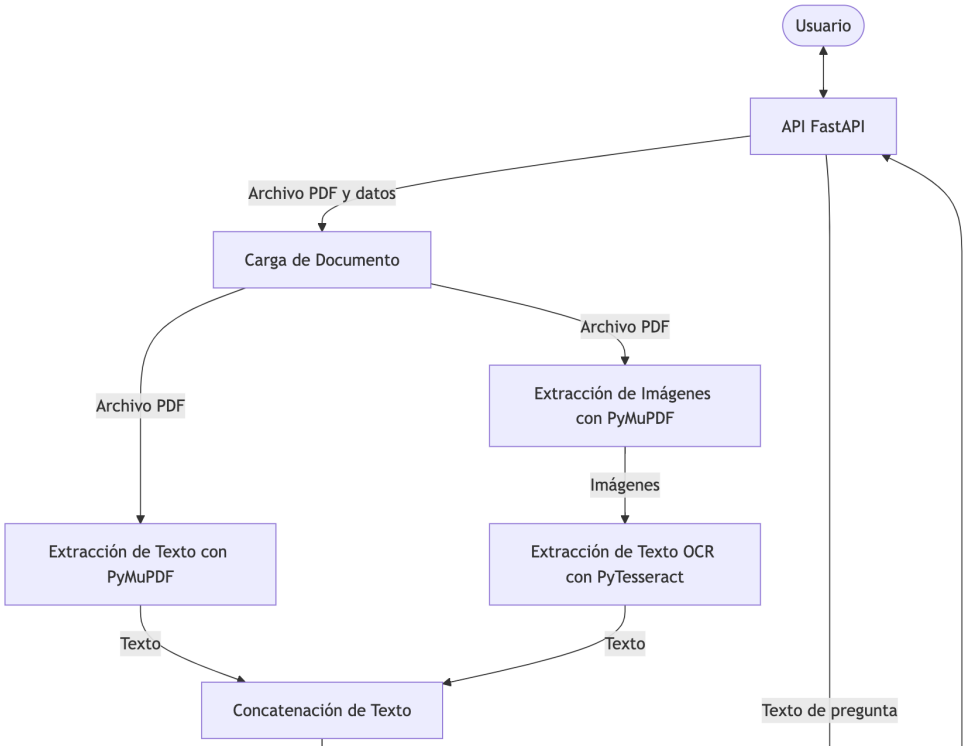


Figura 5.12: *Diagrama de flujo (parte 2)*

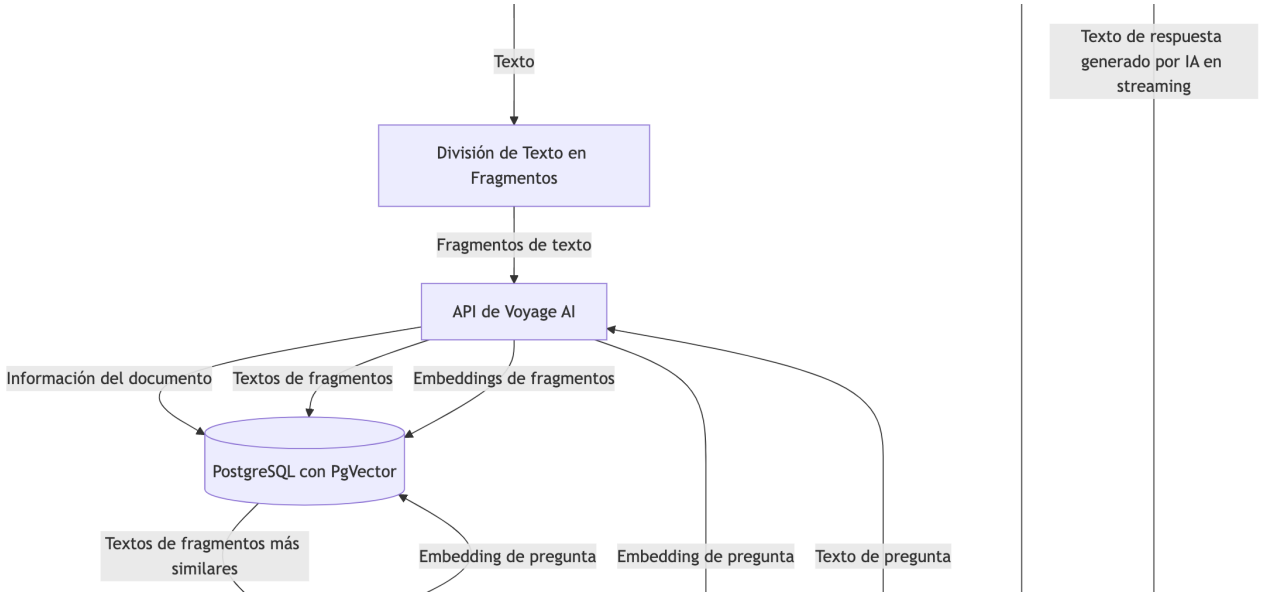
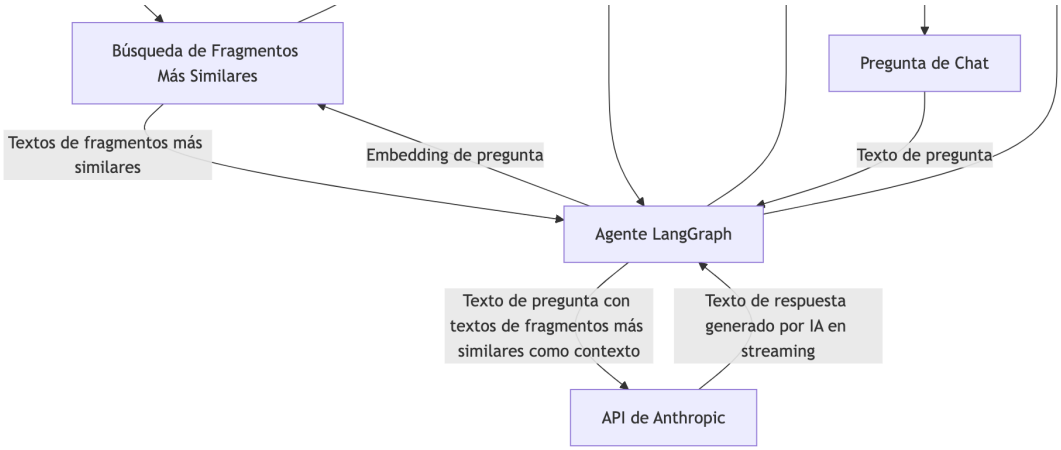


