CARRERA DE ESPECIALIZACIÓN EN INTELIGENCIA ARTIFICIAL

MEMORIA DEL TRABAJO FINAL

Desarrollo de un chatbot especializado para optimizar la búsqueda de información en documentos propietarios

Autor:
Ing. Fabián Alejandro Massotto

Director:
Esp. Ing. Ezequiel Guinsburg (FIUBA)

Jurados:
Dr. Lic. Rodrigo Cárdenas (FIUBA)
Esp. Ing. Abraham Rodriguez (FIUBA)
Esp. Ing. Ariadna Garmendia (FIUBA)

Este trabajo fue realizado en la Ciudad Autónoma de Buenos Aires, entre abril de 2024 y abril de 2025.

Resumen

El presente trabajo aborda el desarrollo de un chatbot que interpreta consultas realizadas en lenguaje natural y ofrece respuestas precisas basadas en documentos empresariales previamente procesados. Su valor radica en la eficiencia operativa que se obtiene al optimizar el acceso a información crítica de una organización.

En esta memoria se detallan todas las etapas del desarrollo, desde la preparación de los datos hasta la evaluación del rendimiento del chatbot. Para lograrlo, se aplicaron conocimientos de procesamiento de lenguaje natural, modelos grandes de lenguaje e inteligencia artificial generativa.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción general

En este capítulo se introduce la problemática que motivó el presente trabajo, seguida de una breve descripción de la solución propuesta. A continuación, se expone el estado del arte de las tecnologías aplicadas. Finalmente, se detallan el alcance y los requerimientos necesarios para su implementación.

1.1. Marco de la propuesta

En un entorno empresarial, la eficiencia en la búsqueda de información es crucial para la productividad y el rendimiento de los empleados. Sin embargo, con la creciente cantidad de datos y documentos disponibles, encontrar información específica de manera rápida y precisa puede convertirse en un desafío.

La abundancia de fuentes de información, lejos de agilizar el trabajo, a menudo lo complica. En una empresa, es común que existan múltiples repositorios de documentos, políticas y datos históricos, pero la falta de centralización y la dificultad para identificar la fuente correcta suelen traducirse en pérdidas de tiempo significativas. En muchas ocasiones, se dedica más tiempo a la búsqueda de información que a la ejecución de las tareas en sí, lo que afecta tanto la productividad como la efectividad en la toma de decisiones.

El presente trabajo surgió como un desarrollo personal para abordar esta problemática. Su propósito fue proponer una solución que facilite el acceso ágil y preciso a la información necesaria, de manera tal que optimice el uso del tiempo en un entorno con gran cantidad de fuentes de información.

Un chatbot especializado ofrece una solución prometedora al permitir a los usuarios realizar consultas en lenguaje natural y obtener respuestas de manera instantánea. Mientras que otros sistemas de inteligencia artificial ampliamente conocidos y utilizados destacan en su capacidad para generar respuestas generales basadas en un amplio conocimiento del lenguaje, el presente trabajo se distingue por su capacidad para trabajar con documentos altamente específicos (y potencialmente privados). Esto le permite ofrecer respuestas adaptadas al contexto interno de la organización, que no podrían obtenerse mediante el uso de los chatbots de propósito general disponibles en el mercado [1] [2] [3].

En la figura 1.1 se presenta un diagrama de alto nivel de la solución. En primer lugar, los usuarios interactúan con el chatbot a través de una interfaz gráfica desde donde pueden realizar consultas sobre la información deseada. Estas consultas, procesadas mediante técnicas de lenguaje natural, permiten extraer la información más relevante de la fuente de documentos. Luego, un modelo de inteligencia

artificial interpreta las consultas y genera respuestas que proporcionan al usuario la información solicitada de manera precisa y contextualizada.

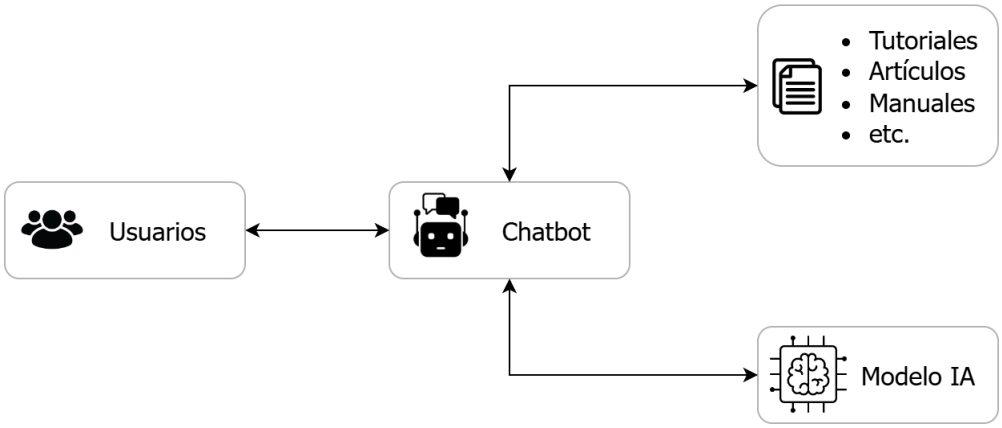

FIGURA 1.1. Diagrama de alto nivel de la solución.