Figura 5.13: *Diagrama de flujo (parte 3)*



5.3.1. Procesamiento de un documento

Cada vez que se sube un documento PDF a la aplicación, se realizan los siguientes pasos:

1. Se inserta un nuevo registro de documento en la base de datos, con estado "pendiente".
2. El documento PDF se sube a una ruta temporal en el sistema de archivos.
3. El estado del registro del documento en la base de datos se actualiza a "procesando".
4. Se extraen el texto y las imágenes del documento con PyMuPDF.
5. Para cada imagen extraída, si la imagen contiene texto, se extrae el texto con PyTesseract.
6. El texto extraído completo se divide en fragmentos.
7. Para cada fragmento de texto, se genera el embedding (vector de números) del fragmento con el modelo de embedding remoto de Voyage AI.
8. Para cada fragmento de texto, se inserta un nuevo registro de fragmento de documento en la base de datos, incluyendo el texto del fragmento y el embedding del fragmento.
9. El estado del registro del documento en la base de datos se actualiza a çompletado". Si ocurre algún error durante los pasos anteriores, el estado se actualiza a "fallido".
10. El documento PDF subido se elimina.

5.3.2. Pregunta y Respuesta

Cada vez que se hace una pregunta a la aplicación, se realizan los siguientes pasos:

1. El agente convierte el texto de la pregunta en un embedding (vector de números) con el modelo de embedding remoto de Voyage AI.
2. El agente busca los embeddings de los fragmentos de documentos almacenados en la base de datos que son los embeddings más similares al embedding de la pregunta. La búsqueda se realiza calculando la distancia coseno entre dos vectores, donde los vectores son cualquiera de los embeddings de fragmentos de documentos y el embedding de la pregunta. Cuanto menor sea la distancia coseno entre dos vectores, más similares serán los vectores.
3. El agente recupera el texto de los fragmentos de documentos más similares encontrados y lo concatena en un solo texto.

4. El agente envía el texto de la pregunta, y el texto de los fragmentos de documentos más similares encontrados como contexto, al modelo LLM remoto de Anthropic.
5. El agente devuelve la respuesta del modelo LLM en streaming, que también es devuelta por la aplicación.

6. Detalles de implementación

A continuación, procedemos a ver algunos detalles sobre cómo se han implementado los diferentes componentes de la arquitectura del sistema. Todos los componentes están orquestados mediante un archivo de Docker Compose.

Figura 6.1: *Archivo Docker Compose*

docker-compose.yml

```yaml
services:
  app:
    build: .
    ports:
      - "8000:${TEAM_COPILOT_APP_PORT:-8000}"
    depends_on:
      - db
    env_file:
      - .env-app
    restart: unless-stopped

  db:
    build:
      dockerfile: Dockerfile-db
    ports:
      - "5432:5432"
    env_file:
      - .env-db
    volumes:
      - postgres_data:/var/lib/postgresql/data
    restart: unless-stopped
    command: ["postgres", "-c", "shared_preload_libraries=pg_stat_statements"]

  db-ui:
    image: adminer:4.17.1
    ports:
      - "8080:8080"
    depends_on:
      - db
    restart: unless-stopped

volumes:
  postgres_data:
```

El servicio ***app*** configura un contenedor para ejecutar la API del sistema, mientras que el servicio ***db*** configura un contenedor donde se ejecutará el servidor de bases de datos PostgreSQL. Adicionalmente, el servicio ***db-ui*** ofrece una aplicación web externa llamada **Adminer** que nos permite gestionar la base de datos.

Cabe destacar el archivo ***Dockerfile-db*** que define la imagen Docker en la que se basará el servicio *db*. En esta imagen, se configura el servidor PostgreSQL para que instale la extensión PostgreSQL y pueda así ser usado como un servidor de bases de datos vectoriales.

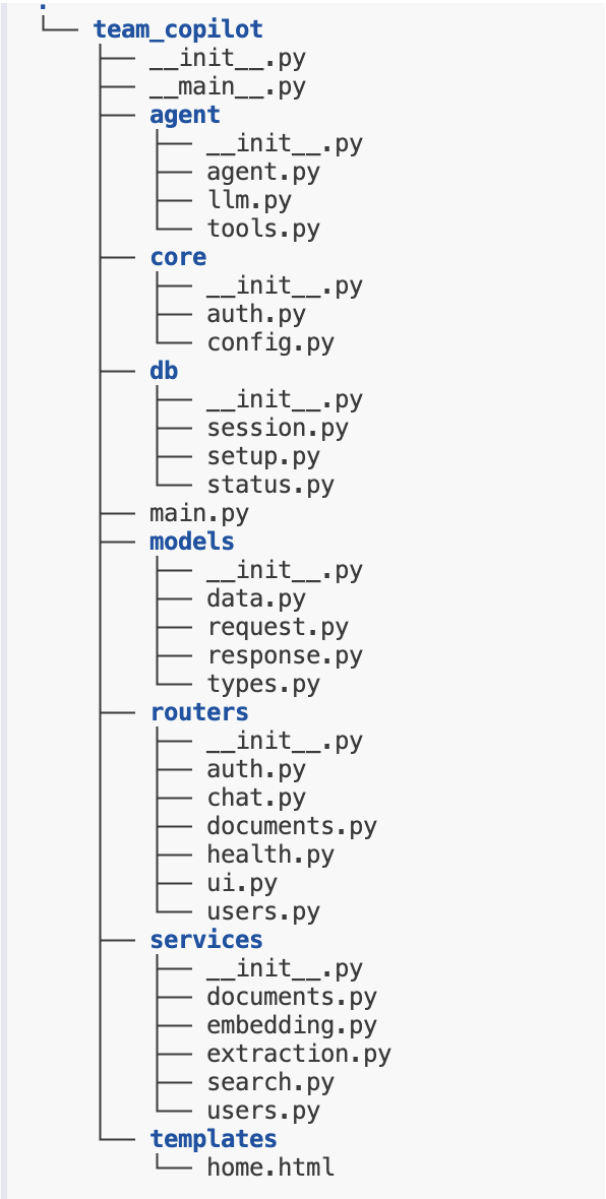
Figura 6.2: *Archivo Dockerfile-db*

Dockerfile-db

```
# Base image
FROM postgres:17.4

ENV PGVECTOR_DIR=/pgvector

# Update the APT package list.
# Install APT packages for building the Pgvector extension.
# Clone the Pgvector repository.
RUN apt-get update && \
    apt-get install -y build-essential git postgresql-server-dev-17 && \
    git clone https://github.com/pgvector/pgvector.git $PGVECTOR_DIR

# Build and install the Pgvector extension.
RUN cd $PGVECTOR_DIR && \
    make && \
    make install

# Clean up the APT cache.
RUN apt-get remove -y build-essential git postgresql-server-dev-17 && \
    apt-get autoremove -y && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/* && \
    rm -rf $PGVECTOR_DIR

```

El código fuente del proyecto está organizado como un paquete de Python dentro del directorio *src* con la siguiente estructura:

Figura 6.3: *Estructura del paquete Python*



Los directorios principales son:

- ***agent***:
  Contiene la lógica del agente de LangGraph que gestiona cada conversación. Éste agente es el encargado de recibir cada pregunta del usuario, de buscar los documentos relacionados a la pregunta y de enviar la pregunta junto a los documentos relacionados como contexto al modelo LLM para obtener una respuesta.
- ***core***:
  Contiene la lógica de la autenticación de usuario y la lógica de la configuración del sistema.
- ***db***:
  Contiene la lógica de la conexión a la base de datos.
- ***models***:

Contiene la definición de los modelos de datos.

- ***routers***:
  Contiene la lógica de los controladores de los *endpoints* de la API.
- ***services***:
  Contiene la lógica de negocio del sistema.

Figura 6.4: *Ejemplo de controlador (get_document) (parte 1)*

documents.py 8 ×

src > team_copilot > routers > documents.py > ...

```python
@router.get(
    "/{id}",
    operation_id="get_document",
    summary=GET_DOC_SUM,
    description=GET_DOC_DESC,
    status_code=status.HTTP_200_OK,
    responses={
        status.HTTP_200_OK: {
            "description": DOC_DAT,
            "model": DocumentResponse,
        },
        status.HTTP_404_NOT_FOUND: {
            "description": DOC_NF_1,
            "model": ErrorResponse,
        },
    },
)
async def get_document(
    id: Annotated[UUID, Path(description=DOC_ID)],
) -> DocumentResponse:
    """Get a document.

    Args:
        id (UUID): Document ID.

    Raises:
        HTTPException: If the document is not found.

    Returns:
        DocumentResponse: Message and document.
    """
```

Figura 6.5: *Ejemplo de controlador (get_document) (parte 2)*

```python
    # Get the document from the database
    doc = get_doc(id=id)

    # Check that the document exists
    if not doc:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail=DOC_NF_2.format(id),
        )

    # Return message and document
    message = DOC_RET.format(doc.id, doc.name)
    return DocumentResponse.create(message=message, document=doc)
```

Aparte de los modelos de datos que utiliza el sistema (*User*, *Document* y *DocumentChunk*), el sistema define el tipo de datos personalizado ***VectorType*** para PostgreSQL. Este tipo de datos es necesario (junto a la extensión ***PgVector*** ya que PostgreSQL no soporta vectores por defecto.