1.2. Estado del arte

El desarrollo de chatbots y sistemas de recuperación de información ha avanzado considerablemente en los últimos años, impulsado por mejoras en el procesamiento de lenguaje natural (NLP, por su sigla en inglés) y el acceso a grandes volúmenes de datos. En este contexto, los chatbots especializados han surgido como soluciones destacadas para el acceso eficiente a información específica en distintos entornos, incluyendo el empresarial. A continuación, se presenta una revisión de las principales tecnologías y enfoques actuales que sustentan el desarrollo del presente trabajo.

Los chatbots modernos han evolucionado desde sistemas de reglas simples hasta modelos sofisticados capaces de mantener conversaciones complejas. Entre los primeros desarrollos, como ELIZA [4] en la década de 1960, se empleaban reglas predefinidas que limitaban la interacción a una cantidad pequeña de respuestas posibles. Sin embargo, el uso de redes neuronales y el aprendizaje profundo en las últimas décadas han revolucionado el campo, al permitir la aparición de sistemas como Siri de Apple, Alexa de Amazon y Google Assistant [5]. Estos asistentes virtuales han popularizado el uso de interfaces de conversación en la vida cotidiana, ya que son capaces de responder a preguntas comunes, realizar tareas administrativas y ofrecer asistencia en tiempo real.

Una tendencia reciente en el desarrollo de chatbots es la aplicación de modelos generativos de lenguaje, como GPT-3 y GPT-4 de OpenAI [6], BERT de Google [7], y LLAMA de Meta [8]. Estos modelos, basados en arquitecturas de *transformers* [9], permiten una comprensión profunda del contexto y del significado en secuencias de palabras. Su capacidad de generar respuestas coherentes y bien estructuradas ha llevado al desarrollo de los tan populares chatbots modernos como ChatGPT [1], Microsoft Copilot [2] o Google Gemini [3], cuyas interfaces se observan en la figura 1.2.

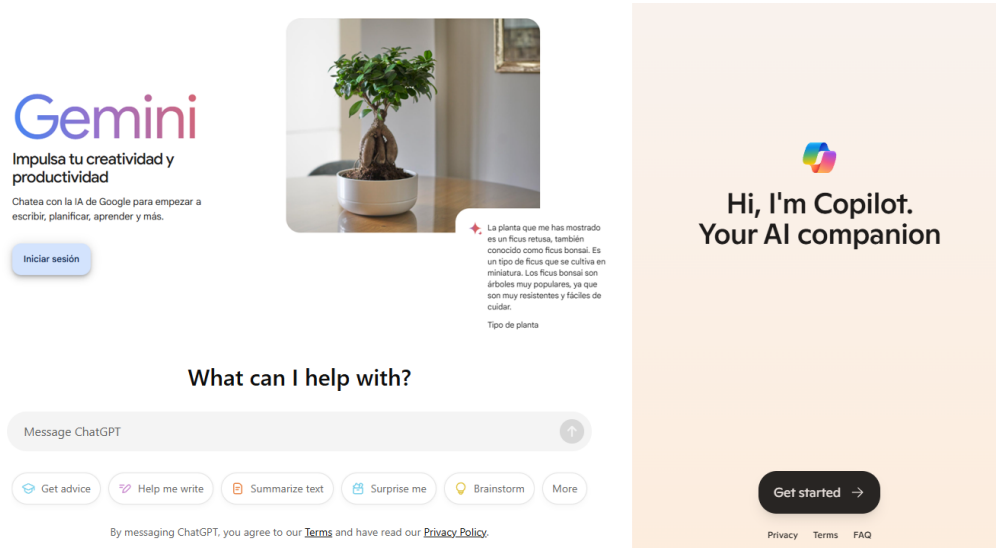

FIGURA 1.2. ChatGPT, Gemini y Copilot, los chatbots más populares actualmente.

Si bien los modelos generativos han alcanzado un alto grado de sofisticación, presentan algunas limitaciones importantes. En primer lugar, su conocimiento es en gran medida de propósito general, dado que han sido entrenados con grandes volúmenes de datos públicos y no específicos, lo que limita su precisión cuando se requiere información particular de una organización. En segundo lugar, estos modelos tienden a "inventar" respuestas cuando no encuentran información relevante, fenómeno conocido como *hallucinations* [10]. En un contexto empresarial, esto puede provocar confusión o incluso proporcionar información errónea.