Figura 6.6: *Tipo de datos VectorType*

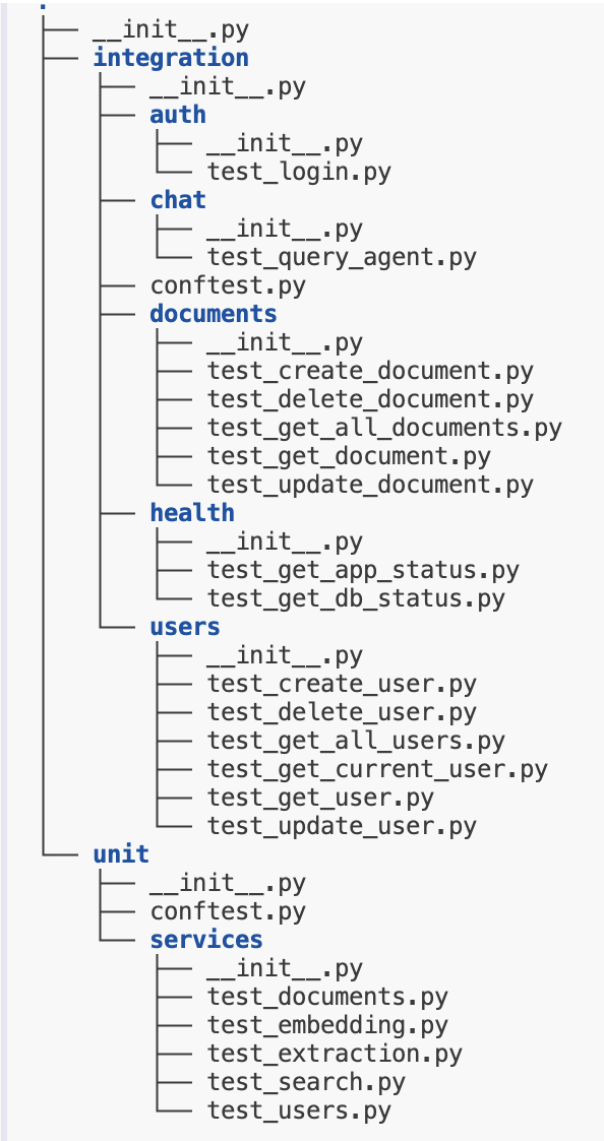
types.py

src > team_copilot > models > types.py > VectorType > bind_processor

```python
"""Team Copilot - Core - Types."""

from sqlalchemy.types import UserDefinedType

from team_copilot.core.config import settings


class VectorType(UserDefinedType):
    """PostgreSQL vector type."""

    def __init__(self, precision=settings.emb_dim):
        """Initialize the vector type.

        Args:
            precision (int): The dimension of the vector (default: "settings.emb_dim").
        """
        self.precision = precision

    def get_col_spec(self, **kwargs):
        """Get the column specification.

        The column specification is the SQL string that defines the column in
        the database. The specification is used during the table creation
        through SQLAlchemy.
        """
        return f"vector({self.precision})"

    def bind_processor(self, dialect):
        """Return a function that converts Python values to database values.

        E.g. [1, 2, 3] would be converted to "[1, 2, 3]".
        """
        return lambda value: value if value is None else str(value)

    def result_processor(self, dialect, coltype):
        """Return a function that converts database values to Python values.

        E.g. "[1, 2, 3]" would be converted to [1, 2, 3].
        """
        return lambda value: value

```

Además, en el directorio *tests* el proyecto consta de una serie de tests unitarios (donde se prueban funciones concretas del código) y de tests de integración (dónde se prueba el comportamiento de los diferentes *endpoints* de la API).

Figura 6.7: *Estructura del directorio de tests*



Los directorios principales son:

- ***integration***:
  Contiene los tests de integración del sistema, los cuales prueban el comportamiento de los diferentes endpoints de la API.
- ***unit***:
  Contiene los tests unitarios del sistema, los cuales prueban el comportamiento de funciones específicas de la API.

Por último, una de las partes más interesantes del código es la que contiene la lógica del agente LangGraph. Este agente se define como un grafo que puede hacer 2 cosas: enviar un mensaje (pregunta) al modelo LLM o llamar a una *herramienta de agente*, que en este caso sólo es una y se encarga de buscar partes de documentos que estén relacionadas con el mensaje que se quiere enviar al modelo LLM (esas partes de documentos son el *contexto* que se envía al modelo LLM).

Figura 6.8: *Agente (parte 1)*

agent.py 3

src > team_copilot > agent > agent.py > Agent > query

```python
"""Team Copilot - Agent - Agent."""

from textwrap import dedent
from typing import Annotated, Generator
from typing_extensions import TypedDict

from langchain_core.messages import SystemMessage, HumanMessage
from langgraph.graph import StateGraph, END
from langgraph.graph.message import add_messages
from langgraph.prebuilt import ToolNode, tools_condition

from team_copilot.agent.llm import get_llm
from team_copilot.agent.tools import search_docs


SYS_MSG = dedent(
    """You are a document assistant that only answers questions based on the content of
    documents stored.

    Important instructions:
    - Never explain your reasoning process.
    - Always use the search_doc tool to find information before answering.
    - Only provide information found in the documents.
    - If you cannot find the answer, say something like "I don't know" instead of making
      up an answer.
    - Never use your general knowledge or make up answers.
    - Always answer in the same language as the question whenever possible. Otherwise,
      answer in English.
    """
)

```

Figura 6.9: *Agente (parte 2)*

```python

class AgentState(TypedDict):
    """Agent state."""

    # Messages to be sent to the LLM, which will be updated by appending new messages to
    # the list instead of overwriting the entire list.
    messages: Annotated[list, add_messages]


class Agent:
    """Agent."""

    def __init__(self):
        """Initialize the agent."""

        # System message
        self.sys_msg = SYS_MSG

        # Tools
        tools = [search_docs]

        # LLM
        self.llm = get_llm().bind_tools(tools)

        # Graph builder
        builder = StateGraph(AgentState)

        # Nodes
        builder.add_node("call_llm", self.call_llm)
        builder.add_node("tools", ToolNode(tools))

        # Edges
        builder.add_conditional_edges("call_llm", tools_condition)
        builder.add_edge("tools", "call_llm")