En la búsqueda de soluciones que combinen la capacidad de los modelos generativos con la precisión de la información propietaria, ha surgido el enfoque de generación aumentada por recuperación (RAG, por su sigla en inglés). Este enfoque combina sistemas de recuperación de información con modelos de generación de texto, lo que permite que las respuestas no solo se basen en la capacidad generativa del modelo, sino también en una búsqueda previa en bases de datos o documentos específicos [11] [12].

El presente trabajo se apoya en el estado del arte de los modelos de lenguaje y la técnica de RAG para crear una solución innovadora.

1.3. Objetivos y alcance

El propósito de este trabajo fue optimizar el proceso de búsqueda de información por parte de los empleados. Se buscó proporcionar una herramienta eficaz que permita acceder rápidamente a los datos relevantes, que mejore la eficiencia y productividad en el entorno laboral.

Para ello, se realizaron las siguientes tareas:

- Procesamiento de los documentos y posterior almacenamiento en una base de datos.
- Integración con un modelo lingüístico grande (LLM) que pueda entender las consultas de los usuarios y proporcionar respuestas precisas basadas en el contenido de los documentos ingestados.
- Diseño e implementación de una interfaz de usuario intuitiva y fácil de utilizar que permita a los empleados interactuar con el chatbot de manera eficiente.
- Desarrollo de un *pipeline* de despliegue continuo que facilite la ingesta de nuevos documentos y la actualización de la aplicación.
- Evaluación del rendimiento del chatbot mediante pruebas exhaustivas con diferentes tipos de consultas.

Las siguientes actividades no formaron parte del alcance:

- Despliegue del chatbot en un ambiente productivo.
- Entrenamiento continuo del chatbot en base a las consultas realizadas por los usuarios.
- Desarrollo de funcionalidades avanzadas de seguridad, tales como autenticación de usuarios o cifrado de datos.

1.4. Requerimientos

A continuación se describen los principales requerimientos establecidos durante la etapa de planificación:

1. Requerimientos funcionales:
   a) El sistema debe permitir a los usuarios consultar por información a través de una interfaz gráfica.
   b) El sistema debe ser capaz de entender consultas escritas en lenguaje natural.
   c) El sistema debe proporcionar respuestas precisas basadas en el contenido de los documentos procesados.
2. Requerimientos de la interfaz:
   a) La interfaz gráfica debe ser intuitiva y fácil de usar para los usuarios.
   b) Se debe proporcionar retroalimentación instantánea al usuario luego de realizar una consulta.
3. Requerimiento de testing:
   a) Se deben realizar pruebas exhaustivas para garantizar la precisión y la robustez del sistema.

4. Requerimientos de documentación:
    a) Se deben documentar las pruebas realizadas y los resultados obtenidos.
    b) Se debe elaborar un informe de avance del proyecto.
    c) Se debe confeccionar una memoria técnica del proyecto.
5. Requerimientos de cumplimiento normativo:
    a) El sistema debe cumplir con las regulaciones de privacidad de datos vigentes.

Capítulo 2

Introducción específica

Este capítulo tiene como objetivo presentar las técnicas y herramientas clave utilizadas en el desarrollo de este trabajo. Se analizan los enfoques fundamentales para el procesamiento de lenguaje natural, junto con los modelos, *frameworks* e infraestructura necesarios para construir un sistema de recuperación de información eficiente y escalable. Este recorrido técnico permite comprender los fundamentos sobre los que se apoya la solución implementada.

2.1. Técnicas de procesamiento de lenguaje natural

El procesamiento de lenguaje natural (NLP) es un área de la inteligencia artificial que permite a las máquinas comprender e interpretar el lenguaje humano [13]. Esto juega un papel esencial al permitir que el chatbot interprete las consultas de los usuarios y localice la información relevante en los documentos procesados. Las técnicas de NLP aplicadas permiten que el sistema entienda el significado de las consultas, independientemente de variaciones lingüísticas o de sintaxis. Entre los métodos de mayor importancia para este trabajo se encuentran los *word embeddings* y *transformers*.

2.1.1. Word embeddings

Los *embeddings* son una poderosa técnica para transformar datos complejos en formas numéricas que son fácilmente procesadas y analizadas por algoritmos de aprendizaje automático. Esta técnica permite representar virtualmente cualquier tipo de dato como vectores, lo que posibilita su manipulación en tareas de procesamiento de lenguaje natural.

Sin embargo, no se trata solo de convertir las palabras en vectores. Es crucial preservar el significado original de los datos para que las tareas realizadas en este espacio transformado mantengan la coherencia semántica. Por ejemplo, al comparar dos frases, no queremos simplemente analizar las palabras que contienen, sino evaluar si ambas expresan un significado similar.