```

Figura 6.10: *Agente (parte 3)*

```python
        # Start node
        builder.set_entry_point("call_llm")

        # Compile the graph
        self.graph = builder.compile()

    def call_llm(self, state: AgentState) -> AgentState:
        """Call the LLM.

        Args:
            state (AgentState): Agent state.

        Returns:
            AgentState: Updated agent state.
        """
        # Add the system message to the state if it's not already there
        messages = state["messages"]

        if not messages or messages[0].type != "system":
            messages = [SystemMessage(content=self.sys_msg)] + messages

        msg = self.llm.invoke(messages)
        return {"messages": [msg]}

```

Figura 6.11: *Agente (parte 4)*

```python
    def query(self, text: str) -> Generator[str, None, None]:
        """Query the agent in stream mode.

        Args:
            text (str): Query text.

        Yields:
            str: Response tokens.
        """
        # Check that the text is not null or empty
        if not text:
            raise ValueError("Query text cannot be null or empty.")

        inp = {"messages": [HumanMessage(content=text)]}

        try:
            for token, _ in self.graph.stream(inp, stream_mode="messages"):
                if (
                    token.type == "AIMessageChunk"
                    and hasattr(token, "content")
                    and token.content
                    and (token_txt := token.content[0].get("text"))
                ):
                    yield token_txt
        except Exception as e:
            if hasattr(e, "error"):
                # Anthropic's LLMs raise exceptions with a "error" attribute and a
                # "message" attribute containing the error message.
                error = getattr(e, "error")

                if hasattr(error, "message"):
                    message = getattr(error, "message")

                    if message:
                        raise Exception(f"Agent error: {message}")

            raise Exception(f"Agent error: {e}")

```

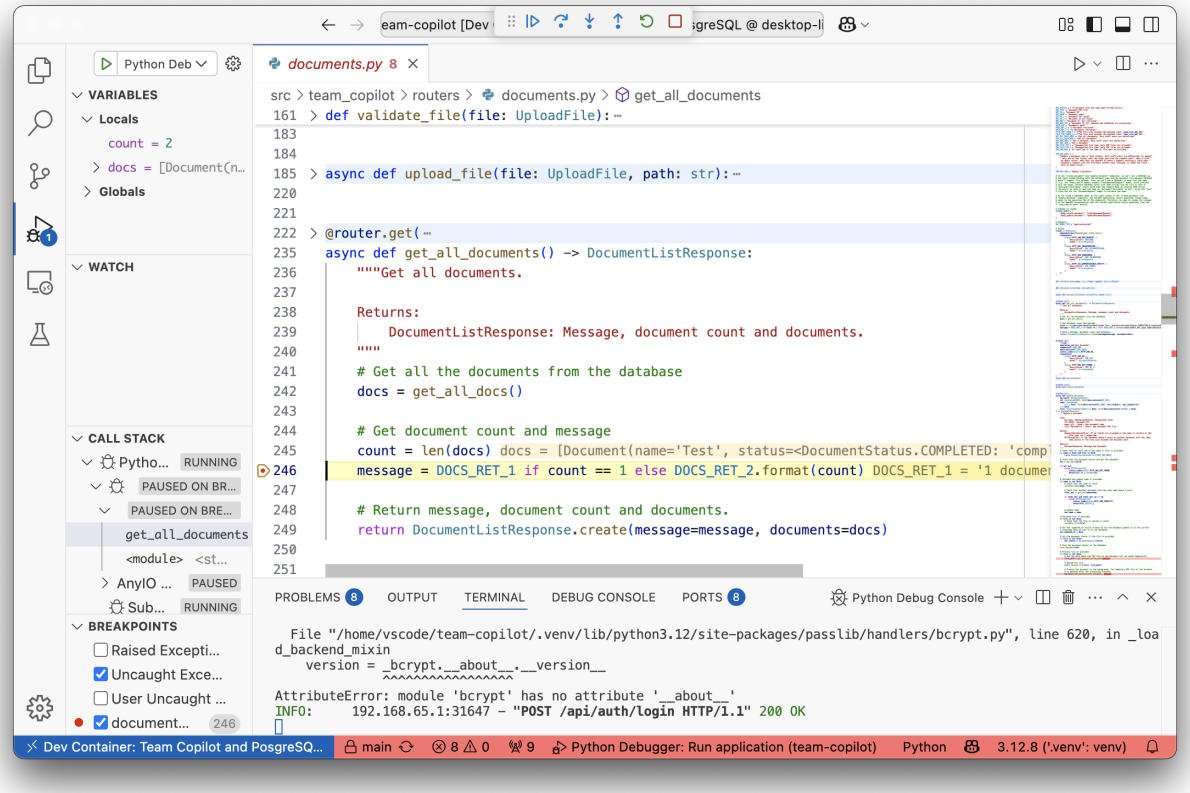
7. Pruebas del sistema

En este capítulo se describen los métodos que se han utilizado para probar que el sistema funcione correctamente. Estos métodos se han realizado en el mismo ordenador portátil donde se ha desarrollado el proyecto.

7.1. Depuración del código y detección de errores en el *back-end*

El *back-end* del proyecto se ha escrito en Python y para depurar su código y detectar posibles errores, se ha utilizado la herramienta de depuración del editor de texto Visual Studio Code, el cual hemos usado como entorno de desarrollo.