Para conservar el significado, es necesario generar vectores donde las relaciones entre ellos sean representativas del contenido. Para ello, se emplea un modelo de *embeddings* pre-entrenado, que produce una representación compacta de los datos mientras mantiene sus características semánticas. El objetivo es capturar el significado o las relaciones semánticas entre los puntos de datos, de modo que los elementos similares se encuentren cercanos en el espacio vectorial y los disímiles estén alejados. Por ejemplo, consideremos las palabras “rey” y “reina”. Un *embedding* puede mapear estas palabras en vectores de modo que la diferencia entre

"rey' y "reina" sea similar a la diferencia entre "hombre" y "mujer", reflejando así las relaciones semánticas subyacentes.

En este trabajo, se ensayaron diferentes modelos de *embeddings* de OpenAI, Google y Hugging Face, cuya evaluación y selección se detallan en el capítulo 4.

2.1.2. Transformers

La arquitectura de *transformers* ha revolucionado el procesamiento de lenguaje natural al introducir un mecanismo de *self-attention*, que permite a un modelo evaluar la relevancia de cada palabra en una secuencia en relación con las demás [9]. Este enfoque supera las limitaciones de modelos secuenciales tradicionales, ya que permite procesar palabras en paralelo y captar dependencias de largo alcance en el texto. En lugar de analizar cada palabra en un orden específico, el mecanismo de *self-attention* permite que el modelo "preste atención" a las palabras más relevantes en el contexto de la frase, al asignar pesos a cada palabra según su importancia relativa en la oración.

Gracias a esta capacidad, los *transformers* pueden capturar relaciones contextuales complejas y matices semánticos entre las palabras, lo que resulta fundamental para tareas como la generación de texto. Esta arquitectura ha hecho posible que los modelos comprendan y generen lenguaje natural de una forma mucho más cercana a la comprensión humana. Esta técnica ha sido fundamental en el desarrollo de los modelos grandes de lenguaje, que se introducen a continuación.

2.2. Modelos grandes de lenguaje

Los modelos grandes de lenguaje (LLM, por su sigla en inglés) son redes neuronales de gran escala, entrenadas para comprender y generar texto en lenguaje natural. Estos modelos, basados en arquitecturas de *transformers*, están compuestos por millones o incluso billones de parámetros, lo que les permite capturar patrones complejos y relaciones contextuales en vastos conjuntos de datos de texto. Los LLM son capaces de realizar múltiples tareas de procesamiento de lenguaje natural, como la generación de texto, la traducción automática y el resumen de documentos. En la tabla 2.1 se presentan algunos de los modelos más relevantes en la actualidad [14].

TABLA 2.1. Modelos LLM disponibles en el mercado

Modelo	Creador	Año de publicación	Cant. de parámetros (aprox.)
GPT-4o	OpenAI	2024	200 mil millones
Gemini 1.5	Google	2024	1.5 billones
LLaMa 3	Meta	2024	70 mil millones
GPT-4	OpenAI	2023	1.76 billones
LLaMa 2	Meta	2023	70 mil millones
Mistral-7B	Mistral AI	2023	7 mil millones
BLOOM	BigScience	2022	176 mil millones
GPT-3.5	OpenAI	2022	20 mil millones

En este trabajo, el modelo LLM desempeña un papel central ya que es el encargado de entender las consultas de los usuarios y generar respuestas coherentes. Para seleccionar el modelo más adecuado, se ensayaron diferentes variantes, cuyos detalles se describen en el capítulo 4.

2.2.1. Modelos LLM versus modelos tradicionales

A continuación se mencionan las principales diferencias entre los modelos grandes de lenguaje y los modelos tradicionales:

- Los LLM pueden adaptarse a nuevas tareas simplemente utilizando ejemplos en el *prompt*, sin necesidad de modificar sus parámetros o re-entrenar. Esto contrasta con los modelos tradicionales, que requieren grandes conjuntos de datos etiquetados y re-entrenarse cada vez que se incorporan nuevas tareas.
- Los LLM comprenden instrucciones en lenguaje natural, lo que les permite seguir instrucciones complejas e incluso inferir reglas implícitas a partir de ejemplos. Los modelos tradicionales, en cambio, suelen necesitar formatos de entrada específicos y siguen reglas programadas explícitamente, sin capacidad para inferir patrones nuevos en tiempo real.
- Un solo LLM es capaz de realizar múltiples tareas sin configuración adicional y de adaptar su salida según el contexto. Por el contrario, los modelos tradicionales están diseñados para una tarea específica, por lo que se requiere un modelo separado para cada tarea.
- Los LLM mantienen la coherencia en conversaciones largas, siendo capaces de recordar información dada en partes previas del diálogo. Los modelos tradicionales procesan cada entrada de manera independiente y no pueden conservar el estado conversacional.
- Los LLM permiten una interacción bidireccional, donde pueden solicitar aclaraciones y ajustar sus respuestas en función del *feedback* recibido. Los modelos tradicionales, en cambio, ofrecen una interacción unidireccional y no manejan *feedback* en tiempo real.