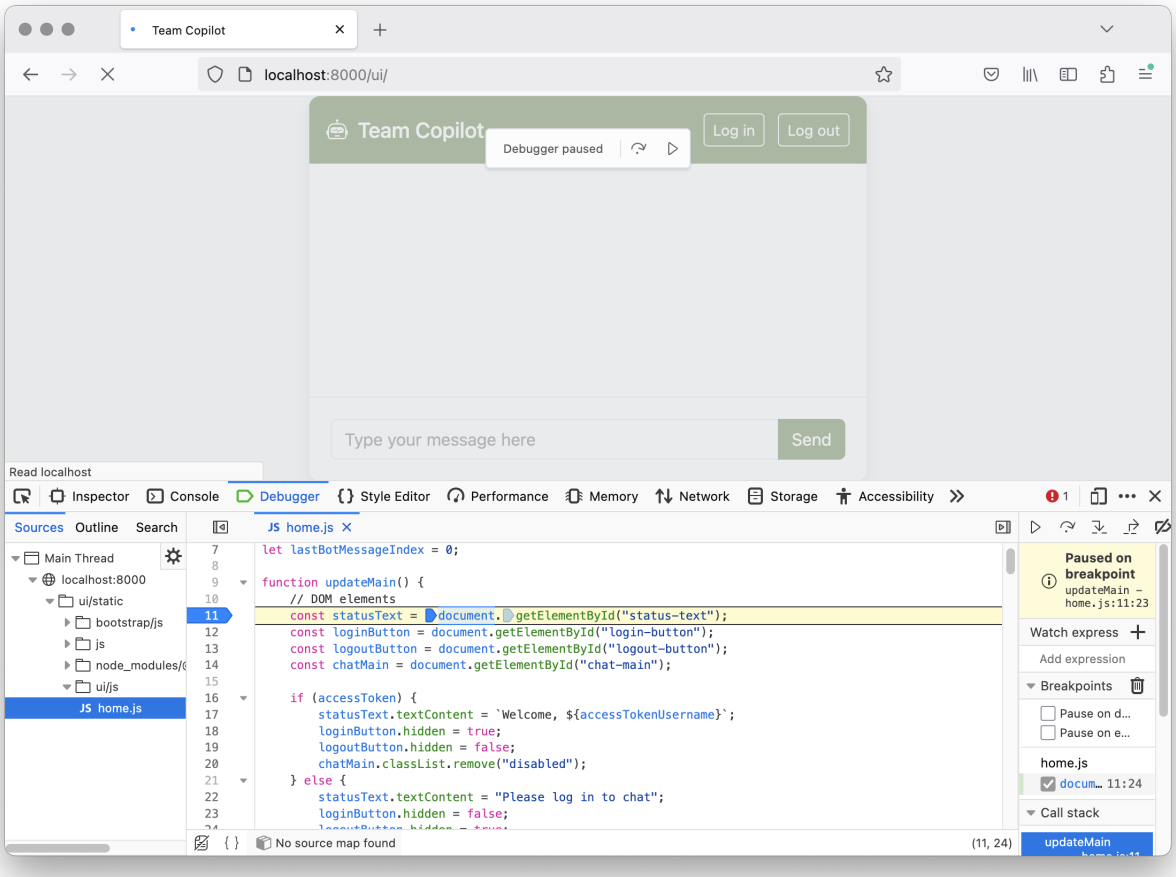
Figura 7.1: *Depuración del código del back-end en Visual Studio Code*



7.2. Depuración del código y detección de errores en el *front-end*

El *front-end* del proyecto se ha escrito en JavaScript y para depurar su código y detectar posibles errores, se han utilizado las herramientas de desarrollo del navegador web Firefox.

Figura 7.2: *Depuración del código del front-end en el navegador Firefox*



7.3. Pruebas unitarias

Para validar el funcionamiento de varias funciones y clases del proyecto, se utilizan tests unitarios en el *back-end* usando la biblioteca de Python `PyTest`. Este tipo de tests sirve para probar el comportamiento de una pieza individual de código, como una función o una clase, simulando cualquier servicio o dependencia externa, como la conexión a la base de datos.

Figura 7.3: *Ejemplo 1 de test unitario*

test_users.py 1

tests > unit > services > test_users.py > ...

```python
class TestDeleteUser:
    """Tests for the `team_copilot.services.users.delete_user` function."""

    @patch("team_copilot.services.users.open_session")
    def test_delete_user(self, mock_open_session: MagicMock, test_users: list[User]):
        """Test deleting a user.

        Args:
            mock_open_session (MagicMock): Mock object for the "open_session" function.
            test_users (list[User]): Test users.
        """
        # Get a test user
        user = test_users[0]

        # Create mock session and configure it to return our test user
        mock_session = MagicMock()
        mock_session.get.return_value = user

        # Simulate the returned value of the "open_session" function
        mock_open_session.return_value.__enter__.return_value = mock_session

        # Call the function being tested
        delete_user(user.id)

        # Check function calls
        mock_open_session.assert_called_once()
        mock_session.get.assert_called_once_with(User, user.id)
        mock_session.delete.assert_called_once_with(user)
        mock_session.commit.assert_called_once()

```

Figura 7.4: *Ejemplo 2 de test unitario*

test_documents.py 6 ×

tests > unit > services > test_documents.py > ...

```
55  class TestGetDocument:
91      @patch("team_copilot.services.documents.open_session")
92      def test_get_document_by_name(
93          self,
94          mock_open_session: MagicMock,
95          test_documents: list[Document],
96      ):
97          """Test getting a document by name.
98
99          Args:
100             mock_open_session (MagicMock): Mock object for the "open_session" function.
101             test_documents (list[Document]): Test documents.
102         """
103         # Get a test document
104         doc = test_documents[0]
105
106         # Create mock session and configure it to return the test document
107         mock_session = MagicMock()
108         mock_session.exec.return_value.first.return_value = doc
109
110         # Simulate the returned value of the "open_session" function
111         mock_open_session.return_value.__enter__.return_value = mock_session
112
113         # Call the function being tested
114         result = get_document(name=doc.name)
115
116         # Check result
117         assert result == doc
118
119         # Check function calls
120         mock_open_session.assert_called_once()
121         mock_session.exec.assert_called_once()
122         mock_session.exec.return_value.first.assert_called_once()
123
```

7.4. Pruebas de integración

Para validar el funcionamiento de los *endpoints* de la API y su flujo de datos, el proyecto también utiliza varios tests (de integración, en este caso) a través de PyTest.

Figura 7.5: *Ejemplo 1 de test de integración*

test_login.py

tests > integration > auth > test_login.py > test_login_invalid_credentials

```python
def test_login(
    mock_authenticate_user: MagicMock,
    test_client: TestClient,
    test_users: list[User],
):
    """Test the "login" endpoint.

    Args:
        mock_authenticate_user (MagicMock): Mock object for the "authenticate_user"
            function.
        test_client (TestClient): FastAPI test client.
        test_users (list[User]): Test users.
    """
    # Mock the returned value of the "authenticate_user" function
    mock_authenticate_user.return_value = test_users[0]

    # Request data
    req_data = {"username": "user", "password": "password"}

    # Make HTTP request
    response = test_client.post("/auth/login", data=req_data)

    # Check response
    assert response.status_code == status.HTTP_200_OK
    res_data = response.json()

    assert len(res_data) == 2
    assert res_data["access_token"] is not None
    assert res_data["token_type"] == "bearer"

    # Check functions call
    mock_authenticate_user.assert_called_once_with(
        req_data["username"],
        req_data["password"],
    )

```

Figura 7.6: *Ejemplo 2 de test de integración (parte 1)*

test_get_all_users.py

tests > integration > users > test_get_all_users.py > test_get_all_users

```python
@patch("team_copilot.routers.users.get_all_us")
def test_get_all_users(
    mock_get_all_us: MagicMock,
    app: FastAPI,
    test_client: TestClient,
    test_admin_user: User,
    test_users: list[User],
):
    """Test the "get_all_users" endpoint.

    Args:
        mock_get_all_us (MagicMock): Mock object for the "get_all_us" function.
        app (FastAPI): FastAPI application.
        test_client (TestClient): FastAPI test client.
        test_admin_user (User): Test enabled administrator user.
        test_users (list[User]): Test users.
    """
    # Simulate injected dependencies
    app.dependency_overrides[get_admin_user] = lambda: test_admin_user

    # Simulate the returned value of the "get_all_us" function
    mock_get_all_us.return_value = test_users

    # Make HTTP request
    response = test_client.get("/users")

    # Check response
    assert response.status_code == status.HTTP_200_OK

    user_count = len(test_users)
    res_data = response.json()

```

Figura 7.7: *Ejemplo 2 de test de integración (parte 2)*

```python
    assert len(res_data) == 3
    assert res_data["message"] == f"{user_count} users retrieved."
    assert res_data["count"] == user_count

    data = res_data["data"]
    assert len(data) == user_count

    for i, u in enumerate(data):
        assert u["id"] == str(test_users[i].id)
        assert u["username"] == test_users[i].username
        assert u["name"] == test_users[i].name
        assert u["email"] == test_users[i].email
        assert u["staff"] == test_users[i].staff
        assert u["admin"] == test_users[i].admin
        assert u["enabled"] == test_users[i].enabled
        assert parse(u["created_at"]) == test_users[i].created_at
        assert parse(u["updated_at"]) == test_users[i].updated_at

    # Check function calls
    mock_get_all_us.assert_called_once()

    # Clear simulated injected dependencies
    app.dependency_overrides.clear()

```

8. Conclusiones y trabajo futuro

Al finalizar este trabajo, he obtenido las siguientes conclusiones:

- Existen modelos de *embedding* y modelos LLM remotos que se pueden utilizar para externalizar la lógica de los modelos y separarla de la aplicación en sí.
- Si se necesita tener control total sobre los modelos utilizados, una opción es utilizar modelos de código abierto como *Llama*. Sin embargo, estos requieren de un cierto nivel de recursos hardware.
- Utilizar una base de datos vectorial tiene ciertas ventajas pero no es estrictamente necesario usar ese tipo de bases de datos cuando se quiere trabajar con documentos y modelos LLM. PostgreSQL, que es un gestor de bases de datos relaciones, puede ser configurado para soportar vectores.
- Utilizar agentes para gestionar las conversaciones otorga un mayor nivel de control sobre las conversaciones y el modelo LLM, ya que un agente define qué acciones se pueden tomar.

En general, estoy satisfecho con el resultado obtenido con este proyecto, ya que cumple con los objetivos establecidos y he podido ampliar mis conocimientos sobre las herramientas utilizadas. Como un futuro trabajo, sería una buena idea probar modelos de código abierto en local que requieran menos recursos hardware que Llama. Esto permitiría al proyecto tener control absoluto sobre los modelos, no depender de ningún servicio externo y ahorrar costes. Además, otra posible mejora sería implementar memoria en el agente de chatbot, de forma que el agente recuerde mensajes pasados de una misma conversación con un usuario en concreto.

Agradezco a Juan Antonio Buero Viana, supervisor del proyecto, su apoyo durante el desarrollo del mismo.

Bibliografía

[1] *ChatGPT*. URL: https://chat.openai.com.

[2] *Claude AI*. URL: https://claude.ai.

[3] *Gemini*. URL: https://gemini.google.com.

[4] *LangGraph*. URL: https://www.langchain.com/langgraph.

[5] *Onyx*. URL: https://github.com/onyx-dot-app/onyx.