2.3. Generación aumentada por recuperación

Una limitación importante que presentan los LLM es que, debido a su naturaleza generalista, pueden carecer de precisión cuando se aplican a dominios específicos, ya que su conocimiento está limitado a la información con la que fueron entrenados. Si bien esta información es sumamente vasta, no alcanza a cubrir la infinidad de temas posibles, y no contempla contenido no disponible en fuentes públicas.

Para abordar esta limitación, los LLM pueden adaptarse mediante la técnica conocida como recuperación aumentada por generación (RAG), integrando fuentes de datos especializadas que mejoran su precisión en contextos concretos, como el entorno empresarial. De esta manera, los LLM pueden aprovechar su capacidad de comprensión profunda del lenguaje, pero también acceder a información precisa y actualizada de documentos específicos.

Básicamente, esta técnica consta de dos fases principales: primero, se realiza una búsqueda entre los documentos provistos y se seleccionan fragmentos que sean relevantes para la consulta del usuario. Luego, un modelo LLM utiliza esa información para construir una respuesta contextualizada. En la figura 2.1 se ilustra un diagrama básico de funcionamiento.

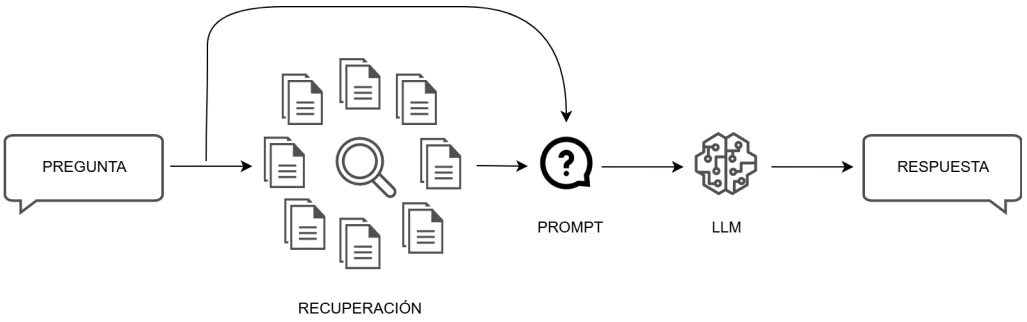

FIGURA 2.1. Diagrama de un sistema RAG.

La combinación de recuperación y generación permite un flujo de información enriquecido, al lograr un equilibrio entre las capacidades generales del modelo y la especificidad requerida en el contexto empresarial.

2.4. Bases de datos vectoriales

Una base de datos de vectorial es una base de datos diseñada para almacenar y gestionar *embeddings*. La misma cumple un papel fundamental en la etapa de recuperación de información.

A diferencia de las bases de datos tradicionales, en las que las consultas suelen coincidir exactamente con los valores almacenados, las bases de datos vectoriales aplican métricas de similitud para identificar el vector más cercano a la consulta. El proceso de recuperación se realiza calculando la distancia entre el vector de la consulta y los vectores almacenados en la base de datos. Los fragmentos con menor distancia al vector de la consulta son seleccionados como los más relevantes y luego utilizados como insumo en la fase de generación de la respuesta.

Para seleccionar la base de datos vectorial más adecuada para este trabajo, se exploraron varias opciones disponibles en el mercado, evaluando aspectos como el rendimiento y la facilidad de integración con el chatbot. Los resultados de estos ensayos se describen en el capítulo 4.

2.5. Frameworks utilizados

En el desarrollo de este trabajo se utilizaron una serie *frameworks* que facilitaron la implementación de cada parte del sistema. A continuación, se detallan sus principales características y su rol específico en el producto final.

- **LangChain:** este *framework* fue fundamental para la construcción del sistema RAG, ya que permite una gestión eficiente de flujos de trabajo con modelos de lenguaje. LangChain facilita la integración de modelos LLM con fuentes de datos externas, y además ofrece herramientas para diseñar flujos

complejos que manejan las consultas del usuario, el proceso de recuperación y la generación de respuestas [15].