[6] *Anything LLM*. URL: https://www.anythingllm.com/.

[7] *Private GPT*. URL: https://github.com/zylon-ai/private-gpt.

[8] *Voyage AI*. URL: https://www.voyageai.com.

[9] *Anthropic*. URL: https://www.anthropic.com.

[10] *Python*. URL: https://www.python.org.

[11] *FastAPI*. URL: https://fastapi.tiangolo.com.

[12] *Bootstrap*. URL: https://www.getbootstrap.com.

[13] *Glassdor*. URL: https://www.glassdoor.es.

[14] *AWS Price Calculator*. URL: https://calculator.aws.

[15] *MacBook Pro*. URL: https://www.apple.com/shop/buy-mac/macbook-pro/14-inch.

Índice de figuras

1.1. Interfaz de usuario de Team Copilot 5

2.1. Captura de pantalla de Onyx 7
2.2. Sitio web de Anything LLM 8
2.3. Captura de pantalla de Private GPT 9

3.1. Diagrama de Gannt del proyecto 13
3.2. Diagrama de recursos del proyecto 13

5.1. Diagrama UML del modelo de datos 45
5.2. Interfaz web de usuario 47
5.3. Interfaz web de documentación (parte 1) 47
5.4. Interfaz web de documentación (parte 2) 48
5.5. Interfaz web de documentación (parte 3) 48
5.6. Interfaz web de documentación (parte 4) 48
5.7. Interfaz web de documentación (parte 5) 49
5.8. Interfaz web de documentación (parte 6) 49
5.9. Interfaz web de documentación (parte 7) 49
5.10. Interfaz web de administración de la base de datos 50
5.11. Diagrama de flujo (parte 1) 51
5.12. Diagrama de flujo (parte 2) 51
5.13. Diagrama de flujo (parte 3) 52

6.1. Archivo Docker Compose 55
6.2. Archivo Dockerfile-db 56
6.3. Estructura del paquete Python 57
6.4. Ejemplo de controlador (get_document) (parte 1) 58
6.5. Ejemplo de controlador (get_document) (parte 2) 58
6.6. Tipo de datos VectorType 59
6.7. Estructura del directorio de tests 60
6.8. Agente (parte 1) 61
6.9. Agente (parte 2) 62
6.10. Agente (parte 3) 62
6.11. Agente (parte 4) 63

7.1. Depuración del código del back-end en Visual Studio Code 65
7.2. Depuración del código del front-end en el navegador Firefox 66
7.3. Ejemplo 1 de test unitario 67
7.4. Ejemplo 2 de test unitario 68
7.5. Ejemplo 1 de test de integración 69
7.6. Ejemplo 2 de test de integración (parte 1) 70
7.7. Ejemplo 2 de test de integración (parte 2) 70

Índice de cuadros

2.1. Comparativa de soluciones de chatbots de documentos 9

3.1. Planificación del proyecto 12
3.2. Organización de personal del proyecto 13
3.3. Hardware y software del proyecto 14
3.4. Presupuesto, coste del personal 15
3.5. Presupuesto, coste de infraestructura y software 16
3.6. Planificación del proyecto 17

4.1. Análisis de requisitos, organización Departamento de Lenguajes y Sistemas Informáticos 19
4.2. Análisis de requisitos, participante Juan Antonio Álvarez García 19
4.3. Análisis de requisitos, participante José Antonio Jiménez Carmona 19
4.4. Objetivo OBJ-0001 20
4.5. Objetivo OBJ-0002 20
4.6. Objetivo OBJ-0003 20
4.7. Objetivo OBJ-0004 21
4.8. Objetivo OBJ-0005 21
4.9. Objetivo OBJ-0006 21
4.10. Objetivo OBJ-0007 22
4.11. Objetivo OBJ-0008 22
4.12. Objetivo OBJ-0009 23
4.13. Objetivo OBJ-0010 23
4.14. Objetivo OBJ-0011 23
4.15. Objetivo OBJ-0012 24
4.16. Objetivo OBJ-0013 24
4.17. Objetivo OBJ-0014 24
4.18. Objetivo OBJ-0015 25
4.19. Requisito de información IRQ-0001 25
4.20. Requisito de información IRQ-0002 26
4.21. Requisito de información IRQ-0003 26
4.22. Requisito funcional FRQ-0001 27
4.23. Requisito funcional FRQ-0002 27
4.24. Requisito funcional FRQ-0003 28
4.25. Requisito funcional FRQ-0004 28
4.26. Requisito funcional FRQ-0005 29
4.27. Requisito funcional FRQ-0006 29
4.28. Requisito funcional FRQ-0007 30
4.29. Requisito funcional FRQ-0008 30
4.30. Requisito funcional FRQ-0009 31
4.31. Requisito no funcional NFR-0001 31
4.32. Requisito no funcional NFR-0002 32
4.33. Requisito no funcional NFR-0003 32
4.34. Requisito no funcional NFR-0004 33

4.35. Requisito no funcional NFR-0005 33
4.36. Requisito no funcional NFR-0006 34
4.37. Requisito no funcional NFR-0007 34
4.38. Requisito no funcional NFR-0008 35
4.39. Requisito no funcional NFR-0009 35
4.40. Caso de uso UC-0001 36
4.41. Caso de uso UC-0002 37
4.42. Caso de uso UC-0003 38
4.43. Caso de uso UC-0004 39
4.44. Caso de uso UC-0005 40
4.45. Caso de uso UC-0006 41
4.46. Caso de uso UC-0007 42
4.47. Caso de uso UC-0008 43
4.48. Caso de uso UC-0009 44