- **FastAPI:** se utilizó para construir la API que conecta los componentes del sistema y gestiona las solicitudes del usuario. Este *framework*, reconocido por su alto rendimiento y facilidad de uso, permitió desarrollar una API eficiente y escalable que procesa las consultas en tiempo real. FastAPI facilita el manejo de solicitudes asíncronas y permite definir de manera clara los puntos de conexión de la API, asegurando que las interacciones entre el *frontend* y el *backend* se realicen de forma fluida y rápida [16].
- **NextUI:** es un *framework* de diseño de interfaces de usuario basado en React que permite construir interfaces modernas y responsivas. NextUI facilitó la creación de una interfaz intuitiva y fácil de usar, que permite a los usuarios interactuar con el chatbot de manera fluida. Este *framework* ofrece una variedad de componentes pre-construidos y altamente personalizables, lo que ayudó a reducir el tiempo de desarrollo de la interfaz y asegurar una experiencia de usuario satisfactoria [17].

2.6. Servicios en la nube utilizados

En este trabajo se empleó Microsoft Azure [18] como la plataforma principal de infraestructura en la nube, seleccionada por su robustez y variedad de servicios orientados a aplicaciones de inteligencia artificial. Azure ofrece soluciones escalables y seguras para gestionar datos y modelos, facilitando la integración de distintos recursos en un entorno controlado y eficiente.

Los recursos específicos de Azure utilizados incluyen:

- **Azure OpenAI:** este recurso se utilizó para desplegar el modelo de lenguaje LLM y el modelo de *embeddings*, lo que provee la capacidad de procesamiento de lenguaje natural avanzada requerida por el sistema [19].
- **Azure AI Search:** utilizado como base de datos vectorial, este servicio permite almacenar y recuperar *embeddings* mediante búsquedas de similitud [20].
- **Azure App Service:** empleado para hospedar el *backend* del chatbot, este servicio garantiza un entorno seguro y escalable para la ejecución de la API y la gestión de consultas de los usuarios [21].
- **Azure Static Web Apps:** utilizado para hospedar el *frontend* del chatbot, proporciona un despliegue rápido y eficiente de la interfaz de usuario y puede ser accedido por los empleados desde cualquier ubicación [22].

Además, se implementó GitHub Actions [23] como *pipeline* de integración y despliegue continuo, no solo para publicar actualizaciones del *backend* y el *frontend*, sino también para automatizar la actualización del contenido de la base de datos.

Capítulo 3

Diseño e implementación

Este capítulo describe el diseño y la implementación de cada uno de los componentes que conforman el sistema. Se explican las decisiones de diseño, los flujos de trabajo y los aspectos técnicos involucrados en la construcción de cada módulo principal.

3.1. Arquitectura del sistema

El sistema está diseñado para permitir la implementación y operación de un chatbot mediante un enfoque de recuperación aumentada por generación. Se implementó una arquitectura modular, basada en servicios en la nube, que facilita la escalabilidad y el mantenimiento del chatbot, y permite una actualización ágil de los componentes y una experiencia optimizada para los usuarios finales. En la figura 3.1 se ilustra el diagrama de arquitectura, donde se observan la totalidad de los componentes que conforman el sistema.

En primer lugar, los usuarios interactúan con el chatbot a través de una interfaz gráfica (desarrollada con NextUI y deployada en Azure Static Web App), donde consultan por la información deseada. Estas consultas son luego transferidas al servidor (Azure App Service) a través de una API desarrollada con la librería FastAPI. Una vez allí, se realiza un proceso de búsqueda por similitud que toma la consulta del usuario y la compara con la información disponible en la base de datos (Azure AI Search), con el objetivo de identificar aquellos fragmentos más relevantes. Previamente, un administrador debe haber cargado aquellos documentos que conforman la base de conocimiento del chatbot en un repositorio de GitHub, tras lo cual se ejecuta la automatización que procesa los documentos y los envía a la base de datos.

Para alimentar esta base de conocimiento, un administrador carga los documentos en un repositorio de GitHub, donde una automatización (desarrollada utilizando GitHub Actions) procesa los documentos y los transfiere a la base de datos.

Finalmente, la consulta del usuario y los fragmentos relevantes se envían al modelo LLM (desplegado en el servicio Azure OpenAI), que genera una respuesta contextualizada. Esta respuesta es devuelta a la interfaz gráfica para ser presentada al usuario.

Además, se incluye una pequeña base de datos adicional (Azure Table Storage) que se utiliza para guardar el *feedback* de los usuarios, para así obtener métricas del desempeño del chatbot.

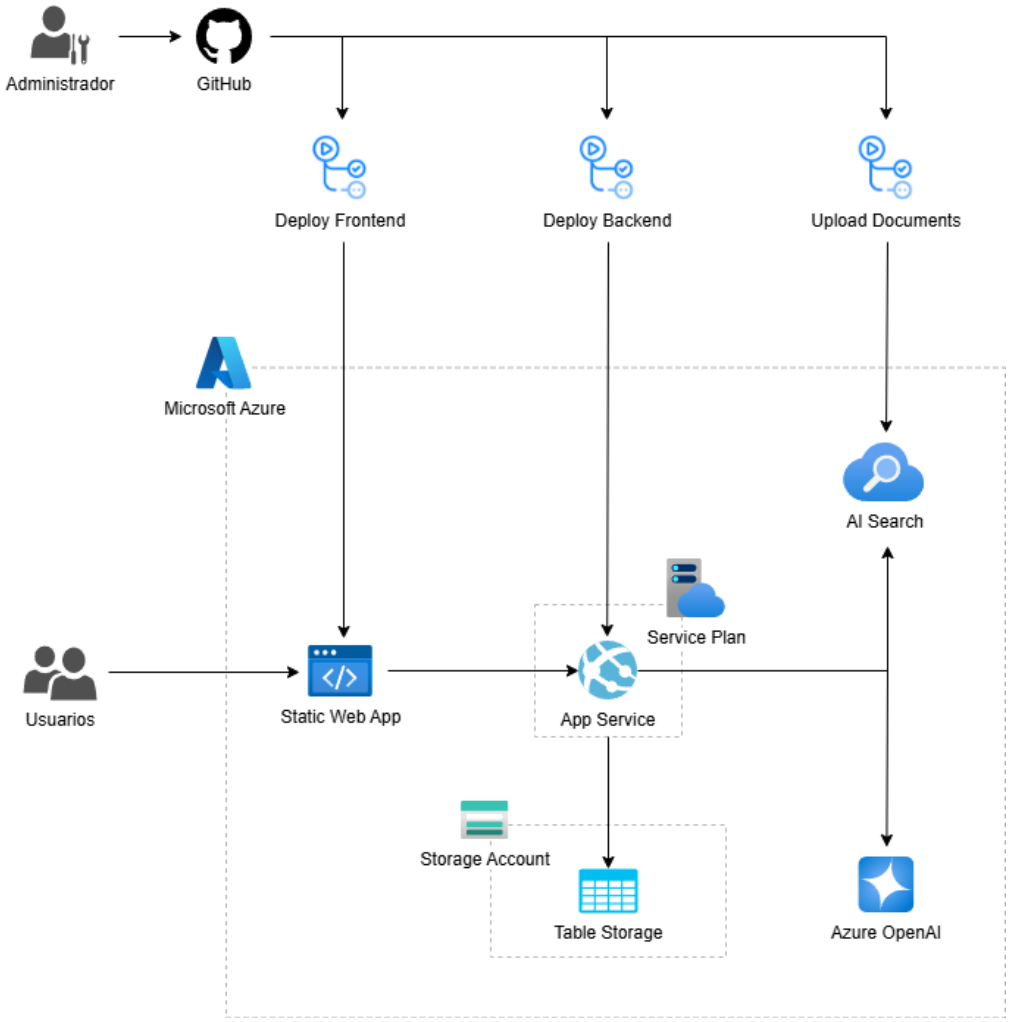

FIGURA 3.1. Diagrama de la arquitectura del sistema.

3.2. Configuración de la infraestructura en la nube

Para garantizar el despliegue, disponibilidad y escalabilidad del sistema, se configuró una infraestructura en la nube basada en Microsoft Azure. A continuación, se describen los pasos principales para la configuración de cada recurso empleado en el proyecto.

3.2.1. OpenAI Service

El servicio de Azure OpenAI proporciona acceso a los potentes modelos de lenguaje de OpenAI, incluidos los modelos más recientes. Estos modelos pueden adaptarse fácilmente a tareas específicas, como en nuestro caso es la generación de contenido.

En la tabla 3.1 se presenta un resumen de las configuraciones realizadas. Se seleccionó *East US* como región, junto con el modelo de precios *Standard S0*, adecuado para balancear el costo y el rendimiento del sistema.

Como medida de seguridad, se configuraron reglas de red para restringir el acceso únicamente al rango de direcciones IP del *backend* alojado en Azure App Service. Esto asegura que solo las solicitudes provenientes de la aplicación puedan acceder al modelo de lenguaje, lo que protege el servicio de accesos no autorizados.

Una vez desplegado el recurso, fue necesario seleccionar y desplegar los modelos específicos requeridos para la funcionalidad del chatbot. En este caso, se optó por los siguientes modelos:

- *GPT-4o* como modelo de lenguaje para generación de respuestas. Este modelo está diseñado para brindar velocidad y eficiencia, iguala la inteligencia de su antecesor *GPT-4 Turbo*, y es notablemente más eficiente al entregar texto al doble de velocidad y a la mitad del costo. Además, exhibe el rendimiento más alto en idiomas distintos del inglés en comparación con los modelos de OpenAI anteriores.
- *Ada-002* para el cálculo de *embeddings* de texto. Este modelo supera a todos los modelos de *embeddings* anteriores en tareas de búsqueda de texto y similitud de oraciones.

Adicionalmente, es fundamental obtener el *endpoint* y la *key* del servicio, valores que se utilizan luego para la comunicación programática con Azure OpenAI. Estos datos se almacenan en el *backend* como variables de entorno para que el sistema pueda acceder al servicio de manera segura.

TABLA 3.1. Configuración de Azure OpenAI

Configuración	**Detalles**
Región	East US
Nivel de precios	Standard S0
Reglas de Firewall	Rango IP de App Service
Modelos desplegados	gpt-4o text-embeddings-ada-002
Credenciales	Endpoint y Key

3.2.2. AI Search

3.2.3. App Service

3.2.4. Static Web App

3.2.5. Table Storage

3.3. Procesamiento de los documentos

3.4. Lógica de comunicación entre el usuario y el modelo

3.5. API

3.6. Interfaz de usuario

3.7. Pipelines de despliegue automático

Capítulo 4

Ensayos y resultados

Todos los capítulos deben comenzar con un breve párrafo introductorio que indique cuál es el contenido que se encontrará al leerlo. La redacción sobre el contenido de la memoria debe hacerse en presente y todo lo referido al proyecto en pasado, siempre de modo impersonal.

4.1. Ensayo de modelos

4.2. Ensayo de embeddings

4.3. Ensayo de bases de datos

4.4. Casos de uso

4.5. Validación de requerimientos

Capítulo 5

Conclusiones

Todos los capítulos deben comenzar con un breve párrafo introductorio que indique cuál es el contenido que se encontrará al leerlo. La redacción sobre el contenido de la memoria debe hacerse en presente y todo lo referido al proyecto en pasado, siempre de modo impersonal.

5.1. Resultados

5.2. Trabajo futuro

Bibliografía

[1] OpenAI. *ChatGPT*. URL: https://chatgpt.com/.
[2] Microsoft. *Copilot*. URL: https://copilot.microsoft.com/.
[3] Google. *Gemini*. URL: https://gemini.google.com/.
[4] Joseph Weizenbaum. *ELIZA — a computer program for the study of natural language communication between man and machine*. Ene. de 1966. DOI: 10.1145/365153.365168.
[5] Matthew B. Hoy. *Alexa, Siri, Cortana, and More: An Introduction to Voice Assistants*. Ene. de 2018. DOI: 10.1080/02763869.2018.1404391.
[6] Alec Radford y col. *Improving Language Understanding by Generative Pre-Training*. Jun. de 2018. URL: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.
[7] Jacob Devlin y col. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. Oct. de 2018. DOI: 10.48550/arXiv.1810.04805.
[8] Hugo Touvron y col. *LLaMA: Open and Efficient Foundation Language Models*. Feb. de 2023. DOI: 10.48550/arXiv.2302.13971.
[9] Ashish Vaswani y col. *Attention Is All You Need*. Jun. de 2017. DOI: 10.48550/arXiv.1706.03762.
[10] Rahul Awati. *What are AI hallucinations and why are they a problem?* URL: https://www.techtarget.com/whatis/definition/AI-hallucination.
[11] Patrick Lewis y col. *Retrieval-augmented generation for knowledge-intensive NLP tasks*. Dic. de 2020. DOI: 10.48550/arXiv.2005.11401.
[12] Kurt Shuster y col. *Retrieval Augmentation Reduces Hallucination in Conversation*. Abr. de 2021. DOI: 10.48550/arXiv.2104.07567.
[13] Nitin Indurkhya y Fred J. Damerau. *Handbook of Natural Language Processing*. Chapman y Hall, 2010.
[14] LifeArchitect.ai. *Models Table*. URL: https://lifearchitect.ai/models-table/.
[15] LangChain. URL: https://python.langchain.com/docs/introduction/.
[16] FastAPI. URL: https://fastapi.tiangolo.com/.
[17] NextUI. URL: https://nextui.org/docs/guide/introduction/.
[18] Microsoft. *Microsoft Azure*. URL: https://azure.microsoft.com/.
[19] Microsoft. *Azure OpenAI Service*. URL: https://azure.microsoft.com/en-us/products/ai-services/openai-service.
[20] Microsoft. *Azure AI Search*. URL: https://azure.microsoft.com/en-us/products/ai-services/ai-search.
[21] Microsoft. *Azure App Service*. URL: https://azure.microsoft.com/en-us/products/app-service.
[22] Microsoft. *Azure Static Web Apps*. URL: https://azure.microsoft.com/en-us/products/app-service/static.
[23] GitHub. *GitHub Actions*. URL: https://github.com/features/actions.